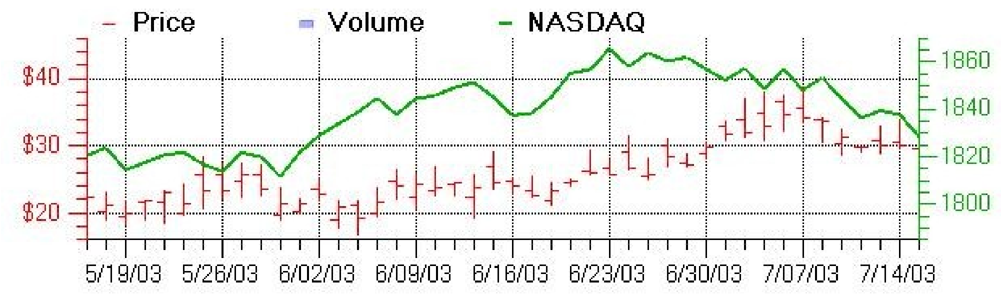
QCChart2D Charting Tools for JavaScript/TypeScript

Quinn-Curtis Software

www.quinn-curtis.com



"On inspecting any one of these Charts attentively, a sufficiently distinct impression will be made, to remain unimparied for a considerable time, and the idea which does remain will be simple and complete, at once including the duration and the amount."

William Playfair, the father of statistical graphics, in 1786

Contact Information
Company Web Site: **http://**www.quinn-curtis.com

Electronic mail

General Information: info@quinn-curtis.com
Sales: sales@quinn-curtis.com

Technical Support Forum

http://www.quinn-curtis.com/ForumFrame.htm

Revision Date 6/21/2020 Rev. 3.0

QCChart2D for JavaScript/TypeScript

QCChart2D for JavasScript/TypeScript Charting Library

1. Introduction

QCChart2D for JavaScript/TypeScript (JSTS-CHT-DEVR)

The **QCChart2D for JavaScript/TypeScript** software represents an adaptation of the **QCChart2D** library to the **JavaScript** and the HTML5 user interface framework. The entire QCChart2D library is written using the TypeScript language, which is a superset of JavaScript. The resulting TypeScript code is transpiled into JavaScript, resulting in a pure JavaScript version of the library. The library can be called from JavaScript within a browser, from JavaScript files external to the browser, or by TypeScript files which have been transpiled into their JavaScript equivalents. The result is an easy to use, interactive, Charting package which will run on any computer which supports a modern browser. A modern browser is considered any browser which supports HTML5, and most importantly, supports the HTML5 Canvas element. The browsers listed below meet this requirement.

IE (9+)
Firefox (1.5+)
Safari (1.3+)
Chrome
Opera (9+)

While IE 8 (Internet Explorer) supports a limited subset of HTML5 features, it does not support the Canvas element to the degree required by this software, and therefore is considered incompatible.

Transpiling vs Compiling

We tend to use the terms *transpiling* and *compiling* interchangeably throughout the manual. It refers to the action whereby a TypeScript file is translated into a functionally equivalent JavaScript file. You will find that others also use the terms interchangeably when dealing with TypeScript too. Transpiling is generally used to refer to the translation between two computer languages which have a similar level of abstraction: translating Java to C++ for example. Or C++ to C#. Compiling is the conversion of one language to another where the target language has a lower level of abstraction than the source language: C# compiles to the .Net Intermediate language (IL), and C++ compiles to C, and C compiles to machine code.

You can argue that TypeScript has a higher lever of abstraction than JavaScript, since it supports object oriented classes, and strong type checking. So the conversion of TypeScript to JavaScript can be called compiling. But TypeScript is also considered a superset of JavaScript, and both TypeScript and JavaScript are considered high level languages, so the action of translating TypeScript into JavaScript can also be considered transpiling. So it seems that the translation of TypeScript to JavaScript is in between compiling and transpiling. So in this instance, compiling and transpiling are both accurate.

Tutorials

Chapter 24 is a tutorial that describes how to define a simple chart and deploy it to a web page. Read the first couple of chapters, and then the tutorial.

HTML

Originally, web pages were static. The purpose of a web browser was to display a static HTML document. The static limitation quickly ran its course and users wanted more and more direct interaction with a web page, analogous to interacting with an application program installed on on a desktop computer. So JavaScript was invented to control elements of the web page, dynamically serve up documents, and asynchronously communicating with servers in support user-driven content. Originally meant as a client-side programming language, to be run in a web browser, its role has been expanded to include include desktop, cloud, and server applications.

HTML is a text-based standard format for the display of text and data by a web browser. JavaScript can automate elements within HTML to render content to the browser. And HTML elements can call JavaScript code in order to process content requests.

HTML5

HTML underwent a major revision upgrade with the introduction of a preliminary HTML5 specification around 2008. HTML5 represents an attempt to integrate many of the features required for the cross platform support of current, and next generation desktop, mobile and cloud applications. It specifically adds elements for the support of audio, video, and vector-based device independent graphics applications. Since 2014 it is a published standard which all of the major browsers support: https://en.wikipedia.org/wiki/HTML5. Because HTML5 is expected to play an critical role in the future of the Internet, the major web browsers have already adopted most all of the features set forth in the working drafts of the specification.

According to the late Steve Jobs, the future of web programming is HTML5. This comment was prompted in an interview with Jobs about his ongoing feud with Adobe and their ubiquitous Flash player plug-in. Even though the Flash player had the dominant market share as means of displaying audio and video in a web browser, Jobs refused to permit support for Flash in the Apple iOS mobile operating systems. His stated reason was that HTML5, with its extensive support for audio and video, renders Flash technology obsolete. Why install a separate plug-in (Flash), when a modern browser with integrated HTML5 support offers vastly superior routines for rendering audio and video. The way to get at all of the features of HTML5 is to use JavaScript. JavaScript and HTML5 are now supported on all major browsers, running on all major operating systems, for both desktop and mobile platforms. Sounds ideal. You might think that all you have to do is use HTML5 and JavaScript in your web page, and it will work flawlessly in every case. But you would be wrong. HTML5 implementations are left up to the web browser companies, rather than a central controlling authority, and because of this, they all differ. Developers programming directly in JavaScript have to become familiar with the differences in the HTML5 support across browsers. In their JavaScript code they must detect which browser is executing the JavaScript code, and branch to code specific to that browser. For an advanced application, this can be very tedious and time consuming.

JavaScript

JavaScript is an interpreted programming language created to make web browsers more dynamic and interactive. It was originally developed by Netscape in 1995 as a feature of their web browser. While it includes "Java" as the root of its name, that was an unfortunate marketing gimmick. Netscape picked the name solely to ride the coat-tails of the hype being written about the Java programming language introduced at approximately the same time. The only resemblance JavaScript has to the Java programming language is a small degree of syntactical similarity, mostly the result of both languages being strongly influenced by C++ (for Java) and C (for JavaScript). Since that time, all of the major browsers have added support for JavaScript.

TypeScript as a Productivity Tool

The conversion of our QCChart2D software to JavaScript and HTML5 posed significant challenges. While JavaScript has some object-oriented features, it is not a true object-oriented language. So adapting software written in an OOP language (C#, Java, and C++, among others) to JavaScript, is a giant step backwards. Originally, in 2014, in order to create a JavaScript version of our QCSPCChart software, we utilized the Google GWT development framework to allow us to maintain the source code of the software in object-oriented Java, while transpiling into JavaScript. But this was not optimal, because due to name mangling, the resulting library was not programmable directly from JavaScript. Instead we had to use JSON templates to define the SPC charts.

During the last 5 years, an open-source, Microsoft developed language, TypeScript, has become increasingly popular as a means of creating JavaScript files. TypeScript is considered a superset of JavaScript. It adds true object-oriented classes, and strong type checking to JavaScript, critical for developing large applications. It also supports source level debugging of both TypeScript source code, and the transpiled JavaScript source code. We encourage you read about TypeScript on the web : https://www.typescriptlang.org/.

Basically a developer can write a program using accepted OOP programming techniques in TypeScript, transpile it into JavaScript, and then call the resulting JavaScript using a script block in a browser based HTML or an external JavaScript file, or some combination of both. It may be possible in future versions of the popular browsers to call TypeScript directly from the browser, without having to transpile it to JavaScript, though this is not yet the case.

TypeScript is designed to be independent of Microsoft development environments. If you want, you can use your own code editor, and simple command line tools to do everything your need to create TypeScript programs, regardless of whether you are using Windows or a Linux derivative. After working with Visual Studio 2017/2019 for a while, we ended up using a free to download, simplified development environment available Microsoft called **Visual Studio Code.**

Visual Studio Code

Visual Studio Code is available for free from a Microsoft website: https://code.visualstudio.com . It is available for x64-based systems only. Once installed, you will also need to install the TypeScript compiler, if you plan to use TypeScript. JavaScript does not require a compiler. You can install the TypeScript compiler using the directions here: https://code.visualstudio.com/docs/typescript/typescript-compiling . It basically uses a command line from a Command Prompt window which looks like:

```
npm install -g typescript
```

You will need a relatively current version of npm to do this. We are using Version 5.6.0. You can check your npm version by entering:

```
npm -v
```

from the command prompt. If you need to install npm, or upgrade it, you will find resources on the web which tell you how to do that.

Using QCChart2D under Linux (Ubuntu)

We tested the software under a Ubuntu workstation, both in development mode using the Ubuntu version of Visual Studio Code, and as a server, serving up the example program HTML and *.js files. In general, everything seems to work exactly the same.

Important Note for Linuz users: Compared to a Windows server, one thing to be aware of is that the file system of Ubuntu (Linux in general) is case sensitive. So be consistent in the case you use for naming and referencing your files: HTML *.js (JavaScript), and *.ts (TypeScript). Note that main library for QCChart2DTS is spelled in lower case: qcchart2dts.js. So wherever it is referenced it must be lower case. If you are using a Windows server, where the file system is not case sensitive, it doesn't matter what the case of the *.js and *.ts files are.

You can install Visual Studio Code using the Ubuntu Software app found in the main Ubuntu Toolbar. Just search on Visual Studio Code and follow the prompts.

We use the npm http-server as the test sever for our examples, though you can use any test server you want. In order to download and install the npm http-server, you first need to install npm. So go to terminal mode (ctrl-alt-t), and enter:

```
sudo apt install npm
```

Assuming npm installs correctly, install the npm http-server module globally using:

```
sudo npm install http-server -g
```

The installs all need to be handled by sudo (super user do). The previously two steps only need to be done once. The next step will probably need to be done each time you use the workstation as a development computer.

Once http-server is installed, you can start it by going to terminal mode (ctrl-alt-t), if you aren't already there. Navigate to the quinn-curtis/JSTS folder using the cd command. Once there, enter:

```
http-server
```

at the command prompt. The defaults it uses (address = 127.0.0.1 (usually the localhost address), and port = 8080) are what our example programs are setup for, so you do not need to change those, unless you are experienced and have a need to change those values. It is very important you start the http-server program from within the quinn-curtis/JSTS folder, because this is the only way the relative file locations we use in the examples work themselves out.

Normally when you are developing you go to the terminal window, start http-server and leave it running while you are working on your code. When done, you close the terminal window, closing the http-server. You would do this each time you work on your code. You can make a Ubuntu script, similar to our startlocalserver.bat file to automate starting of the http-server if you want.

You don't have to use http-server as the test server. You may have another test server, or a real server, you already use. There are ones based on python, php, node, and Ruby among others. Here is a good link describing some of your options: https://askubuntu.com/questions/1102594/how-do-i-set-up-the-simplest-http-local-server

If you are going to use the Visual Studio Code IDE as your development environment, and you plan to program using TypeScript, you will need to install TypeScript on your computer. The example below installs it globally.

```
sudo npm install -g typescript
```

And you don't have to use Visual Studio Code as your development environment. The folder setup of the examples should be usable with any JavaScript or TypeScript editor, where you make the root of your project the same as the example program folder:
quinn-curtis/JSTS/TypescriptExamplesQCChart2D/HelloChart for the HelloChart example.

Programming QCChart2D directly from JavaScript

The goal of this software is to make programming the QCChart2D library from JavaScript just as easy, powerful, and flexible, as it from C# and Java (using other versions of QCChart2D). All you need to do is create a *.js file which imports our library from its location on the server. That file would be similar to our simple HelloChart example found in the Quinn-Curtis/JSTS/JavascriptExamplesQCChart2D/ folder. All of the JavaScript programming examples are found in the
Quinn-Curtis/JSTS/JavascriptExamplesQCChart2D/   examples folder.

```
import * as QCChartTS from '../../QCChart2DTS/qcchart2dts.js';

    export async function BuildSimpleChart(canvasid) {
        let htmlcanvas = document.getElementById(canvasid);
        let chartVu = QCChartTS.ChartView.newChartView(htmlcanvas);
        if (!chartVu)
            return;
        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()));
        }
        y2[94] = 10;
        y3[0] = 95;
        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);
        let xAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.X_AXIS);
        chartVu.addChartObject(xAxis);
        let yAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
        chartVu.addChartObject(yAxis);
        let xAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(xAxis);
        xAxisLab.setTextFont(theFont);
        chartVu.addChartObject(xAxisLab);
        let yAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(yAxis);
        yAxisLab.setTextFont(theFont);
        chartVu.addChartObject(yAxisLab);
        let titleFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
        let yaxistitle = QCChartTS.AxisTitle.newAxisTitle(yAxis, titleFont, "Measurable work
output");
        chartVu.addChartObject(yaxistitle);
        let xaxistitle = QCChartTS.AxisTitle.newAxisTitle(xAxis, titleFont, "# MBAs/1000
employees");
        chartVu.addChartObject(xaxistitle);
        let xgrid = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.X_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
        chartVu.addChartObject(xgrid);
        let ygrid = QCChartTS.Grid.newGrid(xAxis, 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 EnronLabel = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1, 
theFont, "Eronix", x1[94], y2[94], QCChartTS.ChartConstants.PHYS_POS);
        EnronLabel.setXJust(QCChartTS.ChartConstants.JUSTIFY_MAX);
        EnronLabel.setYJust(QCChartTS.ChartConstants.JUSTIFY_MAX);
        EnronLabel.setTextNudge(-4, 4);
        chartVu.addChartObject(EnronLabel);
        let IGG = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1, theFont, 
"Quinn-Curtis", x1[0], y3[0], QCChartTS.ChartConstants.PHYS_POS);
        IGG.setXJust(QCChartTS.ChartConstants.JUSTIFY_MIN);
        IGG.setYJust(QCChartTS.ChartConstants.JUSTIFY_MAX);
        IGG.setTextNudge(4, 4);
        chartVu.addChartObject(IGG);
        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 toolTipFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.REGULAR, 
12);
        let datatooltip = QCChartTS.DataToolTip.newDataToolTip(chartVu);
        let xValueTemplate = 
QCChartTS.NumericLabel.newNumericLabelFormatDecs(QCChartTS.ChartConstants.DECIMALFORMAT, 0);
        let yValueTemplate = 
QCChartTS.NumericLabel.newNumericLabelFormatDecs(QCChartTS.ChartConstants.DECIMALFORMAT, 1);
        let textTemplate = QCChartTS.ChartText.newChartTextFontString(toolTipFont, "");
        textTemplate.setTextBgColor(QCChartTS.ChartColor.fromRgb(255, 255, 204));
        textTemplate.setTextBgMode(true);
        let toolTipSymbol = QCChartTS.ChartSymbol.newChartSymbol(pTransform1,
QCChartTS.ChartConstants.SQUARE,
QCChartTS.ChartAttribute.newChartAttributeColor(QCChartTS.ChartColor.GREEN));
        toolTipSymbol.setSymbolSize(10);
        datatooltip.setTextTemplate(textTemplate);
        datatooltip.setXValueTemplate(xValueTemplate);
        datatooltip.setYValueTemplate(yValueTemplate);
        datatooltip.setDataToolTipFormat(QCChartTS.ChartConstants.DATA_TOOLTIP_XY_ONELINE);
        datatooltip.setToolTipSymbol(toolTipSymbol);
        datatooltip.setEnable(true);
        //   chartVu.setCurrentMouseListener(datatooltip);
        chartVu.updateDraw();
    }
```

Note that the **import** statement imports the QCChart2D library as the file qcchart2dts.js from its location in the ../../QCChart2DTS/ folder, relative to the location of the *.js file you are programming.

```
import * as  QCChartTS from '../../QCChart2DTS/qcchart2dts.js';
```

The “../../QCChart2DTS” folder specification means to back up two directory levels and go to the QCChart2DTS folder, where the qcchart2dts.js library file is located.

Note: You can't back up further than the root directory of the server because that would be a security violation. When using the npm “http-server” command-line server, by running our simple batch file “startlocalserver.bat” in the Quinn-Curtis/JSTS folder, this means you can't back up any further than the Quinn-Curtis/JSTS folder, because that represents the root directory of the server.

Note: In order to test the HTML files you create, you must use a real, or command line, server such as the npm “http-server” to serve up the html files you create. Because of the JavaScript and HTML features used in the software, you **cannot** just reference the HTML files using a browser and point to the physical file location. Instead you must start a local server in one of our folders and access the HTML files from there, using a http:// address like:

http://127.0.0.1:8080/JavascriptExamplesQCChart2D/HelloChart/HelloChart.html

where http://127.0.0.1:8080/ represents the local server. You can usually replace 127.0.0.1 with *localhost*.

http://127.0.0.1:8080/JavascriptExamplesQCChart2D/HelloChart/HelloChart.html

This assumes the local server responds to the URL http://127.0.0.1:8080/ and that the root directory of the local server is the Quinn-Curtis/JSTS folder. We recommend that you install the npm “http-server” command line server globally according to the instructions here: https://www.npmjs.com/package/http-server

Then you can start the server using the **startlocalserver.bat** batch file located in the Quinn-Curtis/JSTS folder. You will need a relatively current version of npm to properly install the http-server package.

As a result of the **import * as QCChartTS** statement, all JavaScript objects found in the library are located in the namespace QCChartTS, which prevents them from conflicting with any other libraries you may be using which have similar named objects. The file qcchart2dts.js contains the entire QCChart2D library.

A new chart drawing space is created inside the BuildSimpleChart function using the call:

```
        let chartVu = QCChartTS.ChartView.newChartView(htmlcanvas);
```

passing in the HTML canvas you have reserved for the chart. The reset of the code creates different types of chart elements as adds them to the chartVu containter.

Our original QCChart2D software made extensive use of overloaded constructors and function names. But JavaScript supports neither of those. So, overloaded constructors needed to be converted into static function calls, with unique names, which return an instantiated and initialized JavaScript object. We tended to just list out important parameter names, or types, after the function name, in order to make the function names unique. Hence, long-winded functions names such as:

```
   let pTransform1 = 
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
```

which is the static **CartesianCoordinates** class constructor of the coordinate system plot objects will be placed in.

Note: (var vs let) - Many of Javascript examples mix variable declartions **var** and **let** interchangeably. But **let** and **var** are not exactly the same. The **let** keyword is used to create variables which are localized to the current block of JavaScript. These variable are not accessible from the global scope of the window. The **var** keyword will define variables which are global. That difference applies to variables created in your Javascript code not defined within a function. In the case of variables created inside a function, **let** and **var** do exactly the same thing. In both cases the variable is accessible only within the function. So, in our examples, as long as the code is inside a function, we use var and let interchangeably. If you look at the JavaScript generated by compiling TypeScript files, you will see that that let declarations remain let declarations and do not get replaced with var. But inside functions, **var** can be used instead. See web-based Javascript/TypeScript tutorials for more of a discussion: https://codeburst.io/difference-between-let-and-var-in-javascript-537410b2d707 ..

The programmer customizes the charts by creating chart and plot objects, and adding them to the parent **ChartView** container. In the example below, the coordinate system is created and then used in the creation of the chart background (**Background**), x- and y-axes (**LinearAxis**), x- and y-axes labels (**NumericAxisLabels**), axis titles (**AxisTitle**), and axis grids (**Grid**).

[JavaScript]

```
    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);
        let xAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, 
QCChartTS.ChartConstants.X_AXIS);
        chartVu.addChartObject(xAxis);
        let yAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, 
QCChartTS.ChartConstants.Y_AXIS);
        chartVu.addChartObject(yAxis);
        let xAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(xAxis);
        xAxisLab.setTextFont(theFont);
        chartVu.addChartObject(xAxisLab);
        let yAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(yAxis);
        yAxisLab.setTextFont(theFont);
        chartVu.addChartObject(yAxisLab);
        let titleFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
        let yaxistitle = QCChartTS.AxisTitle.newAxisTitle(yAxis, titleFont, "Measurable work 
output");
        chartVu.addChartObject(yaxistitle);
        let xaxistitle = QCChartTS.AxisTitle.newAxisTitle(xAxis, titleFont, "# MBAs/1000 
employees");
        chartVu.addChartObject(xaxistitle);
        let xgrid = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.X_AXIS, 
QCChartTS.ChartConstants.GRID_MAJOR);
        chartVu.addChartObject(xgrid);
        let ygrid = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.Y_AXIS, 
QCChartTS.ChartConstants.GRID_MAJOR);
        chartVu.addChartObject(ygrid);
```

Note that all constants used in the setup, and all QCChart2D types in general, need to be prefixed by the QCChartTS namespace. When you start writing programs you will find that you spend most of your time chasing down where you left off the namespace identifier. Also, note that many calls into the QCChart2D objects are function calls (set., get..). There are also getter/setter versions of most of the property calls. While it may be possible to access internal variables we designate public within the library, this is not encouraged and may not have exactly the same affect as calling the appropriate getter or setter function.

Note: The identifier QCChartTS is not fixed, it is just the namespace specified in the **import** statement at the top of the file. We use it all of our examples, and this manual, to avoid any confusion in the presentation of the programming examples.

In this example the BuildSimpleChart function creates the initial graph. Since a graph without data is not of much use, simulated data is added to the chart.

[JavaScript]

```
     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()));
        }
        y2[94] = 10;
        y3[0] = 95;
        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);
```

In it you will find that simulated data is first created in local number arrays, and those are passed into one of our Dataset classes. The Dataset classes are then passed to one of our plotting classes.

In JavaScript, most of the work is done in the BuildSimpleChart function of the HelloChart.js file. But it needs to be invoked from the HTML file you want to place the chart in. Look at the HelloChart/HelloChart.html example to see how this is done.

Most of our examples utilize a browser UI framework called Bootstrap, which lets the web page programmer create web apps which adapt to output devices across a large range of resolutions, ranging from mobile phones, tablets, and PC's. It has a number of include files which are referenced within the HTML page. Also, we use the jQuery library in many of our examples. But in this example we have stripped the HTML to its bare bones, with just a button, and an HTML5 canvas to place the chart in.

```
<!DOCTYPE html>
<head>
    <meta charset="utf-8" />
    <title>Simple Chart Example</title>
</head>
<body>

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

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


    import { BuildSimpleChart} from './HelloChart.js';

          function simplechart() {
            BuildSimpleChart("chartCanvas1");
          }


          // Since loading of script module is asynchronous, need to assign onclick event
handlers after module loaded.
```

```
          document.getElementById("simplechart_menuitem").onclick = simplechart;

        </script>

        <br>
        <br>

</body>
</html>
```

In this example the button click for the Simple Chart button invokes the JavaScript *HelloChart* function call inside the script section. That function in turn calls the **BuildSimpleChart** in the imported HelloChart.js file, passing in the id of the Canvas object you have placed in the HTML page. It is critically important that you place a HTML5 canvas object in your web page, because that is where the chart is placed.

```
    <div id="canvasdiv">
        <canvas id="chartCanvas1" width="800" height="600"></canvas>
     </div>
```

You can give the canvas any id that you want, but this needs to passed into the JavaScript work file (HelloChart.js in this case), where it is converted into an HTML Canvas object and passed to the library using one of our chart constructors. Note in the example below that casting is necessary in the TypeScript code.

[JavaScript]
```
let htmlcanvas = document.getElementById(canvasid);
```

[TypeScript]

```
let htmlcanvas: QCChart2DTS.Canvas = <QCChart2DTS.Canvas>document.getElementById(canvasid);
```

Note: The **import** statement found in the script uses type=”module”. This is part of the ES6 module import specification, supported by all current browsers. It let you import function and variables exported from a JavaScript file, like all of our *.js examples. In this case we import the BuildSimpleChart function from the HelloChart.js source file. ES6 module loading is asynchronous, and generally takes place after the rest of the page has been loaded. Because of this, the click handler for the button (.onclick) cannot be assigned as part of the HTML button code, because the HelloChart function in the script block has not yet been loaded. Instead is it assigned explicitly in the javascript code, after the HelloChart.js module has been loaded.

```
          document.getElementById("simplechart_menuitem").onclick = HelloChart;
```

You can read more about the ES6 module loading specification on the web:
https://www.sitepoint.com/using-es-modules/ .

Programing QCChart2D using TypeScript

When using JavaScript, nothing needs to be compiled. The JavaScript file is loaded by the browser as part of the HTML page loading process. The JavaScript code is then interpreted and executed by the browser without the use of a compiler. This is not the case if you plan to write your code in TypeScript. The current generation of browsers (circa June. 2020) will not execute TypeScript code directly. Instead, the TypeScript code must be compiled into equivalent JavaScript, and it is that JavaScript file which is loaded by the browser. This may change in the future when the browser makers decide it is worthwhile to process TypeScript files directly or through some easily obtainable add-on.

All of the TypeScript examples are found in the Quinn-Curtis/JSTS/TypescriptExamplesQCChart2D folder. Each example folder contains an HTML file to load into a browser to test the example, a *.ts file which loads the qcchart2dts.js library and builds the chart, and a tsconfig.json configuration file which tells the TypeScript compiler how to compile the any TypeScript (*.ts) files it finds.

You open the example folder (HelloChart) using your compiler (Visual Studio Code in this case). This makes the folder the root location of the current TypeScript project. It looks to the folder for a tsconfig.json file for the compiler options it will use to compile the TypeScript code within the project. There are a many compiler options and you will need to study those for advanced projects. Here is one link: https://www.typescriptlang.org/docs/handbook/compiler-options.html to start learning from. Our examples use a very simplified tsconfig.json file which basically tells the compiler to compile all *.ts files it finds in the folder, and place the resulting *.js files in the same folder.

```
{
  "compilerOptions": {
    "target": "es6",
    "lib": [ "es6", "dom" ],
    "sourceMap": true,
    "outDir": "./",
    "declaration": true,
    "module": "es6",
    "strict": true,
    "moduleResolution": "node"
  },
  "exclude": [
    "node_modules"
  ]

}
```

The purpose behind the different options are explained below. We do not really understand the thousands of combinations of options you can use, but hope to give you an idea of what worked for our own program development.

compilerOptions:

target: es6 -	Specify ECMAScript target version. Permitted values are 'es3', 'es5', 'es6', 'es2015', 'es2016', 'es2017', 'es2018', 'es2019', 'es2020' or 'esnext'. We use es6 in all of our tsconfig.json files, but es5 to esnext all seem to work.
lib: [es6, dom] -	Specify library file to be included in the compilation. Requires TypeScript version 2.0 or later.
sourceMap: true -	Generates corresponding '.map' file., used in debugging

outDir: "./" -	send the compiler output (*.js, *.map, *.d.ts) to to this location), the root directory of the project in this case. Specifying "./" places the compiler output files (*.js, *.map, *.d.ts) in the project folder, next to the source *.ts files. In many cases you will want to put them somewhere else. Here are some other examples: "./built" - send files to the current folders /built folder. "../" - up one directory level from the current folder. "../built" - up one directory level from the current folder into the /built folder there. **Note:** Where ever you send them, that is where you should access them from (unless you copy them somewhere else). So the HTML file you use must reference the imported *.js file from the correct folder, which is not necessarily the project folder.
declaration: true –	generate the *.d.ts map file, which enables TypeScript debugging in the browser
module: es6 -	Specify module code generation: 'none', 'commonjs', 'amd', 'system', 'umd', 'es2015' or 'esnext'.
strict: true -	Enable all strict type checking options. Requires TypeScript version 2.3 or later. Not easy to work with, but it does catch countless bad programming errors involving use of null or uninitialized objects. The underlying qcchart2dts.js library was developed using this option.
moduleResolution: node -	Specifies module resolution strategy: 'node' (Node) or 'classic' (TypeScript pre 1.6) .
exclude: [- node_modules]	Specifies a list of files to be excluded from compilation. The 'exclude' property only affects the files included via the 'include' property and not the 'files' property. Glob patterns require TypeScript version 2.0 or later.

The source TypeScript file looks similar to the JavaScript file in the previous section. Since TypeScript is a class oriented language, unlike JavaScript, all of the examples have been converted to simple class equivalents of the of the original JavaScript examples. In this example, a main class (HelloChart) is defined with a BuildSimpleChart member function.

```
import * as  QCChartTS from '../../QCChart2DTS/qcchart2dts.js';

export class HelloChart {

    public constructor() {

    }
    public async  BuildSimpleChart(canvasid: string) {

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

        let chartVu: QCChartTS.ChartView = QCChartTS.ChartView.newChartView(htmlcanvas);
        if (!chartVu) return;
        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);
        let xAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.X_AXIS);
        chartVu.addChartObject(xAxis);
        let yAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
        chartVu.addChartObject(yAxis);
        let xAxisLab: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(xAxis);
        xAxisLab.setTextFont(theFont);
        chartVu.addChartObject(xAxisLab);
        let yAxisLab: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(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(yAxis, titleFont,
"Measurable work output");
        chartVu.addChartObject(yaxistitle);
        let xaxistitle: QCChartTS.AxisTitle = QCChartTS.AxisTitle.newAxisTitle(xAxis, titleFont,
"# MBAs/1000 employees");
        chartVu.addChartObject(xaxistitle);
        let xgrid: QCChartTS.Grid = QCChartTS.Grid.newGrid(xAxis, yAxis,
QCChartTS.ChartConstants.X_AXIS, QCChartTS.ChartConstants.GRID_MAJOR);
        chartVu.addChartObject(xgrid);
        let ygrid: QCChartTS.Grid = QCChartTS.Grid.newGrid(xAxis, 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 toolTipFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Arial",
QCChartTS.ChartFont.REGULAR, 12);
```

```
        let datatooltip: QCChartTS.DataToolTip = QCChartTS.DataToolTip.newDataToolTip(chartVu);
        let xValueTemplate: QCChartTS.NumericLabel =
QCChartTS.NumericLabel.newNumericLabelFormatDecs(QCChartTS.ChartConstants.DECIMALFORMAT, 0);
        let yValueTemplate: QCChartTS.NumericLabel =
QCChartTS.NumericLabel.newNumericLabelFormatDecs(QCChartTS.ChartConstants.DECIMALFORMAT, 1);
        let textTemplate: QCChartTS.ChartText =
QCChartTS.ChartText.newChartTextFontString(toolTipFont, "");
        textTemplate.setTextBgColor(QCChartTS.ChartColor.fromRgb(255, 255, 204));
        textTemplate.setTextBgMode(true);
        let toolTipSymbol: QCChartTS.ChartSymbol =
QCChartTS.ChartSymbol.newChartSymbol(pTransform1, QCChartTS.ChartConstants.SQUARE,
QCChartTS.ChartAttribute.newChartAttributeColor(QCChartTS.ChartColor.GREEN));
        toolTipSymbol.setSymbolSize(10);
        datatooltip.setTextTemplate(textTemplate);
        datatooltip.setXValueTemplate(xValueTemplate);
        datatooltip.setYValueTemplate(yValueTemplate);
        datatooltip.setDataToolTipFormat(QCChartTS.ChartConstants.DATA_TOOLTIP_XY_ONELINE);
        datatooltip.setToolTipSymbol(toolTipSymbol);
        datatooltip.setEnable(true);
        //   chartVu.setCurrentMouseListener(datatooltip);

        chartVu.updateDraw();

    }

}
```

The “../../QCChart2DTS” folder specification means to back up two directory levels and go to the QCChart2DTS folder, where the qcchart2dts.js library file is located. Note that is is the qcchart2dts.js (Javascript) library that is loaded, not a TypeScript (*.ts) file.

Note: You can't back up further than the root directory of the server because that would be a security violation. When using the npm “http-server” command-line server, by running our simple batch file “startlocalserver.bat” in the Quinn-Curtis/JSTS folder, this means you can't back up any further than the Quinn-Curtis/JSTS folder, because that represents the root directory of the server.

Note: In order to test the HTML files you create, you must use a real, or command line, server such as the npm “http-server” to serve up the html files you create. Because of the javascript and html features used in the software, you **cannot** just reference the HTML files using a browser and point to the physical file location. Instead you must start a local server in one of our folders and access the HTML files from there, using a http:// address like:

http://127.0.0.1:8080/TypescriptExamplesQCChart2D/HelloChart/HelloChart.html

You can usually replace 127.0.0.1 with *localhost*.

http://localhost:8080/TypescriptExamplesQCChart2D/HelloChart/HelloChart.html

This assumes the local server responds to the URL http://127.0.0.1:8080/ and that the root directory of the local server is the quinn-curtis/JSTS folder. We recommend that you install the npm http-server

command line server globally according to the instructions here: https://www.npmjs.com/package/http-server
Then you can start the server using the startlocalserver.bat batch file located in the Quinn-Curtis/JSTS. You will need a relatively current version of npm to properly install the http-server package.

As a result of the import * as QCChartTS statement, all JavaScript objects found in the library are located in the namespace QCChartTS, which prevents them from conflicting with any other libraries you may be using which have similar named objects. The file qcchart2dts.js contains the entire QCChart2D library

In it you will find that simulated data is created and placed in standard Javascript arrays. The arrays are usually arrays of number and Date obects, but in same cases may be arrays of strings, and other classes, such as our ElapsedTime class.

So most of the work is done in the BuildSimpleChart member function of the HelloChart clsss. But it needs to be invoked from the HTML file you want to place the chart in. Look at the HelloChart/HelloChart.html example to see how this is done.

Most of our examples utilize a browser UI framework called Bootstrap, which lets the web page programmer create web apps which adapt to output devices across a large range of resolutions, ranging from mobile phones, tablets, and PC's. It has a number of include files which are referenced within the HTML page. Also, we use the jQuery library in many of our examples. But in this example we have stripped the HTML to its bare bones, with just a button, and an HTML5 canvas to place the chart in. The example references the HelloChart.js file found in the JavascriptExamplesQCChart2D/HelloChart folder.

```
<!DOCTYPE html>
<head>
    <meta charset="utf-8" />
    <title>Simple Chart Example</title>
</head>
<body>

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

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


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

     var hellochart = new HelloChart();


          function simplechart() {
            hellochart.BuildSimpleChart("chartCanvas1");
          }


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

        <br>
        <br>

</body>
</html>
```

In this example the button click for the Simple chart button invokes the JavaScript *HelloChart* function call inside the script section. The HelloChart class is imported from the HelloChart.js file, and an instance of the class is created as the local object *hellochart*. The Simple chart button event handler function in turn calls the hellochart.BuildSimpleChart in the imported HelloChart.js file, passing in the id of the Canvas object you have placed in the HTML page.

Note: The **import** statement found in the script uses type=”module”. This is part of the ES6 module import specification, supported by all current browsers. It let you import function and variables exported from a JavaScript file, like all of our *.js examples. In this case we import the HelloChart class from the HelloChart.js source file. ES6 module loading is asynchronous, and generally takes place after the rest of the page has been loaded. Because of this, the click handler for the button (.onclick) cannot be assigned as part of the HTML button code, because the HelloChart function in the script block has not yet been loaded. Instead is it assigned explicitly in the javascript code, after the HelloChart.js module has been loaded.

```
          document.getElementById("simplechart_menuitem").onclick = HelloChart;
```

You can read more about the ES6 module loading specification on the web:
https://www.sitepoint.com/using-es-modules/ .

Programing QCChart2D within Wordpress

Using QCChart2D within a Web content management system, such as WordPress is almost as easy. Normally, in Wordpress pages you don't have access to JavaScript. But if you add a plug-in to your website, such as Code Embed by David Arliss, you can. Once you do that, you can add a section of HTML to a specific location in your Wordpress page, which defines a canvas element, loads a module consisting of an external JavaScript file, and then execute a function (**BuildOHLCChart** in the example below) inside that file to build the chart. In that case the embedded code might look something like:

```
<canvas id="chartCanvas1" width="600" height="600"></canvas>
<script type="module">
   import { BuildOHLCChart} from http://yourwebsite/OHLCChart.js;

   BuildOHLCChart("chartCanvas1");
</script>
```

Alternatively, you can create your chart as a standalone HTML page, and then reference that HTML page Wordpress page using an iframe plug-in, such as Iframe by Webvitaly. You can create your chart in a standalone HTML page you can test and debug independent of Wordpress. Once you get it working right, you can add it to your Wordpress page using the iframe plug-in using code similar to the code below:

```
[iframe src="http://yourwebsite.com/OHLCChart.html" id="iframe_id" width="100%" height = "600"]
```

If your intended use is a simple chart initialized with data, the Code Embed approach is probably the easiest. If you require a complex UI to go along with the chart, you will probably need to use the iframe approach. We demonstrate both methods on our website http://quinn-curtis.com .

Directory Structure of QCChart2D for JavasScript/TypeScript

The QCChart2D for JavasScript/TypeScript class library uses the following directory structure:

Drive:

- Quinn-Curtis\ - Root directory
  - JSTS\ - Quinn-Curtis JavaScript/TypeScript based products directory
    - Docs\ - Quinn-Curtis JavaScript/TypeScript related documentation directory
    - QCChart2DTS\ - The QCChart2DTS library folder
    - AspNetChartVS2019
      - ASPNetCoreMVCWebApplication1- Asp.Net Core 3.0 MVC example
      - AspNetCoreWebApplication1- A basic Asp.Net Core 3.0 application
      - WebFormApplication1 - A basic Asp.Net Form-based web application
      - AspWebAppSinglePageWebApplication1 - A basic Asp.Net Form-based single page application
    - JavasScriptExamplesQCChart2D
      - AntennaCharts\ - A simple antenna chart.example.
      - Bargraphs\ - Horizontal, vertical, stacked, group, histogram, floating
      - CalendarData\ - Line plots, bar plots and log plots that use time/date data. Also reading time/date data from a file.
      - ChartAxes\ - Linear, logarithmic, time/date, custom hours time/date and polar axes. Also axes labels.

ChartEventExamples\ - A variety of examples using ChartEvent objects as the source data.

CustomDataToolTips\ - Creating custom data tooltips for an OHLC plot, multiple stacked graphs and a pie chart.

DataCursorsAndMarkers\ - Using data cursors and markers

DynamicCharts\ - Scrolling lines, bars, scatter plots, data logging, instrument simulation, chart animation.

EditChartExample\ - How to implement edit dialogs for chart objects

FinancialExamples\ - OHLC plots, candlestick plots, financial log plots, option chart, technical analysis chart.

HelloChart – a simple chart

ImageCharts\ - Using images as chart data elements, chart backgrounds and annotations.

LogPlots\ -Logarithmic plots for financial charts and engineering charts.

MiscCharts\ - A line gap chart.

MouseListeners\ - Data tooltips, data cursors, moving data points, moving chart objects.

MultiLinePlots\ - Group multi-line plots, stacked line plots, multiple single line graphs.

MultipleAxes\ - Multiple axes graphs

NewDemosRev2 – New examples for Revision 2.0 features.

OHCChart – A standalone OHLC chart with scroll bar.

PieCharts\ - Simple pie charts and pie charts combined with line and bar plots.

PolarCharts\ - Polar line, line file and scatter plots. Also includes an Antenna plot example.

ResizeExamples\ - Fixed size frame, resizeable frame with fixed sized objects, resizeable frame with resizeable objects, scrollable panel as a view into a much larger fixed size frame.

ScatterPlots\ - Simple scatter, line and line-marker plots, scatter plots with variable size and color symbols, cell plots, arrow plots. Also, labeling the data point values or a line marker plot.

SimpleLinePlots\ - Simple line plots with linear and time/date coordinate systems. Also filled and step lines.

ZoomExamples\ - Zooming simple linear axes, super zooming of multiple axes, zooming of time/data based data.

TypeScriptExamplesQCChart2D

Save folder structure as JavaScriptExamplesQCChart2D

Versions of QCChart2DTS

30-Day Trial Version

The trial version of QCChart2D for JavasScript/TypeScript is downloaded as a zip file named Trial_QCChart2DJSTS.zip. The 30-day trial version stops working 30 days after the initial download. The trial version includes a version message in the upper left corner of the graph window that cannot be removed. The 30-trail version does *not* include a license to utilize the software on a commercial website.

Developer Version

The developer version of QCChart2D for JavasScript/TypeScript is downloaded as a zip file, name something similar to JSTSDEV1R3x0x353x1.zip. The developer version does not time out and you can use it to create browser applications that you can distribute royalty free. You can download free updates for a period of 2-years. When you placed your order, you were e-mailed download link(s) that will download the software. Those download links will remain active for at least 2 years and should be used to download current versions of the software. After 2 years you may have to purchase an upgrade to continue to download current versions of the software

Chapter Summary

The remaining chapters of this book discuss the **QCChart2D for JavasScript/TypeScript** interactive charting package designed to run on any hardware that supports a browser supporting HTML5..

Chapter 2 presents the overall class architecture of the **QCChart2D for JavasScript/TypeScript** and summarizes all of the classes found in the software.

Chapter 3 describes the dataset classes that hold chart data.

Chapter 4 describes the various classes that implement the Cartesian, time and polar coordinate systems supported by the software.

Chapter 5 describes the **ChartView** container class that manages the chart objects**.**

Chapter 6 describes the color, gradient and background classes.

Chapters 7, 8 and 9 describe the classes that create chart axes, axis labels and axis grids.

Chapter 10 describes the classes used to display simple xy data (one y-value for each x-value) as line plots, bar plots, scatter plots., line-marker plots and simple versa plots.

Chapter 11 describes the classes used to display group data (one or more y-values for each x-value) as line plots, group bar plots, stacked bar plots, scatter plots, open-high-low-close plots, candlestick plots, box and whisker plots, floating stacked bar plots, and group versa plots.

Chapter 12 describes some utility Button and Scroll bar classes

Chapter 13, 14, 15 and 16 describe classes that add interactive elements to a chart. Chapter 13 describes data marker and data cursor classes used to mark and highlight data points using the mouse. Chapter 14 describes classes that can move chart objects, individual data points and the coordinate system. Chapter 15 adds a "zooming" class where mouse/touch events define a new scaling range for a chart, redrawing the chart axes and data automatically. Chapter 16 describes a generalized data tool-tip class that can display the x and/or y data values for a data point, or custom text associated with the data point.

Chapter 17 describes classes for the display of pie and ring charts.

Chapter 18 describes classes for the display of data in a polar, and antenna, chart format

Chapter 19 describes classes for the display chart legends, used to create visual aids for the interpretation of different elements making up the chart.

Chapter 20 describes generalized classes for displaying formatted text in a chart.

Chapter 21 describes how the DatasetViewer class is used to display simple and group datasets in a table format.

Chapter 22 describes how to use a generalized shape class for the display of arbitrary lines, shapes, images and arrows.

Chapter 23 describes chart printing and the creation of JPEG files.

Chapter 24 is a tutorial that describes how to use QCChart2D to create browser applications..

Chapter 25 is a collection of Frequently Asked Questions about QCChart2D for JavasScript/TypeScript.

2. Class Architecture of the QCChart2D for JavaScript/TypeScript Class Library

Major Design Considerations

This chapter presents an overview of the **QCChart2D for JavaScript/TypeScript** class architecture. It discusses the major design considerations of the architecture:

- It is based on the HTML5 Canvas API model and standard JavaScript/TypeScript classes.
- New charting objects can be added to the library without modifying the source of the base classes.
- There are no limits regarding the number of data points in a plot, the number of plots in graph, the number of axes in a graph, the number of coordinate systems in a graph.
- There are no limits regarding the number of legends, arbitrary text annotations, bitmap images, geometric shapes, titles, data markers, cursors and grids in a graph.
- Users can interact with charts using classes that support an event driven model.

The chapter also summarizes the classes in the **QCChart2D for JavaScript/TypeScript** library.

There are five primary features of the overall architecture of the **QCChart2D for JavaScript/TypeScript** classes. These features address major shortcomings in existing charting software for use in JavaScript and other computer languages.

- First, **QCChart2D for JavaScript/TypeScript** uses the standard HTML5/Canvas architecture. Charts are placed in a **ChartView** window and that manages the chart objects drawn into the associated HTML5 Canvas. Position one or more **ChartView** objects in web page windows using multipe charts per canvas, or multiple canvases per web page. Use standard HTML layout elements to such as <div> to position canvases on your web page. Mix charts with other HTML elements on the same web page. Charts use the event-processing model for handling mouse and keyboard events.
- Second, the library is extensible. Hundreds of different vertical markets use computer charting. The charts used in each market have a unique look and feel. A well-designed object oriented charting package allows the programmer to extend the software without modifying the source of the underlying classes. Instead, the programmer extends the software by deriving a new class from an existing base class. The new, derived class localizes custom source code and the source of the underlying classes remains unchanged. In the **QCChart2D for JavaScript/TypeScript** classes a user can subclass an existing class and create new, custom charting objects. Examples of custom

charting objects are specialized plots that extend the **SimplePlot**, or **GroupPlot** classes to include new plot types for applications such as stock market technical analysis, statistical process control, and medical instrumentation, to name a few.

- Third, the library has no limits regarding the number of data points in a plot, the number of plots in graph, the number of axes in a graph, and the number of coordinate systems in a graph. A major weakness in many commercial graphics packages is that they have hard coded limits that restrict the number of data points, axes, or coordinate systems. A simple business bar chart may only contain 3 or 4 data points, depicting a sales forecast. An audio mixer application may require 32 million plotted points, represented by 32 traces of 1 million points each, each trace representing 20 seconds of audio sampled at 50 kHz.

  The most effective way to compare data is to overlay it in the same graph. Often the data series have different dynamic ranges. If the data series are plotting using the same scale, it is difficult do see the correlation, or lack of, in the data. A better solution is to create a unique scale for each data series, with associated axes, and plot each data series with respect to its own scale. All of the plots will overlay the same area of the graph. Many advanced charting packages do not support more than one coordinate system per graph, while most others have some fixed limit such as 2, 4 or 8 scales per graph. The number of coordinate systems for a graph can be as large as the number of data series plotted in the graph. Charts can have hundreds of data series at once; therefore, a flexible charting package needs to allow for an equal number of simultaneous coordinate systems.

- Fourth, a well-constructed chart often displays more than just data. Other common chart objects include legends, arbitrary text annotations, bitmap images, geometric shapes, titles, data markers, cursors and grids. A chart can contain zero or more of these objects. It may contain 100 of one type, 5 of another type, and 1 of a third. **QCChart2D for JavaScript/TypeScript** contains no limits restricting the number of instances of a given chart object in a graph, and no limit on the total number of chart objects in a graph.

- Fifth, an end user needs to interact with the graph using the mouse and/or keyboard. A user can use the mouse to select data points, text annotations, axes, image objects, and other shapes, and position them in the graph. Create data markers and move them around the chart under mouse or program control. Automatically rescale one or more chart axes using mouse controlled zooming.

QCChart2D for JavaScript/TypeScript Class Summary

The following categories of classes realize these design considerations.

Chart view class	The chart view class manages the graph objects placed in the graph
Data classes	There are data classes for simple xy and group data types.
Scale transform classes	The scale transform classes handle the conversion of physical coordinate values to working coordinate values for a single dimension.

Coordinate transform classes	The coordinate transform classes handle the conversion of physical coordinate values to working coordinate values for a parametric (2D) coordinate system.
Attribute class	The attribute class encapsulates the most common attributes (line color, fill color, line style, line thickness, etc.) for a chart object.
Auto-Scale classes	The coordinate transform classes use the auto-scale classes to establish the minimum and maximum values used to scale a 2D coordinate system. The axis classes also use the auto-scale classes to establish proper tick mark spacing values.
Charting object classes	The chart object classes includes all objects placeable in a chart. That includes axes, axes labels, plot objects (line plots, bar graphs, scatter plots, etc.), grids, titles, backgrounds, images and arbitrary shapes.
Mouse/Touch interaction classes	These classes, directly and indirectly trap mouse/touch events and permit the user to create and move data cursors, move plot objects, display tooltips and select data points in all types of graphs.
File and printer rendering	The charting software relies on the browser printing system for handling hardcopy output. Image files can be used as elements of a chart, and the chart itself can be used as a browser image.
Miscellaneous utility classes	Other classes use these for data processing.

A summary of each category appears in the following section.

Chart Window Classes

ChartView

The starting point of a chart is the **ChartView** class. The **ChartView** class manages a collection of chart objects in a chart and automatically updates the chart objects when the underlying window processes a paint event.

Data Classes

ChartDataset
- **SimpleDataset**
  - **TimeSimpleDataset**
  - **ElapsedTimeSimpleDataset**
  - **EventSimpleDataset**
- **GroupDataset**
  - **TimeGroupDataset**
  - **ElapsedTimeGroupDataset**
  - **EventGroupDataset**

The dataset classes organize the numeric data associated with a plotting object. There are two major types of data supported by the **ChartDataset** class. The first is simple xy data, where for every x-value there is one y-value. The second data type is group data, where every x-value can have one or more y-values.

ChartDataset	The abstract base class for the other dataset classes. It contains data common to all of the dataset classes, such as the x-value array, the number of x-values, the dataset name and the dataset type.
SimpleDataset	Represents simple xy data, where for every x-value there is one y-value.
TimeSimpleDataset	A subclass of **SimpleDataset**, it is initialized using JavaScript **Date** objects in place of the x- or y-values.
ElapsedTimeSimpleDataset	A subclass of **SimpleDataset**, it is initialized with **ChartTimeSpan** objects, or milliseconds, in place of the x- or y-values.
EventSimpleDataset	A subclass of **SimpleDataset**, it is initialized with **ChartEvent** objects, where the data is to be plotted using one of the simple plot types.
GroupDataset	Represents group data, where every x-value can have one or more y-values.
TimeGroupDataset	A subclass of **GroupDataset**, it uses JavaScript **Date** objects as the x-values, and floating point numbers as the y-values.
ElapsedTimeGroupDataset	A subclass of **GroupDataset**, it uses **ChartTimeSpan** objects, or milliseconds, as the x-values, and floating point numbers as the y-values.
EventGroupDataset	A subclass of **GroupDataset**, it uses **ChartEvent** objects, where the data is to be plotted using one of the group plot types.

Scale Classes

ChartScale
- **LinearScale**
- **LogScale**
- **TimeScale**
- **ElapsedTimeScale**
- **EventScale**

The **ChartScale** abstract base class defines coordinate transformation functions for a single dimension. It is useful to be able to mix and match different scale transform functions for x- and y-dimensions of the **PhysicalCoordinates** class. The job of a **ChartScale** derived object is to convert a dimension from the current *physical* coordinate system into the current *working* coordinate system.

LinearScale	A concrete implementation of the **ChartScale** class. It converts a linear physical coordinate system into the working coordinate system.
LogScale	A concrete implementation of the **ChartScale** class. It converts a logarithmic physical coordinate system into the working coordinate system.
TimeScale	A concrete implementation of the **ChartScale** class. converts a date/time physical coordinate system into the working coordinate system.
ElapsedTimeScale	A concrete implementation of the **ChartScale** class. converts an elapsed time coordinate system into the working coordinate system.
EventScale	A concrete implementation of the **ChartScale** class. converts an event coordinate system into the working coordinate system.

Coordinate Transform Classes

UserCoordinates
- **WorldCoordinates**
  - **WorkingCoordinates**
    - **PhysicalCoordinates**
      - **CartesianCoordinates**
        - **ElapsedTimeCoordinates**
        - **PolarCoordinates**

```
          AntennaCoordinates
          EventCoordinates
     TimeCoordinates
```

The coordinate transform classes maintain a 2D coordinate system. Many different coordinate systems are used to position and draw objects in a graph. Examples of some of the coordinate systems include the device coordinates of the current window, normalized coordinates for the current window and plotting area, and scaled physical coordinates of the plotting area.

UserCoordinates	This class manages the interface to the Canvas drawing functions and contains routines for drawing lines, rectangles and text using Canvas device coordinates.
WorldCoordinates	This class derives from the **UserCoordinates** class and maps a device independent world coordinate system on top of the Canvas device coordinate system.
WorkingCoordinates	This class derives from the **WorldCoordinates** class and extends the physical coordinate system of the *plot area* (the area typically bounded by the charts axes) to include the complete *graph area* (the area of the chart outside of the *plot area*).
PhysicalCoordinates	This class is an abstract base class derived from **WorkingCoordinates** and defines the routines needed to map the physical coordinate system of a plot area into a working coordinate system. Different scale objects (**ChartScale** derived) are installed for converting physical x- and y-coordinate values into working coordinate values.
CartesianCoordinates	This class is a concrete implementation of the **PhysicalCoordinates** class and implements a coordinate system used to plot linear, logarithmic and semi-logarithmic graphs.
TimeCoordinates	This class is a concrete implementation of the **PhysicalCoordinates** class and implements a coordinate system used to plot **Date** time-based data.
ElapsedTimeCoordinates	This class is a subclass of the **CartesianCoordinates** class and implements a coordinate system used to plot elapsed time data.
PolarCoordinates	This class is a subclass of the **CartesianCoordinates** class and implements a coordinate system used to plot polar coordinate data.
AntennaCoordinates	This class is a subclass of the **CartesianCoordinates** class and implements a coordinate system used to plot antenna coordinate data. The antenna coordinate system differs from the more common polar coordinate

	system in that the radius can have plus/minus values, the angular values are in degrees, and the angular values increase in the clockwise direction.
EventCoordinates	This class is a subclass of the **CartesianCoordinates** class and implements a coordinate system used to plot ChartEvent based data.

Attribute Class

ChartAttribute
ChartGradient

This class consolidates the common line and fill attributes as a single class. Most of the graph objects have a property of this class that controls the color, line thickness and fill attributes of the object. The **ChartGradient** class expands the number of color options available in the **ChartAttribute** class.

ChartAttribute	This class consolidates the common line and fill attributes associated with a **GraphObj** object into a single class.
ChartGradient	A **ChartGradient** can be added to a **ChartAttribute** object, defining a multicolor gradient that is applied wherever the color fill attribute is normally used

Auto-Scaling Classes

AutoScale
- **LinearAutoScale**
- **LogAutoScale**
- **TimeAutoScale**
- **ElapsedTimeAutoScale**
- **EventAutoScale**

Usually, programmers do not know in advance the scale for a chart. Normally the program needs to analyze the current data for minimum and maximum values and create a chart scale based on those values. Auto-scaling, and the creation of appropriate axes, with endpoints at even values, and well-rounded major and minor tick mark spacing, is quite complicated. The **AutoScale** classes provide tools that make automatic generation of charts easier.

AutoScale	This class is the abstract base class for the auto-scale classes.
LinearAutoScale	This class is a concrete implementation of the **AutoScale** class. It calculates scaling values based on the numeric values in **SimpleDataset** and **GroupDataset** objects. Linear scales and axes use it for auto-scale calculations.
LogAutoScale	This class is a concrete implementation of the **AutoScale** class. It calculates scaling values based on the numeric values in **SimpleDataset** and **GroupDataset** objects. Logarithmic scales and axes use it for auto-scale calculations.
TimeAutoScale	This class is a concrete implementation of the **AutoScale** class. It calculates scaling values based on the **Date** values in **TimeSimpleDataset** and **TimeGroupDataset** objects. Date/time scales and axes use it for auto-scale calculations.
ElapsedTimeAutoScale	This class is a concrete implementation of the **AutoScale** class. It calculates scaling values based on the numeric values in **ElapsedTimeSimpleDataset** and **ElapsedTimeGroupDataset** objects. The elapsed time classes use it for auto-scale calculations.
EventAutoScale	This class is a concrete implementation of the **AutoScale**class. It calculates scaling values based on the numeric values in **EventSimpleDataset**and **EventGroupDataset**objects. The event classes use it for auto-scale calculations.

Chart Object Classes

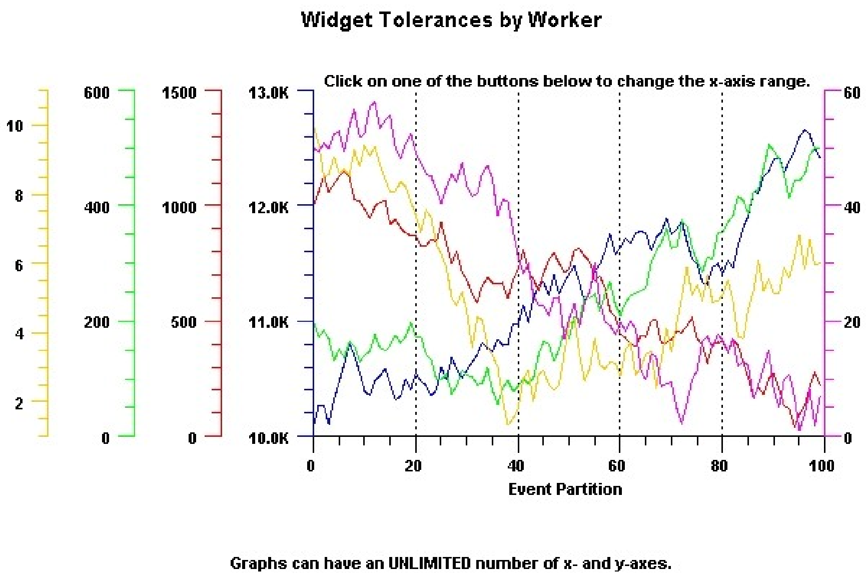
Chart objects are graph objects that can be rendered in the current graph window. This is in comparison to other classes that are purely calculation classes, such as the coordinate conversion classes. All chart objects have certain information in common. This includes instances of **ChartAttribute** and **PhysicalCoordinates** classes. The **ChartAttribute** class contains basic color, line style, and gradient information for the object, while the **PhysicalCoordinates** maintains the coordinate system used by object. The majority of classes in the library derive from the **GraphObj** class, each class a specific charting object such as an axis, an axis label, a simple plot or a group plot. Add **GraphObj** derived objects (axes, plots, labels, title, etc.) to a graph using the **ChartView.addChartObject** method.

GraphObj	This class is the abstract base class for all drawable graph objects. It contains information common to all chart objects. This class includes references to instances of the **ChartAttribute** and **PhysicalCoordinates** classes. The **ChartAttribute** class contains basic color, line style, and gradient information for the object, while the **PhysicalCoordinates** maintains the coordinate system used by object. The majority of classes in the library derive from the **GraphObj** class, each class a specific charting object such as an axis, an axis label, a simple plot or a group plot
Background	This class fills the background of the entire chart, or the plot area of the chart, using a solid color, a color gradient, or a texture.

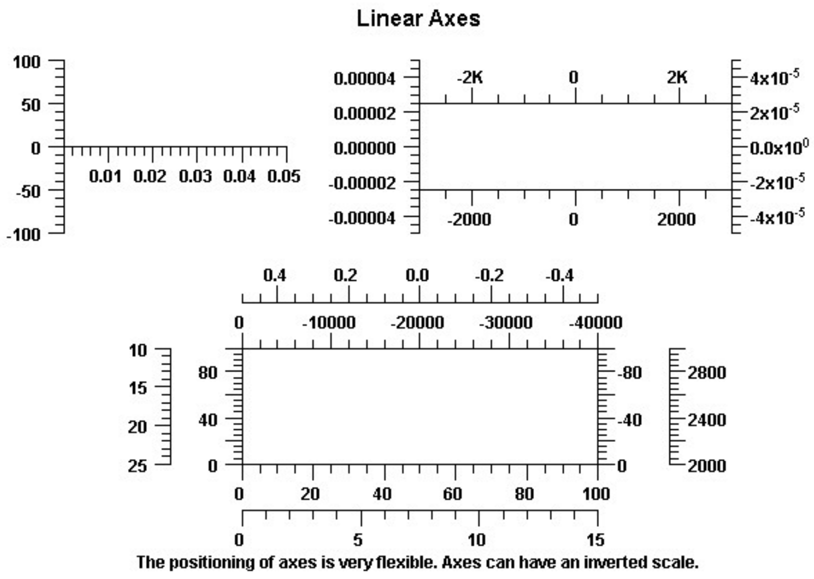
Axis Classes

- **Axis**
  - **LinearAxis**
    - **PolarAxes**
    - **AntennaAxes**
    - **ElapsedTimeAxis**
  - **LogAxis**
  - **TimeAxis**
  - **EventAxis**

Creating a **PhysicalCoordinates** coordinate system does not automatically create a pair of x- and y-axes. Axes are separate charting objects drawn with respect to a specific **PhysicalCoordinates** object. The coordinate system and the axes do not need to have the same limits. In general, the limits of the coordinate system should be greater than or equal to the limits of the axes. The coordinate system may have limits of 0 to 15, while you may want the axes to extend from 0 to 10.

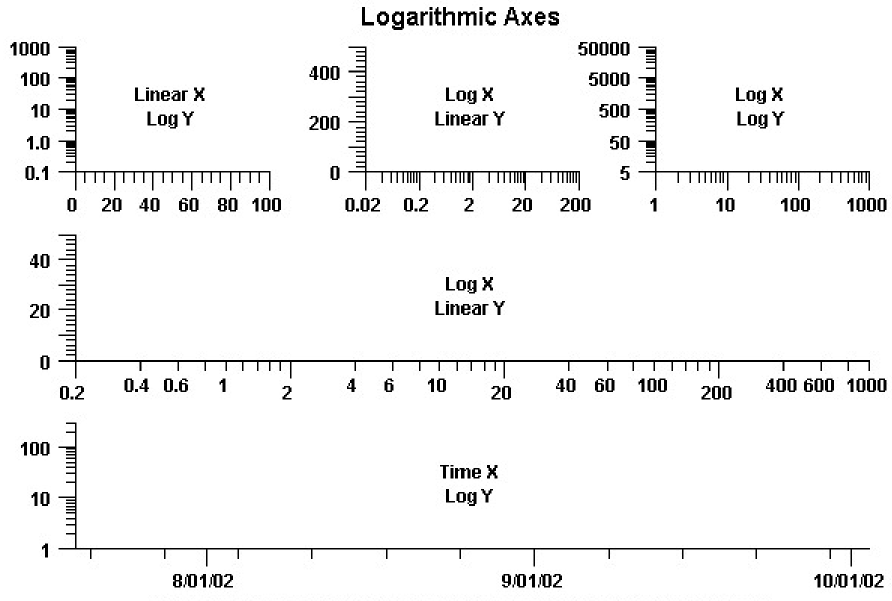


Axis This class is the abstract base class for the other axis classes. It contains data and drawing routines common to all axis classes.



LinearAxis

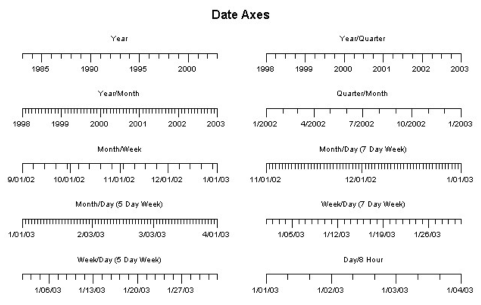
This class implements a linear axis with major and minor tick marks placed at equally spaced intervals.



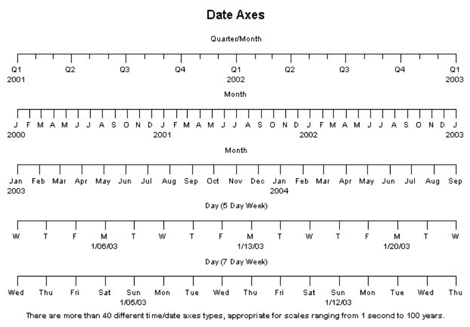
Logarithmic axes can be combined with linear, logarithmic and time axes.

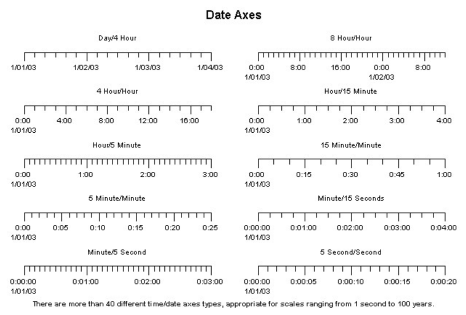
LogAxis

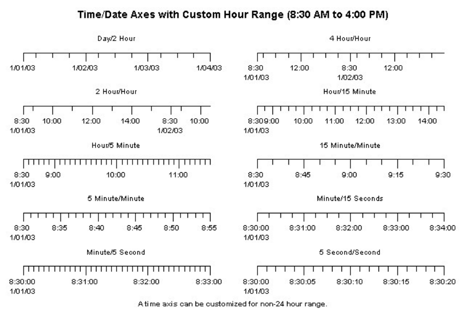
This class implements a logarithmic axis with major tick marks placed on logarithmic intervals, for example 1, 10,100 or 30, 300, 3000. The minor tick marks are placed within the major tick marks using linear intervals, for example 2, 3, 4, 5, 6, 7, 8, 9, 20, 30, 40, 50,.., 90. An important feature of the **LogAxis** class is that the major and minor tick marks do not have to fall on decade boundaries. A logarithmic axis must have a positive range exclusive of 0.0, and the tick marks can represent any logarithmic scale.



There are more than 40 different time/date axes types, appropriate for scales ranging from 1 second to 100 years.

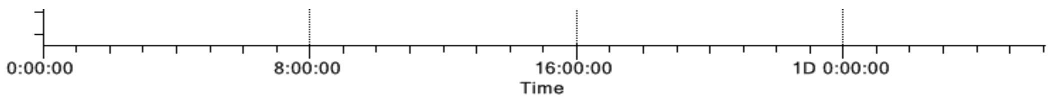






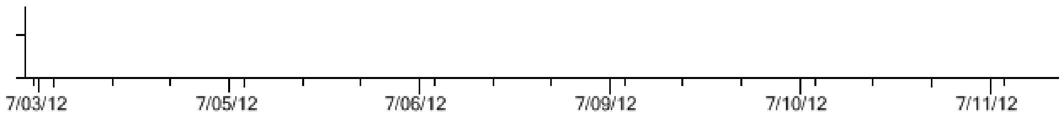
TimeAxis

The time axis supports time scales ranging from milliseonds to hundreds of years. Dates and times are specified using the JavaScript Date class. The major and minor tick marks can fall on any time base, where a time base represents seconds, minutes, hours, days, weeks, months or years. The scale can exclude weekends, for example, Friday, October 20, 2000 is immediately followed by Monday, October 23, 2000. A day can also have a custom range, for example a range of 9:30 AM to 4:00 PM. The chart time axis excludes time outside of this range. This makes the class very useful for the inter-day display of financial market information (stock, bonds, commodities, options, etc.) across several days, months or years.



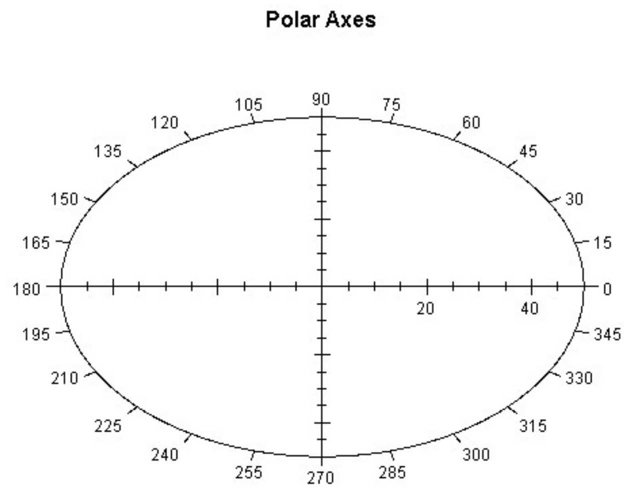
ElapsedTimeAxis

The elapsed time axis is very similar to the linear axis and is subclassed from that class. The main difference is the major and minor tick mark spacing calculated by the calcAutoAxis method takes into account the base 60 of seconds per minute and minutes per hour, and the base 24 of hours per day. It is a continuous linear scale.



EventAxis

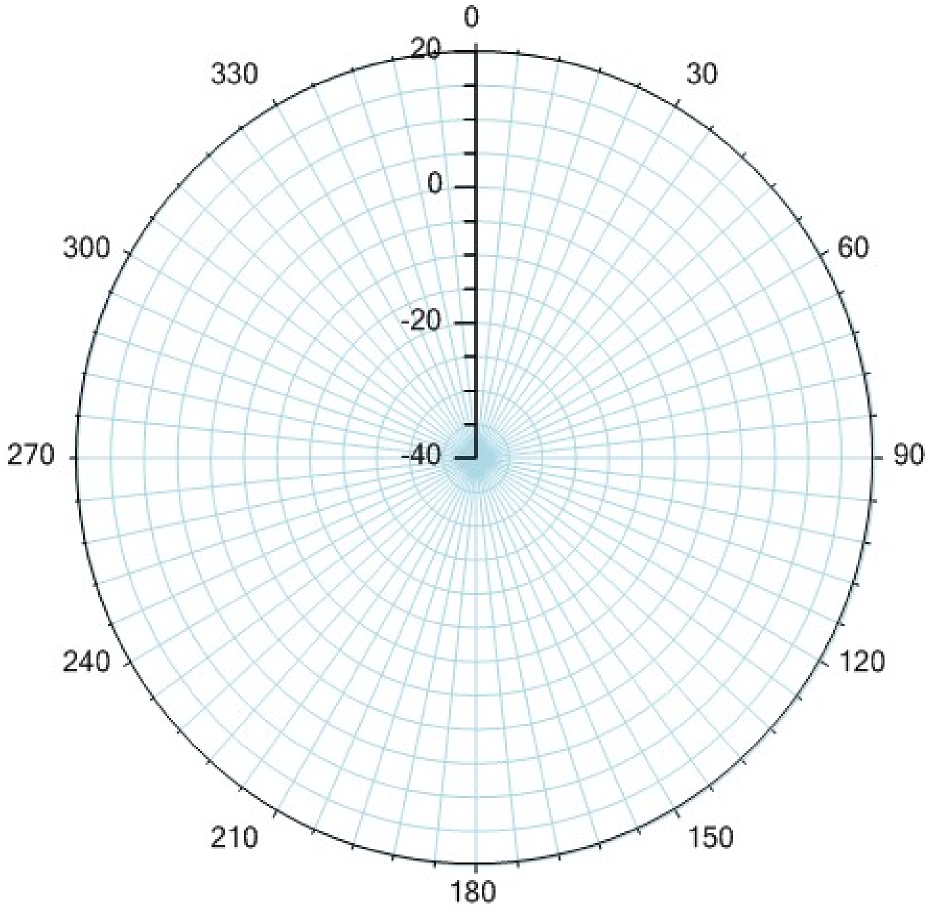
The event axis is a hybrid of the a time axis and the linear axis. It places major and minor tick marks on the axis, based on the event data attached to the coordinate system. Every ChartEvent object in the coordinate system does not necessarily have a tick mark, because where the data values are bunched together, this would create too many tick marks and they would overlap. Every tick mark, though, will have at least one ChartEvent object associated with it. In the case where multiple plots have ChartEvent objects with the same time stamp, a tick mark can have multiple ChartEvent objects associated with it.



A polar axis consists of the x and y axis for magnitude, and the outer circle for the angle.

PolarAxes

This class has three separate axes: two linear and one circular. The two linear axes, scaled for +- the magnitude of the polar scale, form a cross with the center of both axes at the origin (0, 0. The third axis is a circle centered on the origin with a radius equal to the magnitude of the polar scale. This circular axis represents 360 degrees (or 2 Pi radians) of the polar scale and the tick marks that circle this axis are spaced at equal degree intervals.



AntennaAxes This class has two axes: one linear y-axis and one circular axis. The linear axis is scaled for the desired range of radius values. This can extend from minus values to plus values. The second axis is a circle centered on the origin with a radius equal to the range of the radius scale. This circular axis represents 360 degrees of the antenna scale and the tick marks that circle this axis are spaced at equal degree intervals.

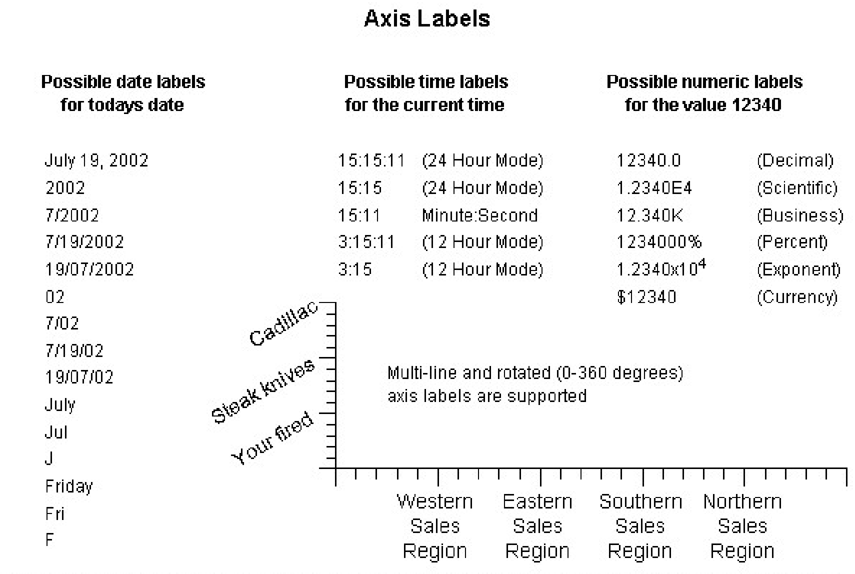
Axis Label Classes

AxisLabels

- **NumericAxisLabels**
- **StringAxisLabels**
- **PolarAxesLabels**
- **AntennaAxesLabels**
- **TimeAxisLabels**

ElapsedTimeAxisLabels
EventAxisLabels

Axis labels inform the user of the x- and y-scales used in the chart. The labels center on the major tick marks of the associated axis. Axis labels are usually numbers, times, dates, or arbitrary strings.



Axis Labels

Possible date labels for todays date	Possible time labels for the current time		Possible numeric labels for the value 12340	
July 19, 2002	15:15:11	(24 Hour Mode)	12340.0	(Decimal)
2002	15:15	(24 Hour Mode)	1.2340E4	(Scientific)
7/2002	15:11	Minute:Second	12.340K	(Business)
7/19/2002	3:15:11	(12 Hour Mode)	1234000%	(Percent)
19/07/2002	3:15	(12 Hour Mode)	1.2340x10^4	(Exponent)
02			$12340	(Currency)
7/02				
7/19/02				
19/07/02				
July				
Jul				
J				
Friday				
Fri				
F				

In addition to the predefined formats, programmers can define custom time, date and numeric formats.

AxisLabels	This class is the abstract base class for all axis label objects. It places numeric labels, date/time labels, or arbitrary text labels, at the major tick marks of the associated axis object. In addition to the standard font options (type, size, style, color, etc.), axis label text can be rotated 360 degrees in one degree increments.
NumericAxisLabels	This class labels the major tick marks of the **LinearAxis**, and **LogAxis** classes. The class supports many predefined and user-definable formats, including numeric, exponent, percentage, business and currency formats.
StringAxisLabels	This class labels the major tick marks of the **LinearAxis**, and **LogAxis** classes using user-defined strings.
TimeAxisLabels	This class labels the major tick marks of the associated **TimeAxis** object. The class supports many time (23:59:59) and date (5/17/2001) formats. It is also possible to define custom date/time formats.

ElapsedTimeAxisLabels	This class labels the major tick marks of the associated **ElapsedTimeAxis** object. The class supports HH:MM:SS and MM:SS formats, with decimal seconds out to 0.00001, i.e. "12:22:43.01234". It also supports a cumulative hour format (101:51:22), and a couple of day formats (4.5:51:22, 4D 5:51:22).
PolarAxesLabels	This class labels the major tick marks of the associated **PolarAxes** object. The x-axis is labeled from 0.0 to the polar scale magnitude, and the circular axis is labeled counter clockwise from 0 to 360 degrees, starting at 3:00.
AntennaAxesLabels	This class labels the major tick marks of the associated **AntennaAxes** object. The y-axis is labeled from the radius minimum to the radius maximum. The circular axis is labeled clockwise from 0 to 360 degrees, starting at 12:00.
EventAxisLabels	This class labels the major tick marks of the associated **EventAxis** object. The class supports many time (23:59:59) and date (5/17/2001) formats. It is also possible to define custom date/time formats.

Chart Plot Classes

ChartPlot
- **GroupPlot**
- **PieChart**
- **PolarPlot**
- **AntennaPlot**
- **SimplePlot**

Plot objects are objects that display data organized in a **ChartDataset** class. There are six main categories: simple, group, polar, antenna, and pie plots. Simple plots graph data organized as a simple set of xy data points. The most common examples of simple plots are line plots, bar graphs, scatter plots and line-marker plots. Group plots graph data organized as multiple y-values for each x-value. The most common examples of group plots are stacked bar graphs, open-high-low-close plots, candlestick plots, floating stacked bar plots and "box and whisker" plots. Polar charts plot data organized as a simple set of data points, where each data point represents a polar magnitude and angle pair, rather than xy Cartesian coordinate values. The most common example of polar charts is the display of complex numbers (a + b*i*), and it is used in many engineering disciplines. Antenna charts plot data organized as a simple set of data points, where each data point represents a radius value and angle pair, rather than xy Cartesian coordinate values. The most common example of antenna charts is the display of antenna performance and specification graphs. The last plot object category is the pie chart, were a pie wedge represents each data value. The size of the pie wedge is proportional to the fraction (data value / sum of all data values).

ChartPlot	This class is the abstract base class for chart plot objects. It contains a reference to a **ChartDataset** derived class containing the data associated with the plot.

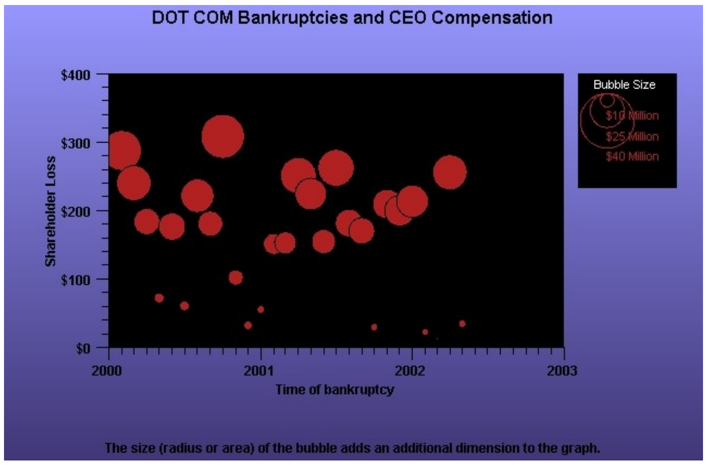
Group Plot Classes

GroupPlot

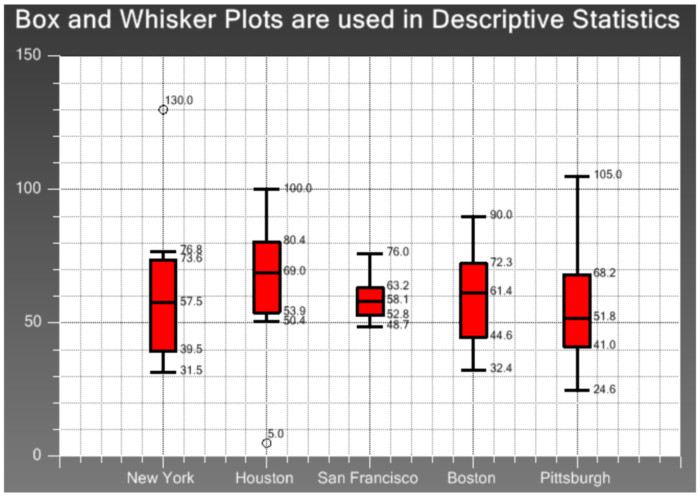
BoxWhiskerPlot
BubblePlot
CandlestickPlot
CellPlot
ErrorBarPlot
FloatingBarPlot
FloatingStackedBarPlot
HistogramPlot
LineGapPlot
MultiLinePlot
OHLCPlot
StackedBarPlot
StackedLinePlot
GroupVersaPlot

Group plots use data organized as arrays of x- and y-values, where there is one or more y for every x.. Group plot types include multi-line plots, stacked line plots, stacked bar plots, group bar plots, error bar plots, floating bar plots, floating stacked bar plots, open-high-low-close plots, candlestick plots, arrow plots, histogram plots, cell plots, “box and whisker” plots, and bubble plots.

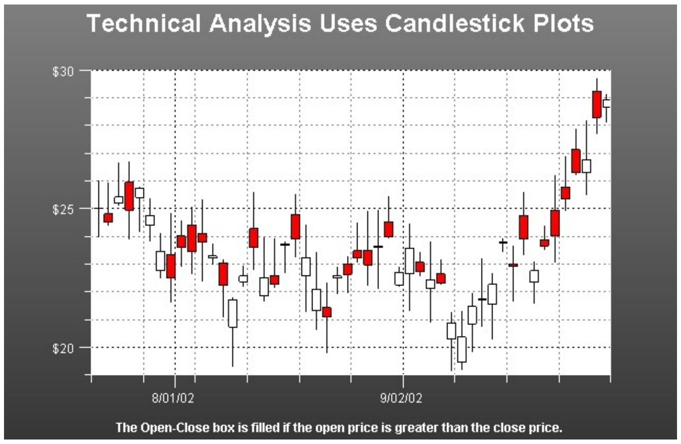
GroupPlot	This class is an abstract base class for all group plot classes.



BubblePlot This class is a concrete implementation of the **GroupPlot** class and displays bubble plots. The values in the dataset specify the position and size of each bubble in a bubble chart.

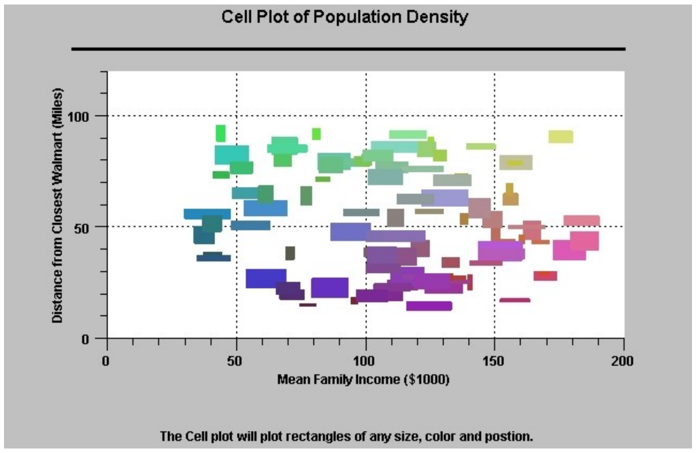


BoxWhiskerPlot This class is a concrete implementation of the **GroupPlot** class and displays box and whisker plots. The BoxWhiskerPlot class graphically depicts groups of numerical data through their five-number summaries (the smallest observation, lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation).



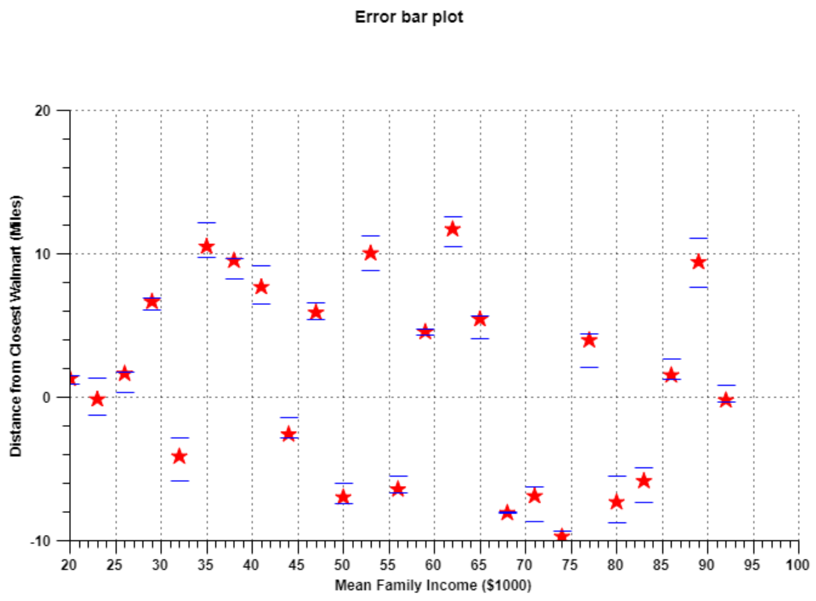
CandlestickPlot

This class is a concrete implementation of the **GroupPlot** class and displays stock market data in an open-high-low-close format common in financial technical analysis.

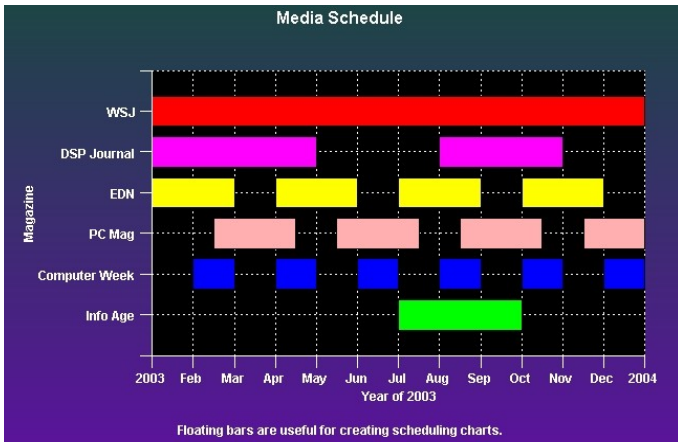


CellPlot

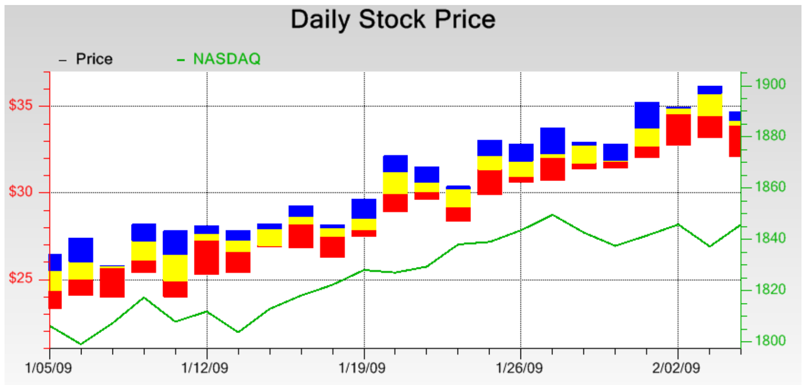
This class is a concrete implementation of the **GroupPlot** class and displays cell plots. A cell plot is a collection of rectangular objects with independent positions, widths and heights, specified using the values of the associated group dataset.



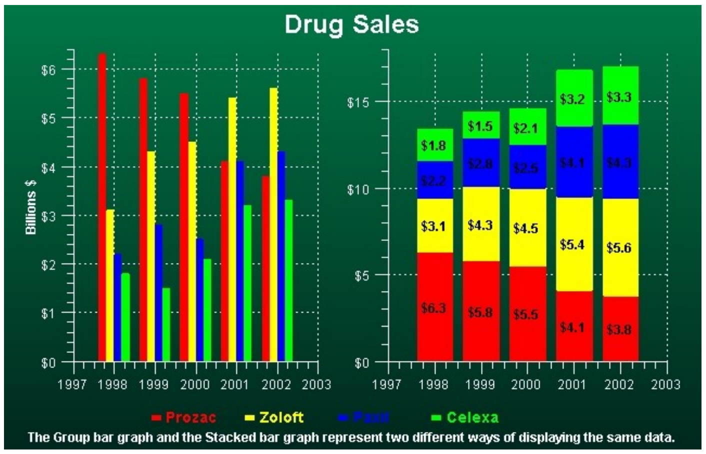
ErrorBarPlot This class is a concrete implementation of the **GroupPlot** class and displays error bars. Error bars are two lines positioned about a data point that signify the statistical error associated with the data point



FloatingBarPlot This class is a concrete implementation of the **GroupPlot** class and displays free-floating bars in a graph. The bars are free floating because each bar does not reference a fixed base value, as do simple bar plots, stacked bar plots and group bar plots.

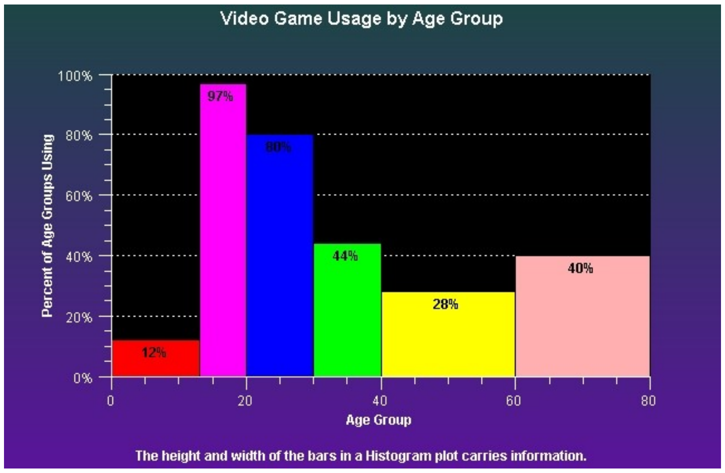


FloatingStackedBarPlot This class is a concrete implementation of the **GroupPlot** class and displays free-floating stacked bars. The bars are free floating because each bar does not reference a fixed base value, as do simple bar plots, stacked bar plots and group bar plots.

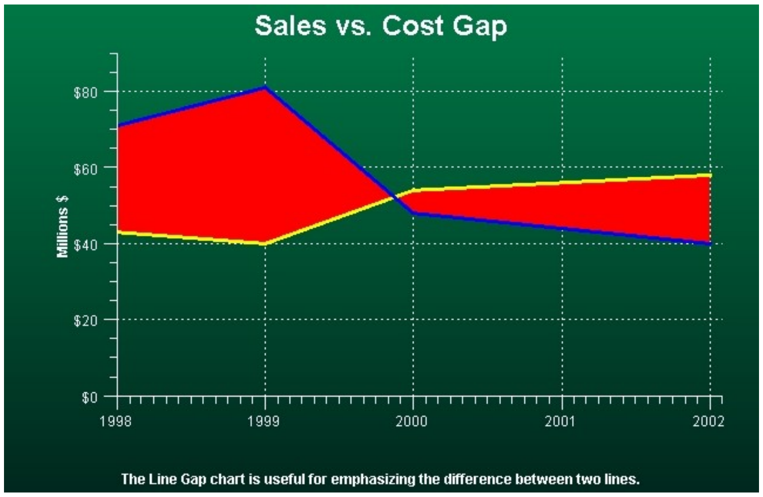


GroupBarPlot This class is a concrete implementation of the **GroupPlot** class and displays group data in a group bar format. Individual bars, the height of which corresponds to the group y-values of the dataset, display side by side, as a group, justified with respect to the x-position value for each group. The group bars share a common base value.

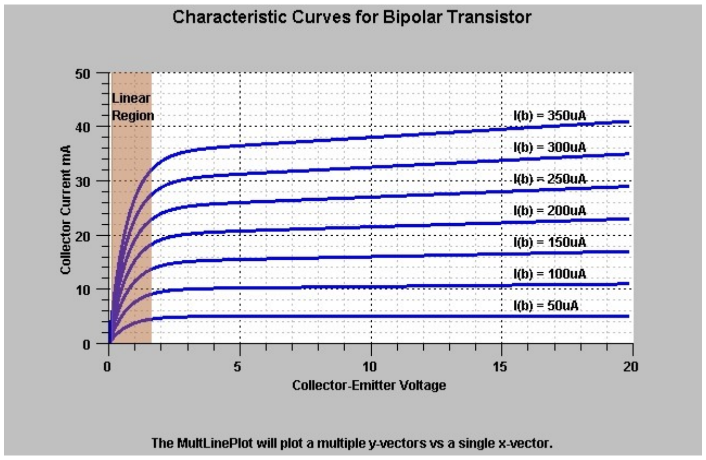
StackedBarPlot This class is a concrete implementation of the **GroupPlot** class and displays data as stacked bars. In a stacked bar plot each group is stacked on top of one another, each group bar a cumulative sum of the related group items before it.



HistogramPlot This class is a concrete implementation of the **GroupPlot** class and displays histogram plots. A histogram plot is a collection of rectangular objects with independent widths and heights, specified using the values of the associated group dataset. The histogram bars share a common base value.

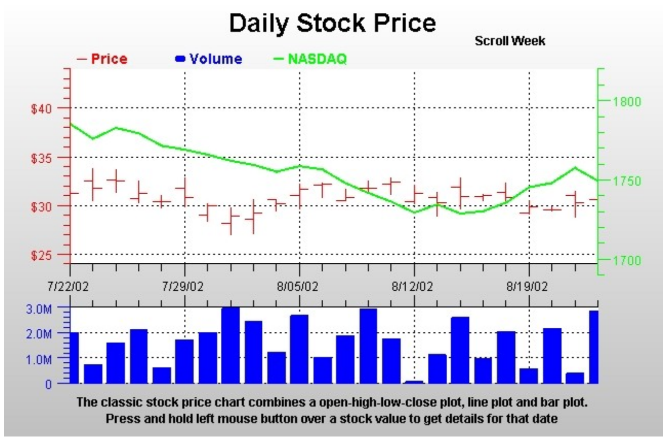


LineGapPlot This class is a concrete implementation of the **GroupPlot** class. A line gap chart consists of two lines plots where a contrasting color fills the area between the two lines, highlighting the difference.



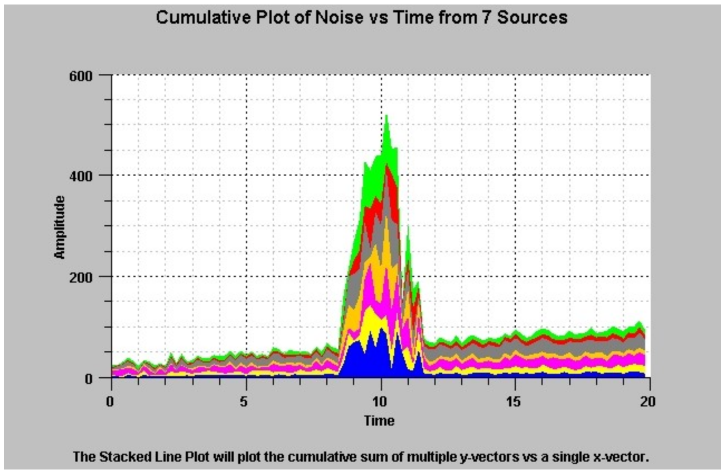
MultiLinePlot

This class is a concrete implementation of the **GroupPlot** class and displays group data in multi-line format. A group dataset with four groups will display four separate line plots. The y-values for each line of the line plot represent the y-values for each group of the group dataset. Each line plot share the same x-values of the group dataset.



OHLCPlot

This class is a concrete implementation of the **GroupPlot** class and displays stock market data in an open-high-low-close format common in financial technical analysis. Every item of the plot is a vertical line, representing High and Low values, with two small horizontal "flags", one left and one right extending from the vertical High-Low line and representing the Open and Close values.



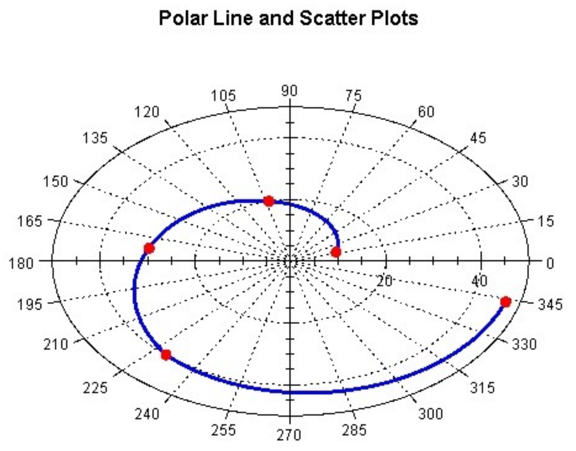
StackedLinePlot This class is a concrete implementation of the **GroupPlot** class and displays data in a stacked line format. In a stacked line plot each group is stacked on top of one another, each group line a cumulative sum of the related group items before it.

Polar Plot Classes

PolarPlot
- **PolarLinePlot**
- **PolarScatterPlot**

Polar plots that use data organized as arrays of x- and y-values, where an x-value represents the magnitude of a point in polar coordinates, and the y-value represents the angle, in radians, of a point in polar coordinates. Polar plot types include line plots and scatter plots.

PolarPlot This class is an abstract base class for the polar plot classes.



The polar line charts use true polar (not linear) interpolation between data points.

PolarLinePlot This class is a concrete implementation of the **PolarPlot** class and displays data in a simple line plot format. The lines drawn between adjacent data points use polar coordinate interpolation.

PolarScatterPlot This class is a concrete implementation of the **PolarPlot** class and displays data in a simple scatter plot format.

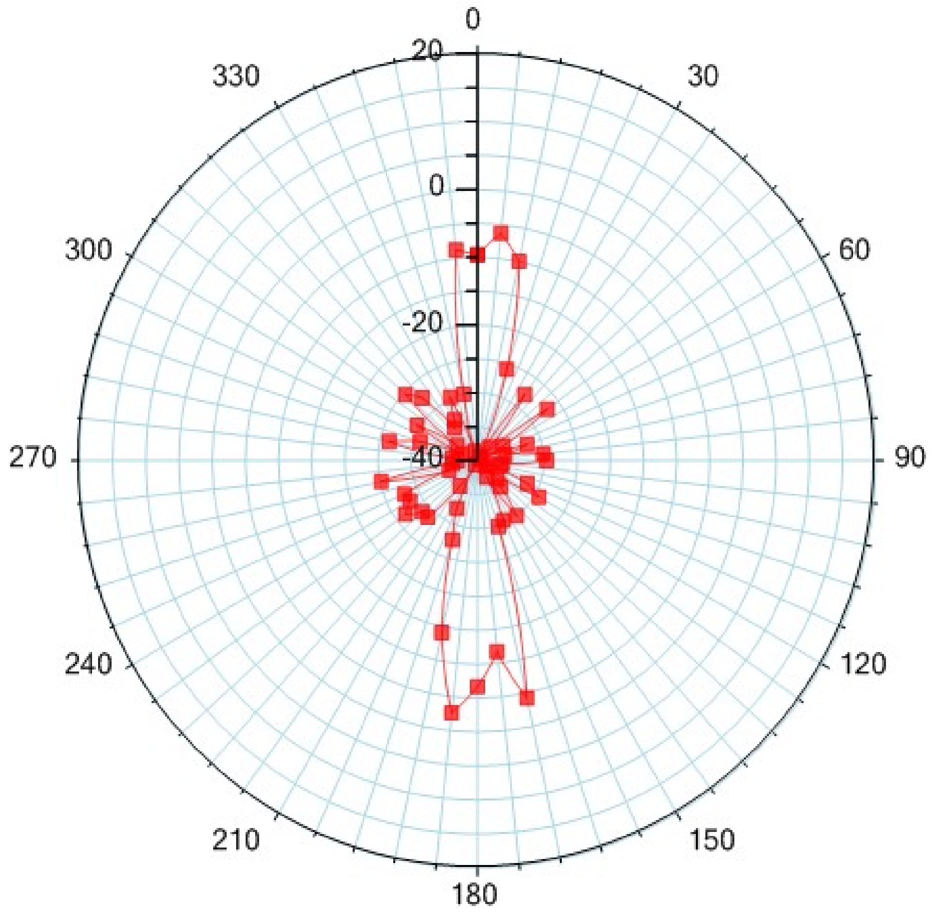
Antenna Plot Classes

AntennaPlot
- **AntennaLinePlot**
- **AntennaScatterPlot**
- **AntennaLineMarkerPlot**

GraphObj
- **AntennaAnnotation**

Antenna plots that use data organized as arrays of x- and y-values, where an x-value represents the radial value of a point in antenna coordinates, and the y-value represents the angle, in degrees, of a point in antenna coordinates. Antenna plot types include line plots, scatter plots, line marker plots, and an annotation class.

AntennaPlot This class is an abstract base class for the polar plot classes.

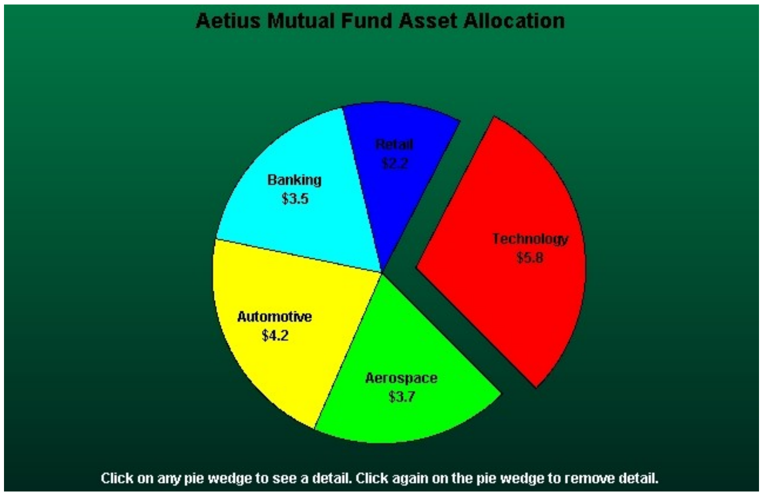


AntennaLineMarkerPlot

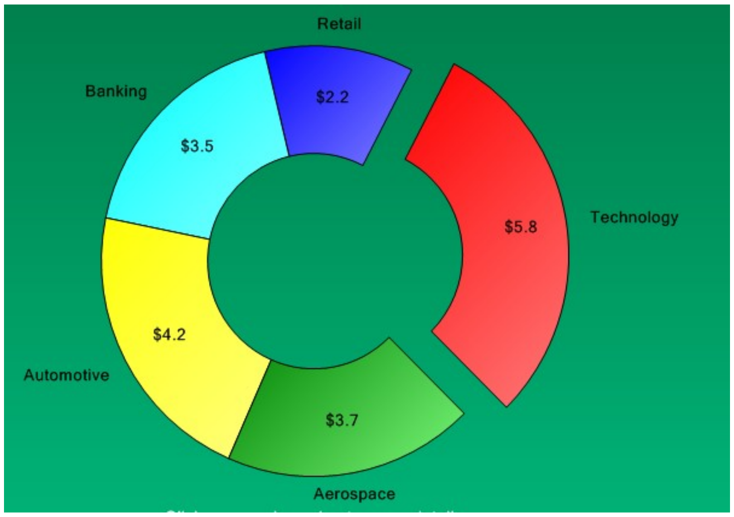
AntennaLinePlot	This class is a concrete implementation of the **AntennaPlot** class and displays data in a simple line plot format. The lines drawn between adjacent data points use antenna coordinate interpolation.
AntennaScatterPlot	This class is a concrete implementation of the **AntennaPlot** class and displays data in a simple scatter plot format.
AntennaLineMarkerPlot	This class is a concrete implementation of the **AntennaPlot** class and displays data in a simple line marker plot format.
AntennaAnnotation	This class is used to highlight, or mark, a specific attribute of the chart. It can mark a constant radial value using a circle, or it can mark a constant angular value using a radial line from the origin to the outer edge of the scale.

Pie and Ring Chart Classes

It uses data organized as arrays of x- and y-values, where an x-value represents the numeric value of a pie wedge, and a y-value specifies the offset (or "explosion") of a pie wedge with respect to the center of the pie.



PieChart The pie chart plots data in a simple pie chart format. It uses data organized as arrays of x- and y-values, where an x-value represents the numeric value of a pie wedge, and a y-value specifies the offset (or "explosion") of a pie wedge with respect to the center of the pie.



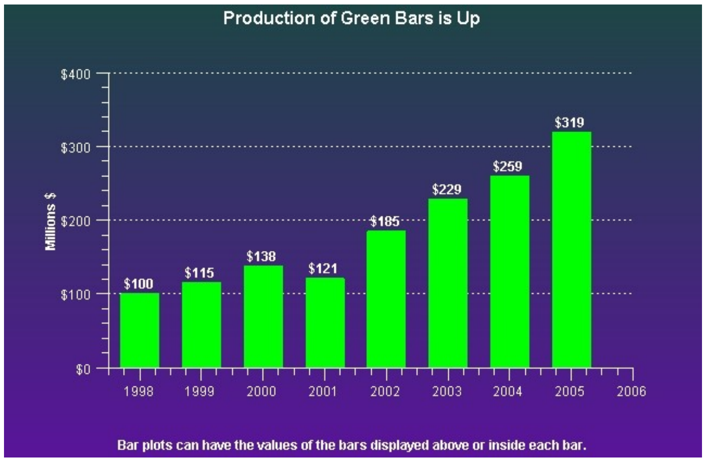
RingChart The ring chart plots data in a modified pie chart format known as a ring chartt. It uses data organized as arrays of x- and y-values, where an x-value represents the numeric value of a ring segment, and a y-value specifies the offset (or "explosion") of a ring segment with respect to the origin of the ring.

Simple Plot Classes

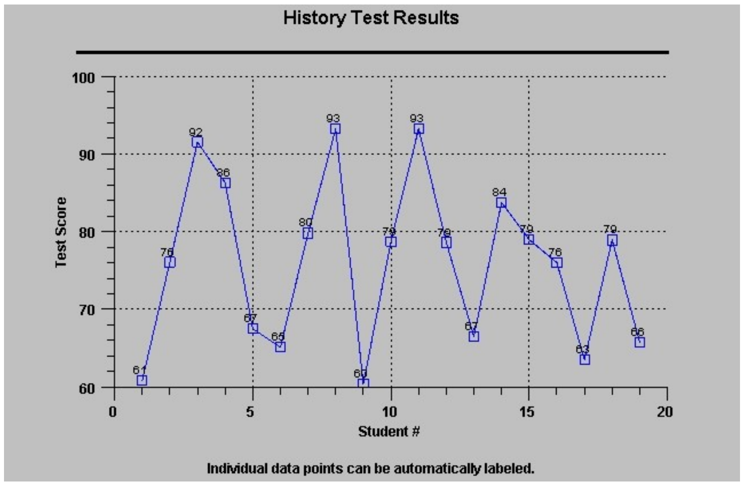
SimplePlot
- **SimpleBarPlot**
- **SimpleLineMarkerPlot**
- **SimpleLinePlot**
- **SimpleScatterPlot**
- **SimpleVersaPlot**

Simple plots use data organized as a simple array of xy points, where there is one y for every x. Simple plot types include line plots, scatter plots, bar graphs, and line-marker plots.

SimplePlot This class is an abstract base class for all simple plot classes.

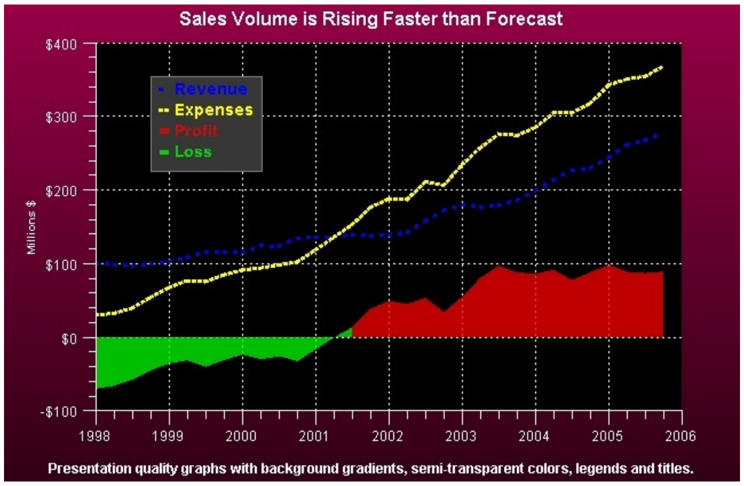


SimpleBarPlot This class is a concrete implementation of the **SimplePlot** class and displays data in a bar format. Individual bars, the maximum value of which corresponds to the y-values of the dataset, are justified with respect to the x-values.

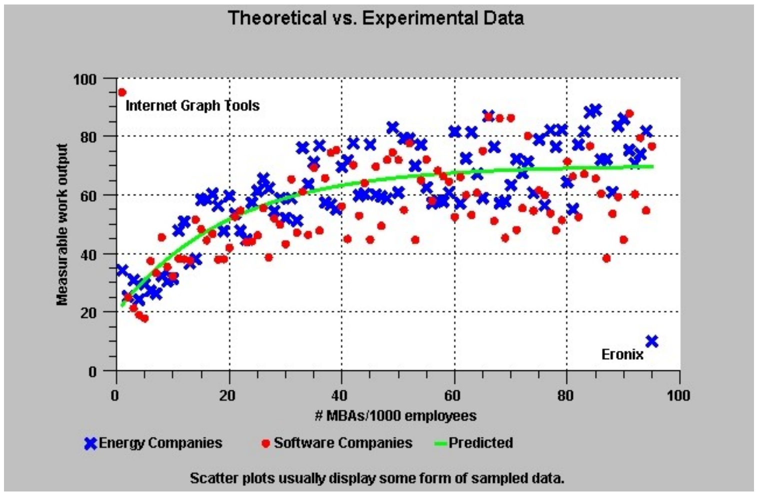


SimpleLineMarkerPlot

This class is a concrete implementation of the **SimplePlot** class and it displays simple datasets in a line plot format where scatter plot symbols highlight individual data points.



SimpleLinePlot This class is a concrete implementation of the **SimplePlot** class it displays simple datasets in a line plot format. Adjacent data points are connected using a straight, or a step line.



SimpleScatterPlot	This class is a concrete implementation of the **SimplePlot** class and it displays simple datasets in a scatter plot format where each data point is represented using a symbol.
SimpleVersaPlot	The **SimpleVersaPlot** is a plot type that can be any of the four simple plot types: LINE_MARKER_PLOT, LINE_PLOT, BAR_PLOT, SCATTER_PLOT. It is used when you want to be able to change from one plot type to another, without deleting the instance of the old plot object and creating an instance of the new.

Legend Classes

LegendItem
BubblePlotLegendItem
Legend
  StandardLegend
  BubblePlotLegend

Legends provide a key for interpreting the various plot objects in a graph. It organizes a collection of legend items, one for each plot objects in the graph, and displays them in a rectangular frame.

Legend	This class is the abstract base class for chart legends.
LegendItem	This class is the legend item class for all plot objects except for bubble plots. Each legend item manages one symbol and descriptive text for that

	symbol. The **StandardLegend** class uses objects of this type as legend items.
BubblePlotLegendItem	This class is the legend item class for bubble plots. Each legend item manages a circle and descriptive text specifying the value of a bubble of this size. The **BubblePlotLegend** class uses objects of this type as legend items.
StandardLegend	This class is a concrete implementation of the **Legend** class and it is the legend class for all plot objects except for bubble plots. The legend item objects display in a row or column format. Each legend item contains a symbol and a descriptive string. The symbol normally associates the legend item to a particular plot object, and the descriptive string describes what the plot object represents.
BubblePlotLegend	This class is a concrete implementation of the **Legend** class and it is a legend class used exclusively with bubble plots. The legend item objects display as offset, concentric circles with descriptive text giving the key for the value associated with a bubble of this size.

Grid Classes

Grid
- **PolarGrid**
- **AntennaGrid**

Grid lines are perpendicular to an axis, extending the major and/or minor tick marks of the axis across the width or height of the plot area of the chart.

Grid	This class defines the grid lines associated with an axis. Grid lines are perpendicular to an axis, extending the major and/or minor tick marks of the axis across the width or height of the plot area of the chart. This class works in conjunction with the **LinearAxis**, **LogAxis, EventAxis** and **TimeAxis** classes.
PolarGrid	This class defines the grid lines associated with a polar axis. A polar chart grid consists of two sets of lines. The first set is a group of concentric circles, centered on the origin and passing through the major and/or minor tick marks of the polar magnitude horizontal and vertical axes. The second set is a group of radial lines, starting at the origin and extending to the outermost edge of the polar plot circle, passing through the major and

	minor tick marks of the polar angle circular axis. This class works in conjunction with the **PolarAxes** class.
AntennaGrid	Analogous to the **PolarGrid**, this class draws radial, and circular grid lines for an Antenna chart.

Chart Text Classes

ChartText
- **ChartTitle**
- **AxisTitle**
- **ChartLabel**
  - **NumericLabel**
  - **TimeLabel**
  - **StringLabel**
  - **ElapsedTimeLabel**

The chart text classes draw one or more strings in the chart window. Different classes support different numeric formats, including floating point numbers, date/time values and multi-line text strings. International formats for floating point numbers and date/time values are also supported.

ChartText	This class draws a string in the current chart window. It is the base class for the **ChartTitle**, **AxisTitle** and **ChartLabel** classes. The **ChartText** class also creates independent text objects. Other classes that display text also use it internally.
ChartTitle	This class displays a text string as the title or footer of the chart.
AxisTitle	This class displays a text string as the title for an axis. The axis title position is outside of the axis label area. Axis titles for y-axes are rotated 90 degrees.
ChartLabel	This class is the abstract base class of labels that require special formatting.
NumericLabel	This class is a concrete implementation of the **ChartLabel** class and it displays formatted numeric values.

TimeLabel	This class is a concrete implementation of the **ChartLabel** class and it displays formatted **ChartCalendar** dates.
ElapsedTimeLabel	This class is a concrete implementation of the **ChartLabel** class and it displays numeric values formatted as elapsed time strings **(12:32:21).**
StringLabel	This class is a concrete implementation of the **ChartLabel** class that formats string values for use as axis labels.

Miscellaneous Chart Classes

Marker
ChartImage
ChartShape
ChartSymbol

Various classes are used to position and draw objects that can be used as standalone objects in a graph, or as elements of other plot objects.

Marker	This class displays one of five marker types in a graph. The marker is used to create data cursors, or to mark data points.
ChartImage	This class encapsulates an **HTMLImageElement** image object, defining a rectangle in chart coordinates that the image is placed in. Which image formats are supported is browser dependent, but it looks like modern browsers support at least JPEG, PNG, BMP, SVG, and ICO.
ChartShape	This class encapsulates a **GPath** object, placing the shape in a chart using a position defined in chart coordinates. A chart can display any object that can be defined using GPath class, mostly line and simple closed polyons.
ChartSymbol	This class defines symbols used by the **SimplePlot** scatter plot functions. Pre-defined symbols include square, triangle, diamond, cross, plus, star, line, horizontal bar, vertical bar, 3D bar and circle.

Mouse Interaction Classes

MouseListener
- **MoveObj**
- **FindObj**
- **DataToolTip**
- **DataCursor**
  - **MoveData**
- **MoveCoordinates**
- **MultiMouseListener**
- **ChartZoom**

Several classes implement delegates for mouse/touch events. The **MouseListener** class implements a generic interface for managing mouse/touch events in a graph window. The **DataCursor**, **MoveData**, **MoveObj, ChartZoom and MoveCoordinates** classes also implement mouse event delegates that use the mouse to mark, move and zoom chart objects and data.

Note: We use the term mouse throughout the manual when it would be more accurate to use mouse/touch. In order to simplify programming for mouse applications (desktop) and touch applications (phone and tablet) our **MouseListener** class processes both mouse and touch events in one core set of event handlers (onMouseDown, onMouseMove, onMouseUp, and onClick events). That way you/we don't have to constantly write two sets of code, one for mouse driven applications and one for touch driven applications. While the software has some button processing built-in for left, right and middle buttons, in order to maintain compatibility with touch devices, programmers should refrain for using anything other than the left button. The right button is hijacked by the browser for context-sensitive menu options and we found no straightforward means of disabling it that works across all browser.

MouseListener	This class implements delegates that trap generic mouse/touch events (button events and mouse motion events) that take place in a **ChartView** window. A programmer can derive a class from **MouseListener** and override the methods for mouse/touch events, creating a custom version of the class.
MoveObj	This class extends the **MouseListener** class and it can select chart objects and move them. Moveable chart objects include axes, axes labels, titles, legends, arbitrary text, shapes and images. Use the **MoveData** class to move objects derived from **SimplePlot**.
FindObj	This class extends the **MouseListener** class, providing additional methods that selectively determine what graphical objects intersect the mouse cursor.
DataCursor	This class combines the **MouseListener** class and **Marker** class. Press a mouse button and the selected data cursor (horizontal and/or vertical line, cross hairs, or a small box) appears at the point of the mouse cursor. The

	data cursor tracks the mouse motion as long as the mouse button is pressed. Release the button and the data cursor disappears. This makes it easier to line up the mouse position with the tick marks of an axis.
MoveData	This class selects and moves individual data points of an object derived from the **SimplePlot** class.
DataToolTip	A data tooltip is a popup box that displays the value of a data point in a chart. The data value can consist of the x-value, the y-value, x- and y-values, group values and open-high-low-close values, for a given point in a chart.
ChartZoom	This class implements mouse controlled zooming for one or more simultaneous axes. The user starts zooming by holding down a mouse button with the mouse cursor in the plot area of a graph. The mouse is dragged and then released. The rectangle established by mouse start and stop points defines the new, zoomed, scale of the associated axes. Zooming has many different modes. Some of the combinations are: • One x or one y axis • One x and one y axes • One x and multiple y axes • One y and multiple x axes • Multiple x and y axes
MoveCoordinates	This class extends the **MouseListener** class and it can move the coordinate system of the underlying chart, analogous to moving (chaging the coordinates of) an internet map by “grabbing” it with the mouse and dragging it.
MultiMouseListener	This class is used by the **ChartView** class to support multiple mouse listeners at the same time.

Miscellaneous Utility Classes

ChartCalendar

CSV
ChartDimension
ChartPoint2D
GroupPoint2D
DoubleArray
DoubleArray2D
BoolArrayNearestPointData
TickMark
ChartRectangle2D

ChartCalendar	This class contains utility routines used to process **Date** date objects.
CSV	This is a utility class for reading and writing CSV (Comma Separated Values) strings.
ChartDimension	This is a utility class for handling dimension (height and width) information using doubles, rather than the integers used by the Size class.
ChartPoint2D	This class encapsulates an xy pair of values as doubles (more useful in this software than the JavaScript Point class.
GroupPoint2D	This class encapsulates an x-value, and an array of y-values, representing the x and y values of one column of a group data set.
DoubleArray	This class is used as an alternative to the standard JavaScript/TypeScript Array class, adding routines for resizing of the array, and the insertion and deletion of double based data elements.
DoubleArray2D	This class is used as an alternative to the standard JavaScript/TypeScript 2D Array class, adding routines for resizing of the array, and the insertion and deletion of double based data elements.
BoolArray	This class is used as an alternative to the standard JavaScript/TypeScript Array class, adding routines for resizing of the array, and the insertion and deletion of bool based data elements.
NearestPointData	This is a utility class for returning data that results from nearest point calculations.
TickMark	The axis classes use this class to to organize the location of the individual tick marks of an axis.
ChartRectangle2D	This is a utility class that extends the RectangleF class, using doubles as internal storage.

A diagram depicts the class hierarchy of the QCChart2D for JavaScript/TypeScript library.

- ChartObj $\longrightarrow$ BoolArray
    - Arrow
    - ChartCalendar
    - ChartEvent
    - CSV
    - ChartDimension
    - NearestPointData
    - ChartScale
        - LinearScale
        - LogScale
        - TimeScale
        - ElapsedTimeScale
        - EventScale
    - UserCoordinates
        - WorldCoordinates
            - WorkingCoordinates
                - PhysicalCoordinates
                    - CartesianCoordinates
                        - PolarCoordinates
                        - AntennaCoordinates
                        - EventCoordinates
                    - TimeCoordinates
                    - ElapsedTimeCoordinates
    - ChartDataset
        - SimpleDataset
            - TimeSimpleDataset
            - ElapsedTimeSimpleDataset
            - EventSimpleDataset
        - GroupDataset
            - TimeGroupDataset
            - ElapsedTimeGroupDataset
            - EventGroupDataset
    - AutoScale
        - LinearAutoScale
        - LogAutoScale
        - TimeAutoScale
        - ElapsedTimeAutoScale
        - EventAutoScale
    - MouseListener
        - MoveObj
        - FindObj
        - DataToolTip
        - ChartZoom
        - MoveCoordinates
        - MultiMouseListener
        - DataCursor
            - MoveData
- ChartAttribute
- ChartGradient
- BufferedImage

System.Windows.Forms.UserControl
ChartView

ChartRectangle2D
ChartPoint2D
GroupPoint2D
DoubleArray
DoubleArray2D

- GraphObj
  - AntennaAnnotation
  - TickMark
  - Axis
    - LinearAxis
      - PolarAxes
      - AntennaAxes
    - LogAxis
    - TimeAxis
    - ElapsedTimeAxis
    - EventAxis
  - ChartText
    - ChartTitle
    - AxisTitle
    - ChartLabel
      - NumericLabel
        - BarDatapointValue
      - TimeLabel
      - ElapsedTimeLabel
      - StringLabel
    - AxisLabels
      - NumericAxisLabels
      - TimeAxisLabels
      - ElapsedTimeAxisLabels
      - StringAxisLabels
      - PolarAxesLabels
      - AntennaAxesLabels
      - EventAxisLabels
  - Grid
    - PolarGrid
    - AntennaGrid
  - LegendItem
  - BubblePlotLegendItem
  - Legend
    - StandardLegend
    - BubblePlotLegend
  - ChartPlot
    - SimplePlot
      - SimpleLinePlot
      - SimpleBarPlot
      - SimpleScatterPlot
      - SimpleLineMarkerPlot
      - SimpleVersaPlot
    - GroupPlot
      - BubblePlot
      - CandlestickPlot
      - CellPlot
      - ErrorBarPlot
      - FloatingBarPlot
      - FloatingStackedBarPlot
      - GroupBarPlot
      - HistogramPlot
      - LineGapPlot
      - MultiLinePlot
      - OHLCPlot
      - StackedBarPlot
      - StackedLinePlot
      - BoxWhiskerPlot
    - PieChart
    - RingChart
    - PolarPlot
      - PolarLinePlot
      - PolarScatterPlot
    - AntennaPlot
      - AntennaLinePlot
      - AntennaScatterPlot
      - AntennaLineMarkerPlot
  - Background
  - ChartImage
  - ChartShape
  - ChartSymbol
  - Marker
  - ChartZoom

3. Chart Datasets

ChartDataset
- **SimpleDataset**
  - **TimeSimpleDataset**
  - **ElapsedTimeSimpleDataset**
  - **EventSimpleDataset**
- **GroupDataset**
  - **TimeGroupDataset**
  - **ElapsedTimeGroupDataset**
  - **EventGroupDataset**

The dataset classes organize the numeric data associated with a plot object. Plot objects are chart objects derived from the **ChartPlot** class. There are two major types of data supported by the dataset classes. The first is simple xy data, where for every x-value there is one y-value. The second data type is group data, where every x-value can have one or more y-values. A couple of variants of the simple xy datasets include a simple dataset type that can substitute **Date** values, or ChartTimeSpan as the x- or y-values. And there is a simple and group datasets which can plot ChartEvent data.

Except in the case of the ChartEvent datasets (EventSimpleDataset and EventGroupDatast), copies of the original data arrays are stored. The original source data can be deleted once the dataset is created. If you want to make any changes to the data, you must change the data in the dataset, not the original source data. The ChartEvent datasets are different. Because they contain an array of ChartEvent objects, and these objects can be quite large, a copy of the ChartEvent objects is NOT made. Instead, the dataset classes reference the ChartEvent objects passed into the constructor.

Datasets can be initialized using CSV (comma separated value) text strings. The CSV text string is a common format that can share data between spreadsheets, databases and word processing programs. Datasets can also write CSV strings, loadable into other programs.

If you need to plot data stored in a database, use database calls to retrieve the x and y data values. If you retrieve them as individual x and y data values, use them to initialize x and y data arrays. Or, if you retrieve the data as x and y arrays, you can use those to initialize a dataset object.

The **ChartDataset** class is the abstract base class for all of the dataset classes. It contains data common to all dataset classes, such as the x-value array, the number of x-values, the dataset name and the dataset type.

Simple Numeric Dataset

Class SimpleDataset

```
ChartObj
     |
     +--ChartDataset
            |
            +--SimpleDataset
```

The **SimpleDataset** class represents simple floating point xy data, where for every x-value there is one y-value. The number of xy data points in a simple dataset is referred to as the number of columns, or as the property **numberDatapoints** Think of a spreadsheet file that looks like:

x-values	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]
y-values	y[0]	y[1]	y[2]	y[3]	y[4]	y[5]

number of xy data pairs = numberDatapoints = numberColumns = 6

This would be the ROW_MAJOR format if the data were stored in a CSV format.

SimpleDataset constructors

```
public static newSimpleDataset(sname: string, x: number[], y: number[]): SimpleDataset

public static newSimpleDatasetValid(sname: string, x: number[], y: number[], valid: boolean[]):
SimpleDataset

public static newSimpleDatasetValidN(sname: string, x: number[], y: number[], valid: boolean[],
n: number): SimpleDataset

public static newSimpleDatasetCSV(csv: CSV,  source: string, rowskip : number, columnskip :
number ): SimpleDataset
```

sname	Specifies the name of the dataset.
x	An array that specifies the x-values of a dataset.
y	An array that specifies the y-values of a dataset. The length of the y array must match the length of the x array.
valid	An array of boolean values, specifying which array elements of x and y are valid.
source	A text string containing CSV formatted data.

csv	An instance of a **CSV** object.

The number of data points is the value of x.length property. The x and y arrays must be the same length and every element must be initialized to a valid value. All values in the arrays are plotted. If the data is outside of the current chart scale the values will be clipped.

You can retrieve references to the internal arrays used to store the data using the **SimpleDataset** methods **getXData** and **getYData**. Change the values in the data using these references. You can also modify a point at a time using one of the **setDataPoint** methods. The **getDataPoint** method will get a xy data point at a specific index as a **ChartPoint2D** object. If you need to add new points to a dataset, increasing its size, use one of the **addDataPoint**, or **insertDataPoint** methods. Delete data points using the **deleteDataPoint** method. In order to see the modified dataset, force the graph to redraw using **ChartView.updateDraw** method.

Example of creating simple datasets from numeric arrays

[JavaScript]

```
import * as  QCChartTS from '../../QCChart2DTS/qcchart2DTS.js';
.
.
.
var x1 = [10, 20, 30, 40, 50];
var y1 = [9, -21, 20, 40, 30];
var Dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("First", x1, y1);
```

[TypeScript]

```
import * as  QCChartTS from '../../QCChart2DTS/qcchart2DTS.js';
.
.
.
let x1: number[] = [10, 20, 30, 40, 50];
let y1: number[] = [9, -21, 20, 40, 30];
let Dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("First", x1, y1);
```

You can also define an array and fill it.

[JavaScript]

```
import * as  QCChartTS from '../../QCChart2DTS/qcchart2DTS.js';
.
.
.
var x1 = new Array(5)
var y1 = new Array(5);
x1[0] = 10; y1[0] = 9;
```

```
x1[1] = 20; y1[1] = -21;
x1[2] = 30; y1[2] = 20;
x1[3] = 40; y1[3] = 40;
x1[4] = 50; y1[4] = 30;


var Dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("First", x1, y1);
```

[TypeScript]

```
import * as  QCChartTS from '../../QCChart2DTS/qcchart2DTS.js';
.
.
.
let x1: number[] = new Array(5)
let y1: number[]  = new Array(5);
x1[0] = 10; y1[0] = 9;
x1[1] = 20; y1[1] = -21;
x1[2] = 30; y1[2] = 20;
x1[3] = 40; y1[3] = 40;
x1[4] = 50; y1[4] = 30;


let Dataset1: QCChartTS.SimpleDataset.newSimpleDataset =
QCChartTS.SimpleDataset.newSimpleDataset("First", x1, y1);
```

Example of modifying simple dataset elements using the setDataPointPoint method.

[TypeScript]

```
import * as  QCChartTS from '../../QCChart2DTS/qcchart2DTS.js';
.
.
.
let x1: number[] = [10, 20, 30, 40, 50];
let y1: number[] = [9, -21, 20, 40, 30];
let Dataset1: QCChartTS.SimpleDataset = QCChartTS.SimpleDataset.newSimpleDataset("First", x1,
y1);

let datapoint: QCChartTS.ChartPoint2D  = Dataset1.getDataPoint(0);

if (datapoint.X < 0)
{  datapoint.X =  Math.abs(datapoint.X); // arbitrary
   Dataset1.setDataPointPoint(0,datapoint); // Change the datapoint
}
```

[JavaScript]

```
import * as  QCChartTS from '../../QCChart2DTS/qcchart2DTS.js';
.
.
.
var x1 = [10, 20, 30, 40, 50];
var y1 = [9, -21, 20, 40, 30];
var Dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("First", x1, y1);

var datapoint = Dataset1.getDataPoint(0);


if (datapoint.X < 0)
{  datapoint.X =  Math.abs(datapoint.X); // arbitrary
   Dataset1.setDataPointPoint(0,datapoint); // Change the datapoint
}
```

Simple Date/Time Dataset

Class TimeSimpleDataset

```
ChartObj
       |
       +--ChartDataset
              |
              +--SimpleDataset
                     |
                     +-TimeSimpleDataset
```

The **TimeSimpleDataset** uses **Date** dates as one set of the x- or y-values, and floating point values as the other. *Y*ou can have Date dates as either the x- or y-values in a **TimeSimpleDatset**. Date values are actually stored internally as their equivalent millisecond values. The **TimeSimpleDataset** class adds a large number of methods to the **SimpleDataset** class that make it easy to create and modify datasets that use Date values.

Note: Do **not** use the **TimeSimpleDataset** if you want to display data using the elapsed time. The **TimeSimpleDataset** uses a full Gregorian Calendar date/time and it is not suitable for the display of elapsed time, since time intervals do not have an explicit date, i.e. 10/11/2008. Use the **ElapsedTimeSimpleDataset** class, in combination with an **ElapsedTimeCoordinateSystem**, if you plan to create an elapsed time chart.

The following constructor creates a time dataset using the x- and y-values stored in arrays.

TimeSimpleDataset constructors

```
public static newTimeSimpleDataset(sname: string, x: Date[], y: number[]): TimeSimpleDataset

public static newTimeSimpleDatasetDateY(sname: string, x: number[], y: Date[]): TimeSimpleDataset

public static newTimeSimpleDatasetNumX(sname: string, x: number[], y: number[]):
TimeSimpleDataset

public static newTimeSimpleDatasetCSV(csv: CSV, source: string, rowskip: number,  columnskip:
number): TimeSimpleDataset
```

sname	Specifies the name of the dataset.
x	An array that specifies the x-values (either *numbers* or Date objects) of a dataset.
y	An array that specifies the y-values of a dataset. (either *numbers* or Date objects). The length of the y array must match the length of the x array.
valid	An array of boolean values, specifying which array elements of x and y are valid.
source	A text string containing CSV formatted data.
csv	An instance of a **CSV** object.

Either x- or y-values should be Date based. The number of data points is the value of x.length property. The x and y arrays must be the same length and every element must be initialized to a valid value. All values in the arrays are plotted. If the data is outside of the current chart scale the values will be clipped.

The next constructor creates a time dataset using the x- and y-values stored in a string that uses the CSV (Comma Separated Value) format. There are two ways to organize the data values within the source text string. If you use the COLUMN_MAJOR format, the first column represents the time values and the second column the y-values. If you use the ROW_MAJOR format, the first row represents the time values and the second row the y-values. Use the **CSV.setOrientation** method to initialize the csv argument for the proper data orientation.

```
[TypeScript]
public static newTimeSimpleDatasetCSV(csv: CSV, source: string, rowskip: number,  columnskip: num
ber): TimeSimpleDataset
```

csv	An instance of a **CSV** object.
source	The source text string.
rowskip	Skip this many rows before starting the read operation.
columnskip	For each row of data, skip this many columns before starting this read operation.

The format of date/time values in the string must be a format which is parsable by the JavaScript Date class, since that is what we use to convert strings to Date objects. So, use a format which successfully produces a valid Date object in the expression => var d = new Date("Your/Date/String Your:Time:String").

You can retrieve a *copy* of the date time data using the **TimeSimpleDataset.getTimeXData** (or **getTimeYData)** method. It returns an array of Date objects, and it is *not* a reference to the underlying data. The underlying data is stored as number values that represent the millisecond equivalent of the date time values. The **getDataPoint** method will get a xy data point at a specific index as a **ChartPoint2D** object. You can also modify a point at a time using one of the **setTimeDataPoint, setTimeXDataValue** (or **setTimeYDataValue)** and **setYDataValue** (or **setXDataValue**) methods. If you need to add new points to the dataset, increasing its size, use one of the **addTimeDataPoint**, or **insertTimeDataPoint** methods. Delete data points using the **deleteTimeDataPoint** method. In order to see the modified dataset, force the graph to redraw using **ChartView.updateDraw** method.

Note: When initializing **Date** objects, it is important that you instantiate a new Date object for each element of the array you fill. We do that with the line *x1[i] = new Date(currentdate)* in the code below. If you instantiate a single date object, and then use that same Date object over and over again, changing the date along the way, you will fill up the array with the same object. Its value for every element will end up the same, which will be the last value you assigned to the date object.

Example of creating a simple time datasets

[TypeScript]

```
import * as  QCChartTS from '../../QCChart2DTS/qcchart2DTS.js';

.
.
.
let nnumpnts: number = 32;
let x1: Date[] = new Array<Date>(nnumpnts);
let y1: number[] = new Array(nnumpnts);
let y2: number[] = new Array(nnumpnts);
let currentdate: Date = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);

let i: number;
y1[0] = 100;
y2[0] = 30;
x1[0] = new Date(currentdate);
QCChartTS.ChartCalendar.add(currentdate,QCChartTS.ChartConstants.MONTH, 3);
for (i = 1; i < nnumpnts; i++) {
    x1[i] = new Date(currentdate);
    y1[i] =  y1[i - 1] + (5 + i) * (0.75 - QCChartTS.ChartSupport.getRandomDouble());
    y2[i] =  y2[i - 1] + (15 + i) * (0.85 - QCChartTS.ChartSupport.getRandomDouble());
    QCChartTS.ChartCalendar.add(currentdate,QCChartTS.ChartConstants.MONTH, 3);
}

let Dataset1: QCChartTS.TimeSimpleDataset =  QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Ac
tual Sales", x1, y1);
let Dataset2: QCChartTS.TimeSimpleDataset = QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("For
ecast Sales", x1, y2);


[JavaScript]

import * as  QCChartTS from '../../QCChart2DTS/qcchart2DTS.js';
```

```
let nnumpnts = 32;
let x1 = new Array(nnumpnts);
let y1 = new Array(nnumpnts);
let y2 = new Array(nnumpnts);
let currentdate = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);
let i;
y1[0] = 100;
y2[0] = 30;
x1[0] = new Date(currentdate);
QCChartTS.ChartCalendar.add(currentdate, QCChartTS.ChartConstants.MONTH, 3);
for (i = 1; i < nnumpnts; i++) {
    x1[i] = new Date(currentdate);
    y1[i] = y1[i - 1] + (5 + i) * (0.75 - QCChartTS.ChartSupport.getRandomDouble());
    y2[i] = y2[i - 1] + (15 + i) * (0.85 - QCChartTS.ChartSupport.getRandomDouble());
    QCChartTS.ChartCalendar.add(currentdate, QCChartTS.ChartConstants.MONTH, 3);
}

let Dataset1 = QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Actual Sales", x1, y1);
let Dataset2 = QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Forecast Sales", x1, y2);
```

Simple ElapsedTime Dataset

Class ElapsedTimeSimpleDataset

```
ChartObj
   |
   +--ChartDataset
         |
         +--SimpleDataset
               |
               +-ElapsedTimeSimpleDataset
```

The **ElapsedTimeSimpleDataset** class uses **ChartTimeSpan** values as one set of the x- or y-values, and floating point values as the other. **ChartTimeSpan** values are actually stored internally as their equivalent millisecond values.

ElapsedTimeSimpleDataset constructors

```
public static newElapsedTimeSimpleDataset(sname: string, xt: ChartTimeSpan[], y: number[]):
ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetXNumber(sname: string, x: number[], y: number[]):
ElapsedTimeSimpleDataset
```

```
public static newElapsedTimeSimpleDatasetXYAxis(sname: string, x: number[], y: number[],
timeaxis: number): ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetXTimeSpanValid(sname: string, xt: ChartTimeSpan[], y:
number[], valid: boolean[]): ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetXTimeSpanValidN(sname: string, xt: ChartTimeSpan[], y:
number[], valid: boolean[], n: number): ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetYTimeSpan(sname: string, x: number[], yt:
ChartTimeSpan[]): ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetYTimeSpanValid(sname: string, x: number[], yt:
ChartTimeSpan[], valid: boolean[]): ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetYTimeSpanValidN(sname: string, x: number[], yt:
ChartTimeSpan[], valid: boolean[], n: number): ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetN(sname: string, n: number): ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetCSV(csv: CSV,  source: string, rowskip : number,
columnskip : number ): ElapsedTimeSimpleDataset
```

sname	Specifies the name of the dataset.
x	An array that specifies the x-values (either *numbers* or **ChartTimeSpan** objects) of a dataset.
y	An array that specifies the y-values of a dataset. (either *numbers* or **ChartTimeSpan** objects). The length of the y array must match the length of the x array.
valid	An array of boolean values, specifying which array elements of x and y are valid.
source	A text string containing CSV formatted data.
csv	An instance of a **CSV** object.

Either x- or y-values should be **ChartTimeSpan** based. The number of data points is the value of x.length property. The x and y arrays must be the same length and every element must be initialized to a valid value. All values in the arrays are plotted. If the data is outside of the current chart scale the values will be clipped.

The next constructor creates an elapsed time dataset using the x- and y-values stored in a file that uses the CSV (Comma Separated Value) format. There are two ways to organize the data values within the source text string. If you use the COLUMN_MAJOR format, the first column represents the time values and the second column the y-values. If you use the ROW_MAJOR format, the first row represents the time values and the second row the y-values. Use the **CSV.setOrientation** method to initialize the csv argument for the proper data orientation.

```
 public static newElapsedTimeSimpleDatasetCSV(csv: CSV,  source: string, rowskip : number, column
skip : number ): ElapsedTimeSimpleDataset
```

source	A text string containing CSV formatted data.

csv	An instance of a **CSV** object.
filename	The source text string.
rowskip	Skip this many rows before starting the read operation.
columnskip	For each row of data, skip this many columns before starting this read operation.

The only supported format for elapsed time values in a CSV file is d.hh:mm:ss.fff, (3.14:23:12.333 as an example of an elapsed time of three days, 14 hours, 23 minutes, 12 seconds and 333 milliseconds).

You can also modify a point at a time using **setElapsedTimeXDataValue** (or **setElapsedTimeYDataValue**) if you are using **ChartTimeSpan** objects**,** and **setYDataValue** or **setXDataValue**) if you use millisecond values. If you need to add new points to the dataset, increasing its size, use one of the **addDataPoint**, or **insertDataPoint** methods. Delete data points using the **deleteDataPoint** method. In order to see the modified dataset, force the graph to redraw using **ChartView.updateDraw** method.

Note: When initializing **ChartTimeSpan** objects, it is important that you instantiate a new ChartTimeSpan object for each element of the array you fill. We do that with the line *x1[i] = QCChartTS.ChartTimeSpan.fromMilliseconds* in the code below. If you instantiate a single ChartTimeSpan object, and then use that same ChartTimeSpan object over and over again, changing the ChartTimeSpan along the way, you will fill up the array with the same object. Its value for every element will end up the same, which will be the last value you assigned to the ChartTimeSpan object.

Example of creating a simple elapsed time datasets, extracted from the NewDemosRev2.ElapsedTimeChart example program.

[TypeScript]

```
let numPoints: number = 100;
let x1: QCChartTS.ChartTimeSpan[] = new Array<QCChartTS.ChartTimeSpan>(numPoints);
let y1: number[] = new Array(numPoints);
let y2: number[] = new Array(numPoints);
let i: number;
for (i = 0; (i < numPoints); i++) {
    x1[i] = QCChartTS.ChartTimeSpan.fromMilliseconds((i * (30 * 1000)));
    //  30000 milliseconds  increment
    //  Or you can use seconds, and the FromSeconds method
    // x1[i] = QCChartTS.ChartTimeSpan.fromSeconds(i * 30);  // 30 milliseconds  increment
    if ((Math.sin((x1[i].TotalSeconds / 20)) > 0)) {
y1[i] = (20 + (50 * ((0.5 - QCChartTS.ChartSupport.getRandomDouble())
    * Math.sin(((x1[i].TotalSeconds / 5)
* -1)))));
}
else {
y1[i] = (20 + (5
    * ((0.5 - QCChartTS.ChartSupport.getRandomDouble())
* Math.sin(((x1[i].TotalSeconds / 2)
    * -1)))));
    }

    y2[i] = (y1[i]
+ ((5 + (0.2 * x1[i].TotalSeconds)) * (0.5 - QCChartTS.ChartSupport.getRandomDouble())));
}


let Dataset1: QCChartTS.ElapsedTimeSimpleDataset =
QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDatasetXTimeSpan("First", x1, y1);
let Dataset2: QCChartTS.ElapsedTimeSimpleDataset =
QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDatasetXTimeSpan("Second", x1, y2);
```

[JavaScript]

```
let numPoints = 100;
let x1 = new Array(numPoints);
let y1 = new Array(numPoints);
let y2 = new Array(numPoints);
let i;
for (i = 0; (i < numPoints); i++) {
    x1[i] = QCChartTS.ChartTimeSpan.fromMilliseconds((i * (30 * 1000)));
    //  30000 milliseconds  increment
    //  Or you can use seconds, and the FromSeconds method
    // x1[i] = QCChartTS.ChartTimeSpan.fromSeconds(i * 30);
    if ((Math.sin((x1[i].TotalSeconds / 20)) > 0)) {
y1[i] = (20 + (50
    * ((0.5 - QCChartTS.ChartSupport.getRandomDouble())
* Math.sin(((x1[i].TotalSeconds / 5)
    * -1)))));
    }
    else {
y1[i] = (20 + (5
    * ((0.5 - QCChartTS.ChartSupport.getRandomDouble())
* Math.sin(((x1[i].TotalSeconds / 2)
    * -1)))));
    }
    y2[i] = (y1[i]
+ ((5 + (0.2 * x1[i].TotalSeconds)) * (0.5 - QCChartTS.ChartSupport.getRandomDouble())));
}

let Dataset1 = QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDatasetXTimeSpan("First", x
1, y1);
let Dataset2 = QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDatasetXTimeSpan("Second",
x1, y2);
```

Simple Event Dataset

Class

EventSimpleDataset

```
ChartObj
    |
    +--ChartDataset
        |
        +--SimpleDataset
            |
            +-EventSimpleDataset
```

Background for ChartEvent datasets

Most coordinate systems used in plotting represent a continuous domain. In the QCChart2D software, these includes linear, logarithmic, simple time/date, elapsed time, polar, and antenna coordinate systems. The one exception is a variant of the time/date scale which allows for periodic, yet discontinuous time. In the time/date scale case, it is possible to remove weekends from the time scale, and to define the hours of the day to be some subset of the standard 24-hour cycle. The most often used example is the stock trading day used in the US, which is from 9:30 to 16:00, and does not include weekends.

Unfortunately, the time coordinate system, even with discontinuous time, is still insufficient to plot the great variety of time plots needed in the financial services, and other, industries. Some of these special requirements are:

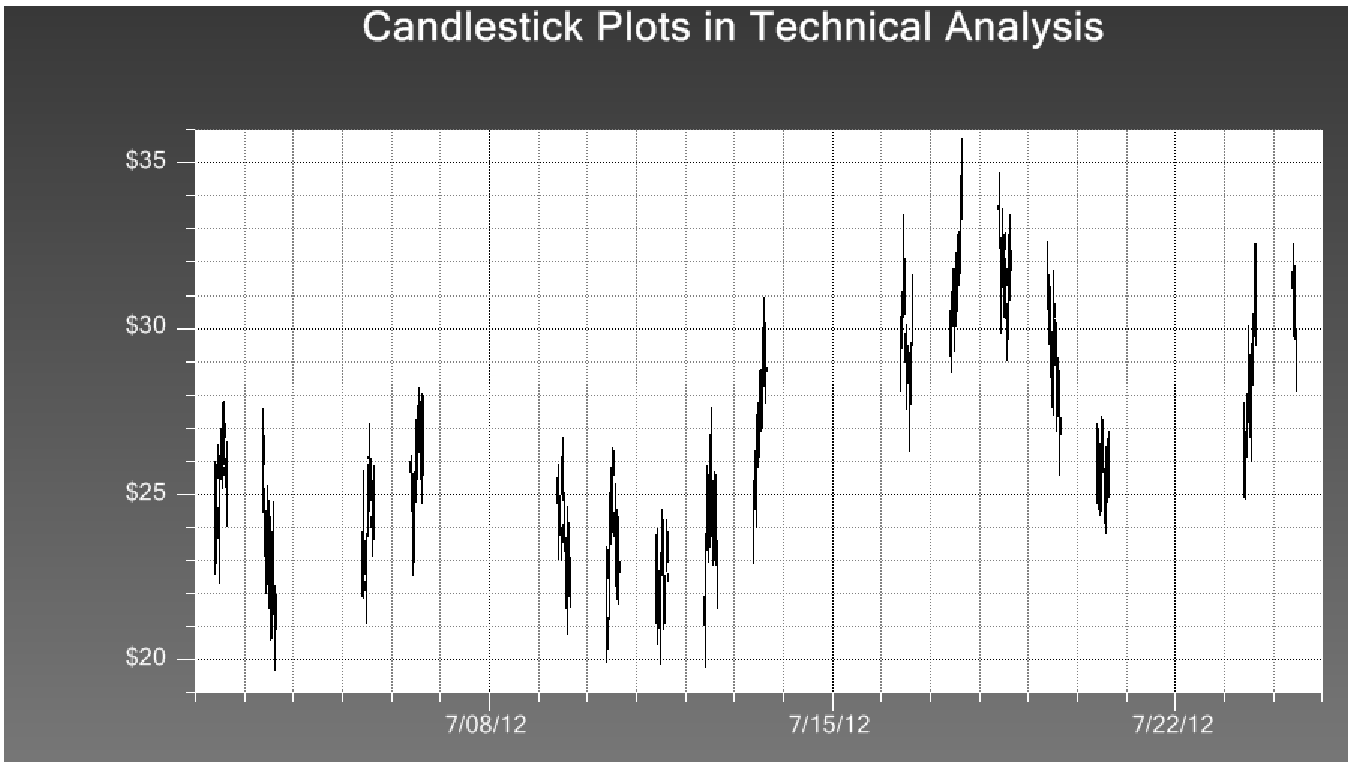
- Must be able to remove arbitrary (non-periodic) days, holidays for example, from the time/date scale.
- Allow for a sub range of a day which crosses 24:00, i.e. 18:00 to 3:00.
- Allow for multiple, active time ranges within the same 24-hour period, i.e. 9:00 AM to 12:00 and 14:00 to 18:00.
- Smooth panning and zooming of data across discontinuous time boundaries.
- Allow for exceptions to the predefined set of rules. For example, be able to include a weekend day, or a specific set of hours normally excluded from the scale.

These requirements are common enough that we wanted to address them with new coordinate system, dataset and axis classes, which can accommodate any set of continuous, or discontinuous time/date values, but not waste display space on gaps where no data exists. Rather than extend the existing time/date coordinate system (TimeCoordinates), we chose to create new coordinate system, EventCoordinates, which uses discrete events, rather than a continuous domain, as the basis for plotting data. The basis of the EventCoordinates system are the event dataset classes: EventSimpleDataset, and EventGroupDataset, and the underlying array of ChartEvent objects . Rather than define a plot using arrays of x- and y-values, a ChartEvent represents a specific point in time. The point in time has the following major properties as distinguishing elements:

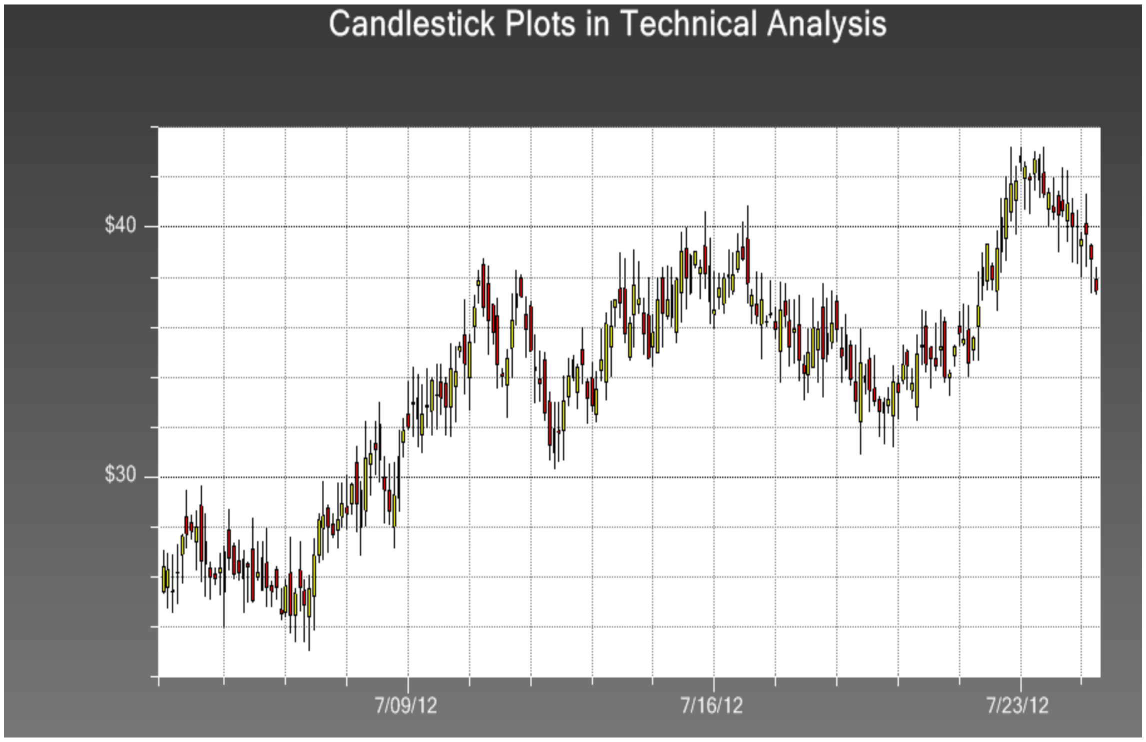
- **Description** – A description of the event.
- **ShortDescription** - An abbreviated description of the event.
- **AxisLabel** – A string which can be uses as an axis label for the event.
- **ToolTip** – A custom tool-tip which can be displayed if the event is clicked on.
- **Position** – The position of the event with respect to the underlying linear coordinate system.

- **TimeStamp** – The time stamp of the event. Indirectly related to the Position of the event in the coordinate system
- **NumericTimeStamp** – The numeric value of the time stamp in milliseconds + an offset (numericTimeStampOffset).
- **NumericValues** - One or more numeric y-values (a simple plot uses a single y-value, while a group plot uses an array of y-values.

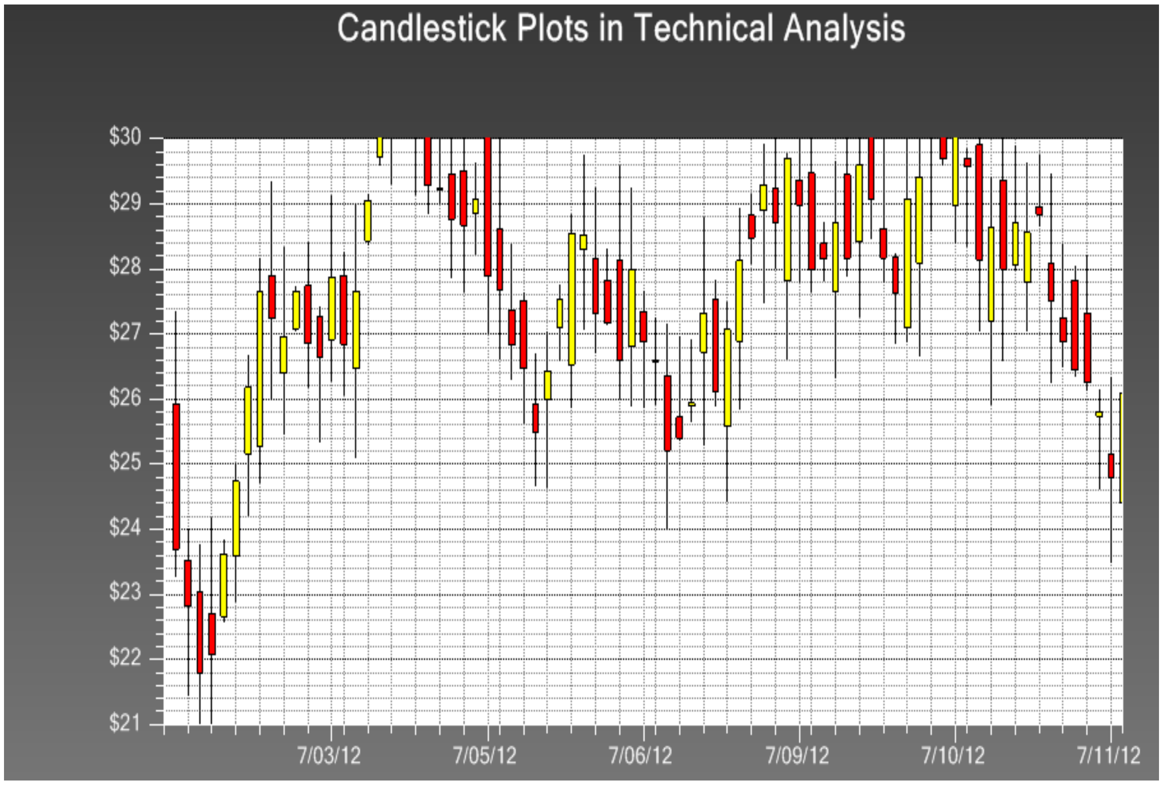
The ChartEvent class incorporates two x-value positioning properties, the Position and the TimeStamp, and one or more numeric y-values for each event. A single event therefore defines both the x-and y-values of the event in the underlying coordinate system. A collection, or array, of ChartEvent objects define the data for a plot, the same way as arrays of x- and y-values define a plot when using a simple dataset class with a Cartesian coordinate system. The critical element of the ChartEvent which permit it to be used for the plotting of discontinuous data is that the Position of the event in a chart is related, but, independent of the TimeStamp of the event. Event data can be positioned contiguously, and evenly spaced, in a chart, even if the time stamps of the events are not contiguous, or evenly spaced. Here is a simple example of a standard financial candlestick plot chart, using our TimeCoordinates class as the coordinate system, where the time/date data is not evenly spaced, and contains large gaps corresponding to weekends, and inactive hours of the day. The July 4th holiday is included in the range, and there is no data for that time interval either.



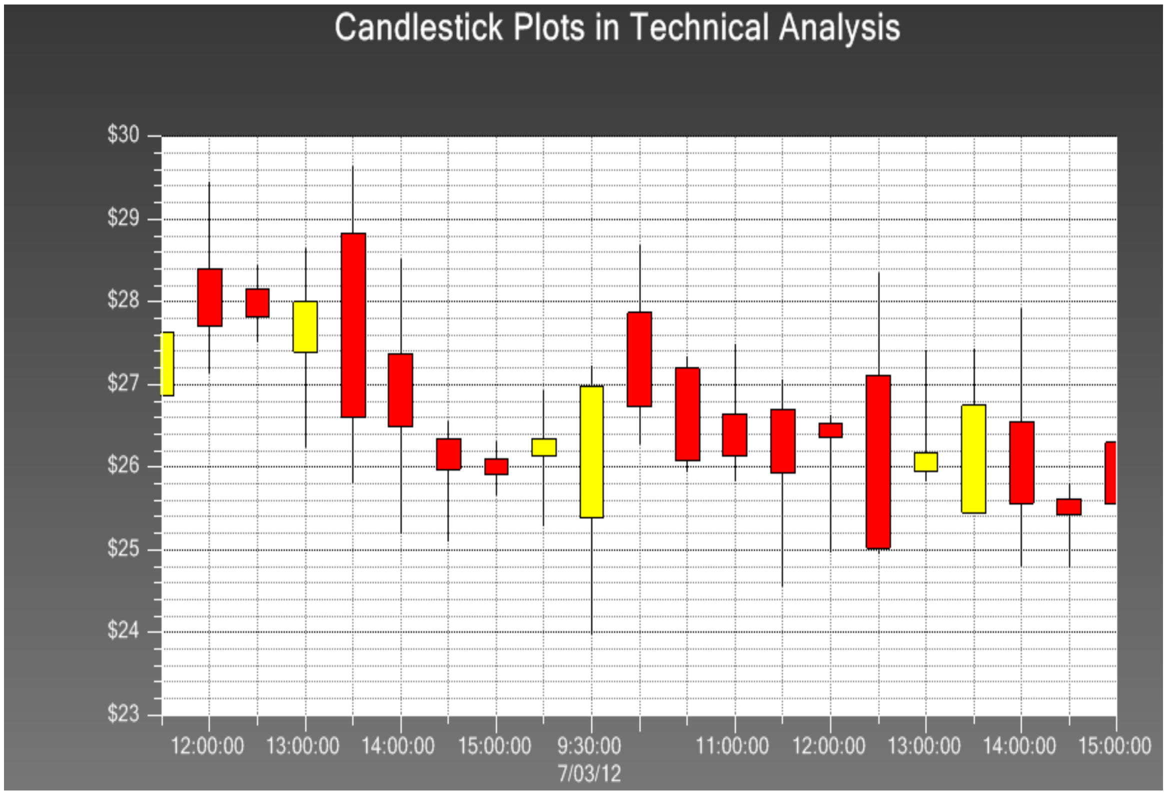
Contrast this to the similar data, using the same time range, plotted using the EventCoordinates class. Note how every event is evenly spaced with its neighbor. Gaps do not exist, since weekends, holidays, and unused hours are bridged over as if they do not exist.he same would be true for gaps due to holidays, and a varying number of work hours in a day.



Zooming in further, you can see the smooth transition across the July 4th holiday, and the following weekend.



Zoom in again, and you can see the smooth transition from one day to the next, even though the working hours are only a 9:30 to 16:00 subset of the 24 hours of a day.



This is accomplished because of the dual positioning values, Position and TimeStamp, of the ChartEvent class. Each element of a plot object (one of the candlestick objects in the plot above) is positioned in a simple linear coordinate system, starting at 0 and incrementing by 1 for each ChartEvent object. In the previous example, the first ChartEvent object is has a Position value of 0.0, and the last ChartEvent object has a position of 199, because there are 200 data points in the chart. This is what keeps the individual elements of a plot o bject evenly spaced, because the plot elements are positioned in the chart using the Position value, not the TimeStamp value. But, the associated x-axis (EventAxis) and x-axis labels objects (EventAxisLabels) look to the TimeStamp property for their values, not the Position property. What you end up with is the clean, evenly spaced look of a simple linear chart, with the axis tick marks and axis labeling of a dedicated time/date axis. The graph can be made to scroll (or pan) left to right, or re-scale along the y-axis, smoothly.

The EventSimpleDataset class is used to supply the simple plot classes: SimpleLinePlot, SimpleBarPlot, SimpleScatterPlot, and SimpleLineMarkerPlot, with data.

A ChartEvent can have one ore more y-values. In the case of an EventSimpleDataset, usually it will have a single value, as seen in the programming example below, one y-value for each x-value. In the EventGroupDataset, each ChartEvent object can have multiple y-values for each x-value.

EventSimpleDataset constructors

```
public static newEventSimpleDatasetNameArrayDates(sname: string, x: Date[], y: number[]):
EventSimpleDataset
```

```
public static newEventSimpleDatasetNameArrayEvents(sname: string, evs: ChartEvent[]):
EventSimpleDataset

public static newEventSimpleDataset(sname: string, evs: ChartEvent[]): EventSimpleDataset

public static newEventSimpleDatasetNameArrayEventsValid(sname: string, evs: ChartEvent[], valid:
boolean[]): EventSimpleDataset

public static newEventSimpleDatasetNameArrayEventsValidN(sname: string, evs: ChartEvent[], valid:
boolean[], n: number): EventSimpleDataset

public static newEventSimpleDatasetNameArrayDatesArrayPos(sname: string, timestamps: Date[], pos:
number[], y: number[]): EventSimpleDataset
```

sname	Specifies the name of the dataset.
evs	An array of ChartEvent objects.
x	An array of Date x-values
y	An array of numeric y-values
source	A text string containing CSV formatted data.
csv	An instance of a **CSV** object.
valid	An array of boolean values, specifying which array elements of x and y are valid.

Create an array of ChartEvent objects, specifying the time-stamp and y-value for each ChartEvent object and use that to initialize a EventSimpleDataset.

The next constructor creates an ChartEvent dataset using the x- and y-values stored in a text string that uses the CSV (Comma Separated Value) format. Only the COLUMN_MAJOR format is supported, where each row presents a ChartEvent object, and the columns are organized as: description, short description, x-axis string label, tool tip string, position, time stamp, numeric time stamp, y-value index (index for the y-value to use when the ChartEvent contains multiple y-values), and the y-values.

```
  public static newEventSimpleDatasetCSV(csv: CSV,  source: string, rowskip : number, columnskip
: number ): EventSimpleDataset
```

csv	An instance of a **CSV** object.
source	The source text string of the file.
rowskip	Skip this many rows before starting the read operation.
columnskip	For each row of data, skip this many columns before starting this read operation.

You can also modify a point at a time using **setEvent.** If you need to add new points to the dataset, increasing its size, use one of the **addEvent**, or **insertEvent** methods. Delete data points using the **deleteEvent** method. In order to see the modified dataset, force the graph to redraw using **ChartView.updateDraw** method.

.

Example of creating a simple event datasets, extracted from the ChartEventExamples.SimpleEventChart example program.

[TypeScript]

```
let nnumpnts: number = 20;
let x1: Date[] = new Array<Date>(nnumpnts);
let y1: number[] = new Array(nnumpnts);
let currentdate: Date = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);
let chartevents: QCChartTS.ChartEvent[] = new Array<QCChartTS.ChartEvent>();
let i: number;
let startx: number = 1;
for (i = 0; (i < nnumpnts); i++) {
    if ((i == 0)) {
      y1[0] = 100;
      x1[0] = new Date(currentdate);

chartevents[0] = QCChartTS.ChartEvent.newChartEventDateTimeStampPosValue(x1[0], startx, y1[0]);
      chartevents[0].AxisLabel = ("XY" + "0");
      QCChartTS.ChartCalendar.add(currentdate, QCChartTS.ChartConstants.MONTH, 12);
    }
    else {
      x1[i] = new Date(currentdate);
      y1[i] = ((y1[(i - 1)] + (25 * (0.55 - QCChartTS.ChartSupport.getRandomDouble()))));

chartevents[i] = QCChartTS.ChartEvent.newChartEventDateTimeStampPosValue(x1[i], (i + startx), y1[
i]);
      chartevents[i].AxisLabel = ("XY" + i.toString());
      QCChartTS.ChartCalendar.add(currentdate, QCChartTS.ChartConstants.MONTH, 12);
    }
}

let Dataset1: QCChartTS.EventSimpleDataset = QCChartTS.EventSimpleDataset.newEventSimpleDatasetNa
meArrayEvents("Actual Sales", chartevents);m1 = QCChartTS.EventCoordinates.newEventCoordinatesSim
pleDataset(Dataset1);
```

[JavaScript]

```
let nnumpnts = 20;
let x1 = new Array(nnumpnts);
let y1 = new Array(nnumpnts);
let currentdate = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);
let chartevents = new Array();
let i;
let startx = 1;
for (i = 0; (i < nnumpnts); i++) {
    if ((i == 0)) {
      y1[0] = 100;
      x1[0] = new Date(currentdate);

chartevents[0] = QCChartTS.ChartEvent.newChartEventDateTimeStampPosValue(x1[0], startx, y1[0]);
      chartevents[0].AxisLabel = ("XY" + "0");
      QCChartTS.ChartCalendar.add(currentdate, QCChartTS.ChartConstants.MONTH, 12);
    }
    else {
      x1[i] = new Date(currentdate);
      y1[i] = ((y1[(i - 1)] + (25 * (0.55 - QCChartTS.ChartSupport.getRandomDouble()))));

chartevents[i] = QCChartTS.ChartEvent.newChartEventDateTimeStampPosValue(x1[i], (i + startx), y1[
i]);
      chartevents[i].AxisLabel = ("XY" + i.toString());
      QCChartTS.ChartCalendar.add(currentdate, QCChartTS.ChartConstants.MONTH, 12);
    }
}

let Dataset1 = QCChartTS.EventSimpleDataset.newEventSimpleDatasetNameArrayEvents("Actual Sales",
chartevents);
let pTransform1 = QCChartTS.EventCoordinates.newEventCoordinatesSimpleDataset(Dataset1);
```

Numeric Group Dataset

Class GroupDataset

ChartObj
```
   |
   +--ChartDataset
          |
          +--GroupDataset
```

The **GroupDataset** class represents group data, where every x-value can have one or more y-values. The number of x-values in a group plot is referred to as the number of columns or as **numberDatapoints** and the number of y-values *for each x-value* is referred to as the number of rows, or **numberGroups**. Think of spreadsheet file that looks like

x-values	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]
y-values group #0	y[0,0]	y[0,1]	y[0,2]	y[0,3]	y[0,4]	y[0,5]
y-values group #1	y[1,0]	y[1,1]	y[1,2]	y[1,3]	y[1,4]	y[1,5]
y-values group #2	y[2,0]	y[2,1]	y[2,2]	y[2,3]	y[2,4]	y[2,5]

number of x-values = **numberDatapoints** = numberColumns = 6

number of y-values for each x-value = **numberGroups** = numberRows = 3

In terms of array access, the indices of the array are [group or row #][column #].

This would be the ROW_MAJOR format if the data were stored in a CSV file.

GroupDataset constructors

```
 public static newGroupDataset(sname: string, x: number[], y: number[][]): GroupDataset

 public static newGroupDatasetX2DY(sname: string, x: number[], y: number[][]): GroupDataset

 public static newGroupDatasetX1DY(sname: string, x: number[], y: number[]): GroupDataset

 public static newGroupDatasetX2DYValid(sname: string, x: number[], y: number[][], valid:
boolean[]): GroupDataset

 public static newGroupDatasetCSV(csv: CSV,  source: string, rowskip : number, columnskip :
number ): GroupDataset
```

sname	Specifies the name of the dataset.
x	An array that specifies the x-values of a group dataset. The length of the x array sets the number of columns for the group dataset.
y	An array that specifies the y-values of a group dataset where y has the dimensions [number of rows, number of columns]. The number of rows in the y-array sets the number of groups in the group dataset. The number of columns in the y-array must match the length of the x-array.
source	A text string containing CSV formatted data.
csv	An instance of a **CSV** object.
valid	An array of boolean values, specifying which array elements of x and y are valid.

The number of columns in the group dataset is the value of x.length property. The number of columns in the y array must match the length of the x array and every element must be initialized to a valid value. All values in the arrays are plotted. If the data is outside of the current chart scale the values will be clipped.

The next constructor creates a dataset using the x- and y-values stored in a file that uses the CSV (Comma Separated Value) format. There are two ways to organize the data values within the source text string. If you use the COLUMN_MAJOR format, the first column represents the x-values and subsequent columns represent the y-values, where each column is a group. If you use the

ROW_MAJOR format, the first row represents the x-values and subsequent rows represent the y-values, where each row is a group. Use the **CSV.setOrientation** method to initialize the csv argument for the proper data orientation.

```
public static newGroupDatasetCSV(csv: CSV,  source: string, rowskip : number, columnskip : number
): GroupDataset
```

csv	An instance of a **CSV** object.
source	The source text string.
rowskip	Skip this many rows before starting the read operation.
columnskip	For each row of data, skip this many columns before reading the first value from the row.

You can retrieve references to the internal arrays used to store the data using the **GroupDataset** methods **getXData** and **getGroupData**. Change the values in the data arrays using these references. You can also modify a point at a time using one of the **setYDataValue** and **setXDataValue** methods. If you need to add new points to dataset, increasing its size, use one of the **addGroupDataPoints**, or **insertGroupDataPoints** methods. Delete data points using the **deleteGroupDataPoints** method. In order to see the modified dataset, force the graph to redraw using **ChartView.updateDraw** method.

Example of creating a group datasets from numeric arrays

[TypeScript]

```
let nNumPnts: number = 5, nNumGroups = 4;
let xValues: number[] = [0,1,2,3,4];
let groupBarData: number[] = [[6, 3, 2, 1, 1], [6, 4, 3, 1, 2], [7, 4, 5, 6, 3], [2, 3, 4, 5,
6]];

let Dataset1: QCChartTS.GroupDataset = QCChartTS.TimeGroupDataset.newGroupDataset("GroupData",
xValues, groupBarData);
```

[JavaScript]

```
let nNumPnts = 5, nNumGroups = 4;
let xValues = [0,1,2,3,4];
let groupBarData = [[6, 3, 2, 1, 1], [6, 4, 3, 1, 2], [7, 4, 5, 6, 3], [2, 3, 4, 5, 6]];

let Dataset1 = QCChartTS.GroupDataset.newGroupDatasetDateX2DY("GroupData", xValues,
groupBarData);
```

While creating 2-dimensional arrays such as groupBarData is pretty easy using the bracked embedded initialization as seen above, it is a little more difficult to create an empty 2D array. The utility routine newArray2D in ChartSupport will do that for you.

[TypeScript]

```
let nNumPnts: number = 5, nNumGroups = 4;
let xValues: number = new Array(nNumPnts);
let groupBarData: number[][] = QCChartTS.ChartSupport.newArray2D(nNumGroups, nNumPnts);

xValues[0] = 0;
groupBarData[0][0] = 6.3;
groupBarData[1][0] = 3.1;
groupBarData[2][0] = 2.2;
groupBarData[3][0] = 1.8;
xValues[1] = 1;
groupBarData[0][1] = 5.8;
groupBarData[1][1] = 4.3;
groupBarData[2][1] = 2.8;
groupBarData[3][1] = 1.5;
xValues[2] = ne2;
groupBarData[0][2] = 5.5;
groupBarData[1][2] = 4.5;
groupBarData[2][2] = 2.5;
groupBarData[3][2] = 2.1;
xValues[3] = 3;
groupBarData[0][3] = 4.1;
groupBarData[1][3] = 5.4;
groupBarData[2][3] = 4.1;
groupBarData[3][3] = 3.2;
xValues[4] = 4;
groupBarData[0][4] = 3.8;
groupBarData[1][4] = 5.6;
groupBarData[2][4] = 4.3;
groupBarData[3][4] = 3.3;
let Dataset1: QCChartTS.GroupDataset = QCChartTS.GroupDataset.newGroupDataset("GroupData", xValue
s, groupBarData);
```

[JavaScript]

```
let nNumPnts = 5, nNumGroups = 4;
let xValues = new Array(nNumPnts);
let groupBarData = QCChartTS.ChartSupport.newArray2D(nNumGroups, nNumPnts);

xValues[0] = 0;
groupBarData[0][0] = 6.3;
groupBarData[1][0] = 3.1;
groupBarData[2][0] = 2.2;
groupBarData[3][0] = 1.8;
xValues[1] = 1;
groupBarData[0][1] = 5.8;
groupBarData[1][1] = 4.3;
groupBarData[2][1] = 2.8;
groupBarData[3][1] = 1.5;
xValues[2] = ne2;
groupBarData[0][2] = 5.5;
groupBarData[1][2] = 4.5;
groupBarData[2][2] = 2.5;
groupBarData[3][2] = 2.1;
xValues[3] = 3;
groupBarData[0][3] = 4.1;
groupBarData[1][3] = 5.4;
groupBarData[2][3] = 4.1;
groupBarData[3][3] = 3.2;
```

```
xValues[4] = 4;
groupBarData[0][4] = 3.8;
groupBarData[1][4] = 5.6;
groupBarData[2][4] = 4.3;
groupBarData[3][4] = 3.3;
let Dataset1: QCChartTS.GroupDataset = QCChartTS.GroupDataset.newGroupDataset("GroupData", xValue
s, groupBarData);
```

Example of creating a group datasets from a CSV file

[TypeScript]

```
let csvDataFile: QCChartTS.CSV  = new QCChartTS.CSV();
// read the dataset from a CSV string (source)
let Dataset1: QCChartTS.GroupDataset  =
    QCChartTS.GroupDataset.newGroupDatasetCSV(csvDataFile,source,0,0);
.
.


// convert the dataset to a CSV string
let csvstring: string =  Dataset1.writeGroupDatasetCSV (csvDataFile);
```

[JavaScript]

```
let csvDataFile = new QCChartTS.CSV();
// read the dataset from a CSV string (source)
let Dataset1 = QCChartTS.GroupDataset.newGroupDatasetCSV(csvDataFile, source, 0, 0);
.
.
.
// convert the dataset to a CSV string
let csvstring = Dataset1.writeGroupDatasetCSV(csvDataFile);
```

Date/Time Group Dataset

Class TimeGroupDataset

```
ChartObj
        |
        +--ChartDataset
                |
                +--GroupDataset
                        |
                        +-TimeGroupDataset
```

The **TimeGroupDataset** uses **Date** dates as the x-values, and floating point numbers as the y-values. Date values are actually stored internally as their equivalent millisecond values. The **TimeGroupDataset** class adds a large number of methods to the **GroupDataset** class that make it easy to create and modify datasets that use Date values.

Note - Do **not** use the **TimeGroupDataset** if you want to display data using the elapsed time. The **TimeGroupDataset** uses a full Gregorian Calendar date/time and it is not suitable for the display of elapsed time, since time intervals do not have an explicit date, i.e. 10/11/2008. Use the **ElapsedTimeGroupDataset** class, in combination with an **ElapsedTimeCoordinateSystem**, if you plan to create an elapsed time chart.

This constructor creates a new group **TimeGroupDataset** object where the x-values are **Date** values and the y-values are floating point numbers.

TimeGroupDataset constructors

```
public static newTimeGroupDataset(sname: string, x: Date[], y: number[][]): TimeGroupDataset

public static newTimeGroupDatasetDateX2DY(sname: string, x: Date[], y: number[][]):
TimeGroupDataset

public static newTimeGroupDatasetDateXY(sname: string, x: Date[], y: number[]): TimeGroupDataset

public static newTimeGroupDatasetXDate2DY(sname: string, x: number[], y: Date[][]):
TimeGroupDataset

public static newTimeGroupDatasetDateXDate2DY(sname: string, x: Date[], y: Date[][]):
TimeGroupDataset

public static newTimeGroupDatasetRC(sname: string, nrows: number, ncols: number):
TimeGroupDataset

public static newTimeGroupDatasetCSV(csv: CSV, source: string, rowskip: number,  columnskip:
number): TimeGroupDataset
```

sname	Specifies the name of the dataset.
x	An array of **Date** dates, that specifies the x-values of a dataset. The length of the x array sets the number of columns for the group dataset.
y	An array that specifies the y-values of a group dataset where y has the dimensions [number of rows, number of columns]. The number of rows in the y-array sets the number of groups in the group dataset. The number of columns in the y-array must match the length of the x-array.
source	A text string containing CSV formatted data.
csv	An instance of a **CSV** object.
valid	An array of boolean values, specifying which array elements of x and y are valid.

The number of columns in the group dataset is the value of x.length property. The number of columns in the y array must match the length of the x array and every element must be initialized to a valid value. All values in the arrays are plotted. If the data is outside of the current chart scale the values will be clipped.

In terms of array access, the indices of the array are [group or row #][column #].

```
public static newTimeGroupDatasetCSV(csv: CSV, source: string, rowskip: number,  columnskip: numb
er): TimeGroupDataset
```

csv	An instance of a **CSV** object.
source	A text string containing CSV formatted data
rowskip	Skip this many rows before starting the read operation.
columnskip	For each row of data, skip this many columns before reading the

There are two ways to organize the data values within the source text string. If you use the COLUMN_MAJOR format, the first column represents the time values and subsequent columns represent the y-values, where each column is a group. If you use the ROW_MAJOR format, the first row represents the time values and subsequent rows represent the y-values, where each row is a group. Use the **CSV.setOrientation** method to initialize the csv argument for the proper data orientation.

You can retrieve a *copy* of the date time data using the **TimeGroupDataset.getTimeXData** method. It returns an array of **Date** objects, and it is *not* a reference to the underlying data. The underlying data is stored as number values that represent the millisecond equivalent of the date time values. The **TimeGroupDataset getXData** and **getYData** methods return references to the underlying data**.** You can also modify a point at a time using one of the **TimeGroupDataset, setTimeXDataValue** and **setYDataValue** methods. If you need to add new points to dataset, increasing its size, use one of the **AddTimeGroupDataPoints**, or **InsertTimeGroupDataPoints** methods. Delete data points using the **deleteDataPoint** method. In order to see the modified dataset, force the graph to redraw using **ChartView.updateDraw** method.

Example of creating a group time datasets

[TypeScript]

```
let nNumPnts: number = 5, nNumGroups = 4;
let xValues: Date[] = new Array<Date>(nNumPnts);
let groupBarData: number[] = [[6, 3, 2, 1, 1], [6, 4, 3, 1, 2], [7, 4, 5, 6, 3], [2, 3, 4, 5,
6]];

xValues[0] = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);
xValues[1] = new Date(1999, QCChartTS.ChartConstants.JANUARY, 1);
xValues[2] = new Date(2000, QCChartTS.ChartConstants.JANUARY, 1);
xValues[3] = new Date(2001, QCChartTS.ChartConstants.JANUARY, 1);
```

```
xValues[4] = new Date(2002, QCChartTS.ChartConstants.JANUARY, 1);

let Dataset1: QCChartTS.TimeGroupDataset =
QCChartTS.TimeGroupDataset.newTimeGroupDatasetDateX2DY("GroupTimeData", xValues, groupBarData);
```

[JavaScript]

```
let nNumPnts = 5, nNumGroups = 4;
let xValues = new Array(nNumPnts);
let groupBarData = [[6, 3, 2, 1, 1], [6, 4, 3, 1, 2], [7, 4, 5, 6, 3], [2, 3, 4, 5, 6]];

xValues[0] = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);
xValues[1] = new Date(1999, QCChartTS.ChartConstants.JANUARY, 1);
xValues[2] = new Date(2000, QCChartTS.ChartConstants.JANUARY, 1);
xValues[3] = new Date(2001, QCChartTS.ChartConstants.JANUARY, 1);
xValues[4] = new Date(2002, QCChartTS.ChartConstants.JANUARY, 1);

let Dataset1 = QCChartTS.TimeGroupDataset.newTimeGroupDatasetDateX2DY("GroupTimeData", xValues,
groupBarData);
```

While creating 2-dimensional arrays such as groupBarData is pretty easy using the bracked embedded initialization as seen above, it is a little more difficult to create an empty 2D array. The utility routine newArray2D in ChartSupport will do that for you.

[TypeScript]

```
let nNumPnts: number = 5, nNumGroups = 4;
let xValues: Date[] = new Array<Date>(nNumPnts);
let groupBarData: number[][] = QCChartTS.ChartSupport.newArray2D(nNumGroups, nNumPnts);


xValues[0] = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][0] = 6.3;
groupBarData[1][0] = 3.1;
groupBarData[2][0] = 2.2;
groupBarData[3][0] = 1.8;
xValues[1] = new Date(1999, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][1] = 5.8;
groupBarData[1][1] = 4.3;
groupBarData[2][1] = 2.8;
groupBarData[3][1] = 1.5;
xValues[2] = new Date(2000, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][2] = 5.5;
groupBarData[1][2] = 4.5;
groupBarData[2][2] = 2.5;
groupBarData[3][2] = 2.1;
xValues[3] = new Date(2001, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][3] = 4.1;
groupBarData[1][3] = 5.4;
groupBarData[2][3] = 4.1;
groupBarData[3][3] = 3.2;
xValues[4] = new Date(2002, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][4] = 3.8;
groupBarData[1][4] = 5.6;
groupBarData[2][4] = 4.3;
groupBarData[3][4] = 3.3;
let Dataset1: QCChartTS.TimeGroupDataset = QCChartTS.TimeGroupDataset.newTimeGroupDatasetDateX2DY
("GroupTimeData", xValues, groupBarData);
```

[JavaScript]

```
let nNumPnts = 5, nNumGroups = 4;
let xValues = new Array(nNumPnts);
let groupBarData = QCChartTS.ChartSupport.newArray2D(nNumGroups, nNumPnts);

xValues[0] = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][0] = 6.3;
groupBarData[1][0] = 3.1;
groupBarData[2][0] = 2.2;
groupBarData[3][0] = 1.8;
xValues[1] = new Date(1999, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][1] = 5.8;
groupBarData[1][1] = 4.3;
groupBarData[2][1] = 2.8;
groupBarData[3][1] = 1.5;
xValues[2] = new Date(2000, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][2] = 5.5;
groupBarData[1][2] = 4.5;
groupBarData[2][2] = 2.5;
groupBarData[3][2] = 2.1;
xValues[3] = new Date(2001, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][3] = 4.1;
groupBarData[1][3] = 5.4;
groupBarData[2][3] = 4.1;
groupBarData[3][3] = 3.2;
xValues[4] = new Date(2002, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][4] = 3.8;
groupBarData[1][4] = 5.6;
groupBarData[2][4] = 4.3;
groupBarData[3][4] = 3.3;
let Dataset1 = QCChartTS.TimeGroupDataset.newTimeGroupDatasetDateX2DY("GroupTimeData", xValues,
groupBarData);
```

Example of creating a time datasets from a CSV text string

[TypeScript]

```
let csvDataFile: QCChartTS.CSV  = new QCChartTS.CSV();
// read the dataset from a CSV string (source)
let Dataset1: QCChartTS.TimeGroupDataset  =
    QCChartTS.TimeGroupDataset.newTimeGroupDatasetCSV(csvDataFile,source,0,0);
.
.


// convert the dataset to a CSV string
let csvstring: string =  Dataset1.writeTimeGroupDatasetCSV (csvDataFile);
```

[JavaScript]

```
let csvDataFile = new QCChartTS.CSV();
// read the dataset from a CSV string (source)
let Dataset1 = QCChartTS.TimeGroupDataset.newTimeGroupDatasetCSV(csvDataFile, source, 0, 0);
.
.
.
// convert the dataset to a CSV string
let csvstring = Dataset1.writeTimeGroupDatasetCSV(csvDataFile);
```

Elapsed Time Dataset

Class ElapsedTimeGroupDataset

```
ChartObj
   |
   +--ChartDataset
         |
         +--GroupDataset
               |
               +--ElapsedTimeGroupDataset
```

The **ElapsedTimeGroupDataset** class represents group data, where every x-value can have one or more y-values. The number of x-values in a group plot is referred to as the number of columns or as **numberDatapoints** and the number of y-values *for each x-value* is referred to as the number of rows, or **numberGroups**.

ElapsedTimeGroupDataset constructors

```
public static newElapsedTimeSimpleDataset(sname: string, xt: ChartTimeSpan[], y: number[]):
ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetXNumber(sname: string, x: number[], y: number[]):
ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetXYAxis(sname: string, x: number[], y: number[],
timeaxis: number): ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetXTimeSpanValid(sname: string, xt: ChartTimeSpan[], y:
number[], valid: boolean[]): ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetXTimeSpanValidN(sname: string, xt: ChartTimeSpan[], y:
number[], valid: boolean[], n: number): ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetYTimeSpan(sname: string, x: number[], yt:
ChartTimeSpan[]): ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetYTimeSpanValid(sname: string, x: number[], yt:
ChartTimeSpan[], valid: boolean[]): ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetYTimeSpanValidN(sname: string, x: number[], yt:
ChartTimeSpan[], valid: boolean[], n: number): ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetN(sname: string, n: number): ElapsedTimeSimpleDataset

public static newElapsedTimeSimpleDatasetCSV(csv: CSV,  source: string, rowskip : number,
columnskip : number ): ElapsedTimeSimpleDataset
```

sname	Specifies the name of the dataset.
x	An array that specifies the x-values of a group dataset. The length of the x array sets the number of columns for the group dataset.

y	An array that specifies the y-values of a group dataset where y has the dimensions [number of rows, number of columns]. The number of rows in the y-array sets the number of groups in the group dataset. The number of columns in the y-array must match the length of the x-array.
source	A text string containing CSV formatted data.
csv	An instance of a **CSV** object.
valid	An array of boolean values, specifying which array elements of x and y are valid.

The number of columns in the group dataset is the value of x.length property. The number of columns in the y array must match the length of the x array and every element must be initialized to a valid value. All values in the arrays are plotted. If the data is outside of the current chart scale the values will be clipped.

The next constructor creates a dataset using the x- and y-values stored in a file that uses the CSV (Comma Separated Value) format. There are two ways to organize the data values within the source text string. If you use the COLUMN_MAJOR format, the first column represents the x-values and subsequent columns represent the y-values, where each column is a group. If you use the ROW_MAJOR format, the first row represents the x-values and subsequent rows represent the y-values, where each row is a group. Use the **CSV.setOrientation** method to initialize the csv argument for the proper data orientation.

```
 public static newElapsedTimeGroupDatasetCSV(csv: CSV,  source: string, rowskip : number,
columnskip : number ): ElapsedTimeGroupDataset
```

csv	An instance of a **CSV** object.
source	A text string containing CSV formatted data.
rowskip	Skip this many rows before starting the read operation.
columnskip	For each row of data, skip this many columns before reading the first value from the row.

The only supported format for elapsed time values in a CSV file is d.hh.mm.ss.fff, (3.14:23:12.333 as an example of an elapsed time of three days, 14 hours, 23 minutes, 12 seconds and 333 milliseconds).

You can also modify a point at a time using **setElapsedTimeXDataValue** (or **setElapsedTimeYDataValue**) if you are using **ChartTimeSpan** objects, and **setYDataValue** or **setXDataValue**) if you use millisecond values. If you need to add new points to the dataset, increasing its size, use one of the **addDataPoint**, or **insertDataPoint** methods. Delete data points using the

deleteDataPoint method. In order to see the modified dataset, force the graph to redraw using **ChartView.updateDraw** method.

Example of creating a group datasets from numeric arrays

[TypeScript]

```
let nNumGroups: number = 4;
let nNumPnts: number = 5;
let xValues: QCChartTS.ChartTimeSpan[] = new Array<QCChartTS.ChartTimeSpan>(nNumPnts);
let groupBarData: number[][] = QCChartTS.ChartSupport.newArray2D(nNumGroups, nNumPnts);

xValues[0] = QCChartTS.ChartTimeSpan.fromMinutes(1);
groupBarData[0][0] = 6.3;
groupBarData[1][0] = 3.1;
groupBarData[2][0] = 2.2;
groupBarData[3][0] = 1.8;
xValues[1] = QCChartTS.ChartTimeSpan.fromMinutes(2);
groupBarData[0][1] = 5.8;
groupBarData[1][1] = 4.3;
groupBarData[2][1] = 2.8;
groupBarData[3][1] = 1.5;
xValues[2] = QCChartTS.ChartTimeSpan.fromMinutes(3);
groupBarData[0][2] = 5.5;
groupBarData[1][2] = 4.5;
groupBarData[2][2] = 2.5;
groupBarData[3][2] = 2.1;
xValues[3] = QCChartTS.ChartTimeSpan.fromMinutes(4);
groupBarData[0][3] = 4.1;
groupBarData[1][3] = 5.4;
groupBarData[2][3] = 4.1;
groupBarData[3][3] = 3.2;
xValues[4] = QCChartTS.ChartTimeSpan.fromMinutes(5);
groupBarData[0][4] = 3.8;
groupBarData[1][4] = 5.6;
groupBarData[2][4] = 4.3;
groupBarData[3][4] = 3.3;
let Dataset1: QCChartTS.ElapsedTimeGroupDataset =
QCChartTS.ElapsedTimeGroupDataset.newElapsedTimeGroupDataset("ElapsedTimeGroupData", xValues,
groupBarData);
```

[JavaScript]

```
let nNumGroups = 4;
let nNumPnts = 5;
let xValues = new Array<QCChartTS.ChartTimeSpan>(nNumPnts);
let groupBarData = QCChartTS.ChartSupport.newArray2D(nNumGroups, nNumPnts);

xValues[0] = QCChartTS.ChartTimeSpan.fromMinutes(1);
groupBarData[0][0] = 6.3;
groupBarData[1][0] = 3.1;
groupBarData[2][0] = 2.2;
groupBarData[3][0] = 1.8;
xValues[1] = QCChartTS.ChartTimeSpan.fromMinutes(2);
groupBarData[0][1] = 5.8;
groupBarData[1][1] = 4.3;
groupBarData[2][1] = 2.8;
groupBarData[3][1] = 1.5;
xValues[2] = QCChartTS.ChartTimeSpan.fromMinutes(3);
groupBarData[0][2] = 5.5;
groupBarData[1][2] = 4.5;
groupBarData[2][2] = 2.5;
groupBarData[3][2] = 2.1;
xValues[3] = QCChartTS.ChartTimeSpan.fromMinutes(4);
```

```
groupBarData[0][3] = 4.1;
groupBarData[1][3] = 5.4;
groupBarData[2][3] = 4.1;
groupBarData[3][3] = 3.2;
xValues[4] = QCChartTS.ChartTimeSpan.fromMinutes(5);
groupBarData[0][4] = 3.8;
groupBarData[1][4] = 5.6;
groupBarData[2][4] = 4.3;
groupBarData[3][4] = 3.3;
let Dataset1 =
QCChartTS.ElapsedTimeGroupDataset.newElapsedTimeGroupDataset("ElapsedTimeGroupData", xValues,
groupBarData);
```

Event Group Dataset

Class EventGroupDataset

```
ChartObj
   |
   +--ChartDataset
         |
         +--GroupDataset
               |
               +--EventGroupDataset
```

The **EventGroupDataset** class is the group dataset version of EventSimpleDataset. See the background information under the EventSimpleDataset. It is used to supply data to the group plotting classes: OHLC, Candlestick, GroupBar, Stacked Bar, etc..

A ChartEvent can have one or more y-values. In the case of an EventSimpleDataset, usually it will have a single value, as seen in the EventSimpleDataset programming example, one y-value for each x-value. In the EventGroupDataset, each ChartEvent object can have multiple y-values for each x-value, as seen in the programming example below..

EventGroupDataset constructors

```
public static newEventGroupDataset(sname: string, evs: ChartEvent[], numgroups: number):
EventGroupDataset

public static newEventGroupDatasetNameArrayEventsGroups(sname: string, evs: ChartEvent[],
numgroups: number): EventGroupDataset

public static newEventGroupDatasetNameArrayEventsArrayValidGroups(sname: string, evs:
ChartEvent[], valid: boolean[], ngroups: number): EventGroupDataset

public static newEventGroupDatasetNameArrayEventsArrayValidRC(sname: string, evs: ChartEvent[],
valid: boolean[], nrows: number, ncols: number): EventGroupDataset

public static newEventGroupDatasetCSV(csv: CSV, filetext: string, rowskip: number, columnskip:
number): EventGroupDataset
```

sname	Specifies the name of the dataset.
evs	An array of ChartEvent objects.

numgroups	Number of groups in the dataset
source	A text string containing CSV formatted data.
csv	An instance of a **CSV** object.
valid	An array of boolean values, specifying which array elements of x and y are valid.

Create an array of ChartEvent objects, specifying the time-stamp and y-values for each ChartEvent object and use that to initialize a EventSimpleDataset.

The next constructor creates an EventGroupDataset using the x- and y-values stored in a file that uses the CSV (Comma Separated Value) format. Only the COLUMN_MAJOR format is supported, where each row presents a ChartEvent object, and the columns are organized as: description, short description, x-axis string label, tool tip string, position, time stamp, numeric time stamp, y-value index (index for the y-value to use when the ChartEvent contains multiple y-values), and the y-values.

```
public static newEventGroupDatasetCSV(csv: CSV, source: string, rowskip: number, columnskip:
number): EventGroupDataset
```

csv	An instance of a **CSV** object.
source	The data text string
rowskip	Skip this many rows before starting the read operation.
columnskip	For each row of data, skip this many columns before starting this read operation.

You can also modify a point at a time using **setEvent.** If you need to add new points to the dataset, increasing its size, use one of the **addEvent**, or **insertEvent** methods. Delete data points using the **deleteEvent** method. In order to see the modified dataset, force the graph to redraw using **ChartView.updateDraw** method.

.

Example of creating a simple event datasets, extracted from the ChartEventExamples.CandlestickEventChart example program.

[TypeScript]

```
let nNumGroups: number = 4;
let nNumPnts: number = 50;
let weekmode: number = QCChartTS.ChartConstants.WEEK_5D;
let xValues: Date[] = new Array<Date>(nNumPnts);
let stockPriceData: number[] = new Array(nNumGroups);
let maxval: number = 0;
let minval: number = 0;
let i: number;
let currentdate: Date = new Date();
QCChartTS.ChartCalendar.setTOD(currentdate, 0, 0, 1);
theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
//  Make sure not to start on a weekend
xValues[0] = new Date(currentdate);
let eventArray: QCChartTS.ChartEvent[] = new Array<QCChartTS.ChartEvent>();
let currentEvent: QCChartTS.ChartEvent = new QCChartTS.ChartEvent();

currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
stockPriceData[3] = 25;
//  close
stockPriceData[0] = 25;
//  open
stockPriceData[1] = 26;
//  high
stockPriceData[2] = 24;
//  low
for (i = 0; (i < nNumPnts); i++) {
    let position: number = i + 1;
    xValues[i] = new Date(currentdate);
    if (i > 0) {
        stockPriceData[3] += 2 * (0.5 - QCChartTS.ChartSupport.getRandomDouble()); // close
        stockPriceData[0] += 2 * (0.5 - QCChartTS.ChartSupport.getRandomDouble()); // open
        minval = Math.min(stockPriceData[3], stockPriceData[0]);
        maxval = Math.max(stockPriceData[3], stockPriceData[0]);
        stockPriceData[1] = maxval + 1.5 * QCChartTS.ChartSupport.getRandomDouble();  // high
        stockPriceData[2] = minval - 1.5 * QCChartTS.ChartSupport.getRandomDouble();  // low
    }

    currentEvent = QCChartTS.ChartEvent.newChartEventDateTimeStampPosValues(xValues[i], position,
stockPriceData);
    currentEvent.AxisLabel = "XXX" + position.toString();
    currentEvent.ToolTip = "ToolTip" + position.toString();
    eventArray[i] = currentEvent;

    //  low
    currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
}

let Dataset1: QCChartTS.EventGroupDataset =
QCChartTS.EventGroupDataset.newEventGroupDatasetNameArrayEventsGroups("Stock Data", eventArray,
4);
let Dataset2: QCChartTS.EventSimpleDataset = Dataset1.convertToEventSimpleDatasetIndex(1);
```

[JavaScript]

```
let xValues = new Array(nNumPnts);
let stockPriceData = new Array(nNumGroups);
let maxval = 0;
let minval = 0;
let i;
let currentdate = new Date();
QCChartTS.ChartCalendar.setTOD(currentdate, 0, 0, 1);
theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
//  Make sure not to start on a weekend
xValues[0] = new Date(currentdate);
let eventArray = new Array();
let currentEvent = new QCChartTS.ChartEvent();
currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
```

```
stockPriceData[3] = 25;
//  close
stockPriceData[0] = 25;
//  open
stockPriceData[1] = 26;
//  high
stockPriceData[2] = 24;
//  low
for (i = 0; (i < nNumPnts); i++) {
    let position = i + 1;
    xValues[i] = new Date(currentdate);
    if (i > 0) {
        stockPriceData[3] += 2 * (0.5 - QCChartTS.ChartSupport.getRandomDouble()); // close
        stockPriceData[0] += 2 * (0.5 - QCChartTS.ChartSupport.getRandomDouble()); // open
        minval = Math.min(stockPriceData[3], stockPriceData[0]);
        maxval = Math.max(stockPriceData[3], stockPriceData[0]);
        stockPriceData[1] = maxval + 1.5 * QCChartTS.ChartSupport.getRandomDouble(); // high
        stockPriceData[2] = minval - 1.5 * QCChartTS.ChartSupport.getRandomDouble(); // low
    }
    currentEvent = QCChartTS.ChartEvent.newChartEventDateTimeStampPosValues(xValues[i], position,
stockPriceData);
    currentEvent.AxisLabel = "XXX" + position.toString();
    currentEvent.ToolTip = "ToolTip" + position.toString();
    eventArray[i] = currentEvent;
    //  low
    currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
}
let Dataset1 = QCChartTS.EventGroupDataset.newEventGroupDatasetNameArrayEventsGroups("Stock
Data", eventArray, 4);
let Dataset2 = Dataset1.convertToEventSimpleDatasetIndex(1);
```

4. Scaling and Coordinate Systems

- **ChartScale**
  - **LinearScale**
  - **LogScale**
  - **TimeScale**
  - **ElapsedTimeScale**
  - **EventScale**
- **UserCoordinates**
  - **WorldCoordinates**
    - **WorkingCoordinates**
      - **PhysicalCoordinates**
        - **CartesianCoordinates**
          - **PolarCoordinates**
          - **AntennaCoordinates**
          - **EventCoordinates**
        - **TimeCoordinates**
        - **ElapsedTimeCoordinates**

The starting point for all drawing in a window is the Canvas 2D device coordinate system. The coordinate system uses a default device resolution of the underlying Canvas, regardless of the output device. A Canvas maintains a viewport for the client area of the window, controlling the position and size of the drawing area in the window. Graphics output is clipped to the viewport, preventing graphics output in one window from over-writing graphics in another window. The user coordinate system for the window starts at (0,0) in the upper left corner and extends in the positive direction down and to the right.

Plot area and graph area

The *plot area* of a graph is the area where the plot data objects (line plots, bar plots, etc.) are drawn. The *graph area* is the entire area of the chart window. The graph area includes the plot area as a subset. Usually, the plot area is smaller than the graph area and resides roughly centered in the graph area. The border around the plot area is sized large enough to display the axis tick mark labels, axis titles, legends, chart titles, footers, and any other object in the graph. Create a physical coordinate system for a chart, and you are setting the minimum and maximum values for the x and y dimensions of the plot area.

Most chart objects require access to the chart coordinate system for proper positioning in the chart window. Some chart objects, axis objects in particular, often reside on the edge or outside of the plot area. Important parts of the axis, the tick marks and tick mark labels, are usually outside of the plot area. The tick marks and tick mark labels must align perfectly with the coordinate system inside the plot area.

There are many different techniques to align the coordinate system inside the plot area with the coordinate system used in drawing chart objects outside of the plot area. One technique is to maintain the physical coordinate system inside the plot area, and use a normalized coordinate system for the graph area. Whenever a chart object in the graph area, a y-axis tick mark for example, needs to be aligned with the coordinate system inside the plot area, the software converts the tick mark placement value from physical coordinates to normalized coordinates using standardized coordinate conversion routines. The drawback of this technique is what I will call the "odd pixel problem". The odd pixel problem shows up when you try map physical, normalized and user coordinate systems based on floating point numbers onto a pixel coordinate system using an integer coordinate system. Unless the corners of the plot area fall on exact pixel boundaries, converting from plot area coordinates to graph area coordinates, once translated to pixels, can be up to one pixel off.

The alternative technique, used in this software library, is to use a single coordinate system. The physical coordinate system defined for the plot area is extended in all four directions – left, right, top and bottom. It is extended so that the physical coordinates of the four corners of the plot area remain unchanged. The four corners of the graph area are assigned calculated, physical coordinate values so that when it is overlaid on to the plot area there is an exact 1:1 correspondence for all points inside the plot area. Once this calculation is made, there is no need to use the physical coordinate system assigned to the plot area. Instead, the physical coordinate system of the graph area is used instead. Chart axes objects and plotted data always align because they are plotted using the exact same physical coordinate system. The only difference is that plotted data is clipped to the plot area while axes objects are not clipped.

For example, assume a graph area with the dimensions of 400x400 units, and a plot area with the dimensions 200x200 centered inside the graph area. This implies that there is a 100 unit boundary around all four sides of the plot area. The desired chart uses a physical coordinate system of (0, 0,100,100). These coordinates apply to the plot area. Instead of using the plot area coordinate system, the coordinate system (-50,-50,150,150) is calculated and used to scale the graph area. This does not guarantee that any point plotted in plot area coordinate system will always map to the exact same pixel as the same point plotted in the graph coordinate system, the odd pixel problem still exists. We avoid the odd pixel problem by never plotting points using the plot area coordinate system, using only the graph area coordinate system instead. The calculated physical coordinate system applied to the graph area is referred to as the *working* coordinate system.

Coordinate Systems

A **QCChart2D for JavaScript/TypeScript** library uses other coordinate systems mapped onto the default Canvas device coordinate system. These other coordinate systems include world coordinates, working coordinates, and physical coordinates.

User Coordinates

The **UserCoordinates** class manages a simple viewport drawing system using the Canvas drawing classes.

World Coordinates

The **WorldCoordinates** class maps a linear, double based, coordinate system onto the integer based user coordinate system of the **UserCoordinates** class. Where the underlying user coordinate system may have an integer range (for example 0-400, 0-300 units), the world coordinate system is able to map this to a completely arbitrary, double based, linear range (for example (0.0 to 10.0, 0.0 to 10.0) . The world coordinate system applies to the entire graph window, and not just the plot area.

Working Coordinates

The **WorkingCoordinates** class manages a working coordinate system that maps the physical coordinate system of the plot area into a linear, world coordinate system applied to the whole viewport. For example, if the desired chart plot area uses a physical coordinate system of (0.0, 0.0, 100.0, 100.0) and the plot area is centered in the graph area with the plot area ½ the width and height of the graph area, then the coordinate system (-50, -50, 150, 150) is calculated and used to scale the graph area. The **WorkingCoordinates** class uses the underlying **WorldCoordinates** class to scale the viewport to the final world coordinates scale.

Physical Coordinates

The **PhysicalCoordinates** abstract class is responsible for mapping the plot area coordinate system (whether it is linear, logarithmic, date/time, polar, antenna, continuous or discontinuous) into a continuous linear coordinate system. It uses the **WorkingCoordinates** class to map this plot area coordinate system to the entire viewport. The **PhysicalCoordinates** system uses independent scale objects, derived from **ChartScale**, to manage coordinate conversions for the x- and y-dimensions. This way the x-coordinate can use one coordinate conversion object (**LinearScale**, **LogScale**, **TimeScale**) and the y-coordinate another.

There are six concrete implementations of the **PhysicalCoordinates** class: **CartesianCoordinates**, **TimeCoordinates**, **ElapsedTimeCoordinates**, **EventCoordinates**, **PolarCoordinates** and **AntennaCoordinates**. Use the **CartesianCoordinates** class for any combination of linear and logarithmic scaling for the x- and y-coordinate. Use the **TimeCoordinates** class when you want a time/date scale for the x-coordinate and a linear or logarithmic scale for the y-coordinate. Use the **EventCoordinates** if you have discontinuous, or irregular, time, suchs as that found in worldwide financial markets. Use the **ElapsedTimeCoordinates** class when you want a elapsed time scale (no date information) for the x-coordinate and a linear or logarithmic scale for the y-coordinate. Use the **PolarCoordinates** for polar coordinates where the magnitude coordinate is linear and the polar angle coordinate extends from 0 to 360 degrees (or 0 to 2*pi radians) counter-clockwise, starting at 3:00. Use the **AntennaCoordinates** for antenna coordinates where the radius value is linear and the angle coordinate extends from 0 to 360 degrees clockwise, starting at 12:00.

Normalized coordinates

Normalized coordinates are a special case of linear physical coordinates, where the linear physical scale (0.0 - 1.0, 0.0 – 1.0) is applied to either the graph area, or the plot area of the chart. ***Graph normalized***

coordinates maps the upper left corner of the graph window to the xy coordinates (0.0,0.0) and the lower right corner of the graph area to the xy coordinate (1.0,1.0). ***Plot normalized coordinates*** maps the upper left corner of the plot area to the xy coordinates (0.0,0.0) and the lower right corner of the plot area to the xy coordinates (1.0,1.0).

Important numeric considerations

Value limiting

A chart should be scaleable to any numeric range that a user wants to plot. This incorporates the entire range of floating point numbers supported by JavaScript. A range of +-10^-30 is just as valid as a range of +-10^30, even though there is 60 orders of magnitude of difference. A user can attempt to plot data with an extremely large dynamic range (the +-10^30 range) in a chart scaled for an extremely small range (the +-10^-30 range). The resulting user coordinate values resulting from such an extreme case can easily exceed the numeric range supported by the plotting functions.

Bad value checking

Invalid data often finds its way into chart. Invalid data can take many different forms. The most obvious is the introduction of numeric values that do not fit the JavaScript floating point format. These types of numbers are often found in databases and representing non-initialized or improperly initialized data. JavaScript cannot include an invalid floating point number in a calculation, so it is best to try and avoid them. Another type of invalid data is data that is a valid floating point number, but is never less considered invalid by the user. Often when data is outside of a predetermined range, it is invalid. Mark a data value in a dataset invalid using the **ChartDataset.setValidData** method. If a data value equals Number.MAX_VALUE it is also considered invalid. The software also includes a static constant, ChartConstants.rBadDataValue which can also be used to mark a bad data value. Internally, ChartConstants.rBadDataValue equals Number.MAX_VALUE.

Taking the logarithm of 0 or a negative number

If a charting package is capable of logarithmic and semi-logarithmic plotting, it must be protected against the error condition of taking the log of any number <= 0.0. This often happens when a chart is initially setup with a linear scale, with a minimum physical coordinate value of 0.0 and a maximum coordinate value equal to some large number. The user changes to a logarithmic scale, but forgets to change the minimum coordinate value of the scale from 0.0 to some positive non-zero number. The coordinate conversion routines will halt the first time the log(0.0) is in a calculation. The same is also true if the minimum coordinate value is any negative number. This software always checks for this condition and changes the minimum coordinate value using the following criteria. If the minimum coordinate value for a logarithmic scale is less than or equal to 0.0 it is assumed that the user made an error and coordinate value is set to 1.0. If the minimum coordinate value is greater than 0.0 but less than the value of MIN_LOG_VALUE, the minimum coordinate value is set to MIN_LOG_VALUE.

Positioning the Plot Area in Graph Area

The **WorkingCoordinates** class has a group of methods - **setGraphBorderFrame**, **setGraphBorderDiagonal**, and **setGraphBorderInsets** - that position the plot area of the chart in the graph viewport. Since the coordinate system scaling classes are subclasses of **WorkingCoordinates**, these methods are part of those classes. These methods are redundant and only one need be called. The default position of the plot area in the graph view port is at x = 0.2, y= 0.2, width = 0.6, height = 0.6, specified using graph normalized coordinates.

This method initializes the position and size of the plot area inside the graph area, specified using a rectangle to specify graph normalized coordinates.

setGraphBorder methods

```
 public setGraphBorderFrameRect(border: ChartRectangle2D)
```

border Specifies the rectangle defining the plot area border.

This method initializes the position and size of the plot area inside the graph area, specified using graph normalized position and size values.

```
public setGraphBorderFrame(rLeft: number, rTop: number, width: number, height: number)
```

where

rLeft	The left x-position of the plot area inside the graph area specified using graph normalized coordinates.
rTop	The top y-position of the plot area inside the graph area specified using graph normalized coordinates.
width	The width of the plot area inside the graph area specified using graph normalized coordinates.

height	The height of the plot area inside the graph area specified using graph normalized coordinates.

This method initializes the size and position of the plot area inside the graph area, specified using graph normalized values for the opposite corners of the region.

```
public setGraphBorderDiagonal(rLeft: number, rTop: number, rRight: number, rBottom: number)
```

rLeft	The left x-position of the plot area inside the graph area specified using graph normalized coordinates.
rTop	The top y-position of the plot area inside the graph area specified using graph normalized coordinates.
rRight	The right x-position of the plot area inside the graph area specified using graph normalized coordinates.
rBottom	The bottom y-position of the plot area inside the graph area specified using graph normalized coordinates.

This method initializes the insets of the plot area inside the graph area, specified using graph normalized values.

```
public setGraphBorderInsets(rLeft: number, rTop: number, rRight: number, rBottom: number)
```

rLeft	The left inset of the plot area inside the graph area specified using graph normalized coordinates.
rTop	The top inset of the plot area inside the graph area specified using graph normalized coordinates.
rRight	The right inset of the plot area inside the graph area specified using graph normalized coordinates.

rBottom	The bottom inset of the plot area inside the graph area specified using graph normalized coordinates.

The following examples all position the plot area of the chart in the upper right quadrant of the graph viewport.

[TypeScript]

```
let simpleScale: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinates(xMin, yMin, xMax, yMax);
//  Use ONE of the example below
//  Example #1
simpleScale.setGraphBorderFrame(QCChartTS.ChartRectangle2D.newChartRectangle2D(0.5, 0, 0.5,
0.5));
//  Example #2
simpleScale.setGraphBorderFrame(0.5, 0, 0.5, 0.5);
//  Example #3
simpleScale.setGraphBorderDiagonal(0.5, 0, 1, 0.5);
//  Example #4
simpleScale.setGraphBorderInsets(0.5, 0, 0, 0.5);
```

[JavaScript]

```
let simpleScale = QCChartTS.CartesianCoordinates.newCartesianCoordinates(xMin, yMin, xMax, yMax);
//  Use ONE of the example below
//  Example #1
simpleScale.setGraphBorderFrame(QCChartTS.ChartRectangle2D.newChartRectangle2D(0.5, 0, 0.5,
0.5));
//  Example #2
simpleScale.setGraphBorderFrame(0.5, 0, 0.5, 0.5);
//  Example #3
simpleScale.setGraphBorderDiagonal(0.5, 0, 1, 0.5);
//  Example #4
simpleScale.setGraphBorderInsets(0.5, 0, 0, 0.5);
```

Linear and Logarithmic Coordinate Scaling

Class CartesianCoordinates

PhysicalCoordinates

```
        |
        +-- CartesianCoordinates
```

The **CartesianCoordinates** class scales the chart plot area for a physical coordinate system that uses linear and/or logarithmic scaling.

There are three main ways to scale the plot area:

- Scale the minimum and maximum x- and y- values explicitly
- Use an auto-scale method that calculates appropriate minimum and maximum x- and y-values based on the x- and y-values in one or more datasets
- Use a combination of the first two methods. It is useful to be able to run an auto-scale function, and then change the minimum or maximum value of one or more coordinate endpoints.

CartesianCoordinates constructors

```
public static newCartesianCoordinates(rX1: number, rY1: number, rX2: number, rY2: number):
CartesianCoordinates

public static newCartesianCoordinatesScaleXY(xscale: number, yscale: number):
CartesianCoordinates
```

Linear Coordinate Scaling

The default coordinate system for the **CartesianCoordinates** class is linear for both x and y. If you already know the range for x and y for the plot area, you can scale the plot area explicitly.

The example below uses a **CartesianCoordinates** constructor to initialize the coordinates to the proper values.

CartesianCoordinates constructor with explicit scaling

[TypeScript]

```
let xMin: number = -5;
let xMax: number = 15;
let yMin: number = 0;
let yMax: number =  105;

let simpleScae: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinates(xMin, yMin, xMax, yMax);
```

[JavaScript]

```
var xMin = -5
var xMax= 15
var yMin = 0
var yMax =  105

let simpleScae = QCChartTS.CartesianCoordinates.newCartesianCoordinates(xMin, yMin, xMax, yMax);
```

Another technique uses the default constructor and scales the coordinates using the **CartesianCoordinates.setCoordinateBounds** method.

Example of explicit scaling of a CartesianCoordinates object using the CartesianCoordinates.setCoordinateBounds method

[TypeScript]

```
let xMin: number = -5;
let xMax: number = 15;
let yMin: number = 0;
let yMax: number =  105;
let simpleScae: QCChartTS.CartesianCoordinates = new QCChartTS.CartesianCoordinates();

simpleScale.setCoordinateBounds(xMin, yMin, xMax, yMax);
```

[JavaScript]

```
let xMin: number = -5;
let xMax: number = 15;
let yMin: number = 0;
let yMax: number =  105;
let simpleScae = new QCChartTS.CartesianCoordinates();

simpleScale.setCoordinateBounds(xMin, yMin, xMax, yMax);
```

It is possible to scale the bounds of the coordinate system based on the data values in a dataset. There are constructors and methods that take a single dataset and others that take an array of datasets.

```
public autoScaleDatasetRoundMode(dataset: ChartDataset,
        nroundmodex: number, nroundmodey: number): void;

public autoScaleDataset(dataset: ChartDataset): void;

public autoScaleDatasetsRoundMode(datasets: ChartDataset[],
        nroundmodex: number, nroundmodey: number): void;

public autoScaleDatasets(datasets: ChartDataset[]): void;
```

dataset	The source dataset used to scale the coordinate system.
datasets	Ann array of source source datasets used to scale the coordinate system.
nroundmodex	The rounding mode for the x-axis auto-scaling. Use one of the auto-scale rounding mode constants: AUTOAXES_FAR, AUTOAXES_NEAR, AUTOAXES_EXACT
nroundmodey	The rounding mode for the yaxis auto-scaling. Use one of the auto-scale rounding mode constants: AUTOAXES_FAR, AUTOAXES_NEAR, AUTOAXES_EXACT

Example of auto-scaling a CartesianCoordinates object using a single dataset

[TypeScript]

```
let xData: number[] = [1,2,3,4,5,6,7,8,9,10];
let yData: number[] = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];

let dataset: QCChartTS.SimpleDataset = QCChartTS.SimpleDataset.newSimpleDataset("Sales", xData,
yData);
let simpleScale: QCChartTS.CartesianCoordinates = new QCChartTS.CartesianCoordinates();
simpleScale.autoScale(dataset);
```

[JavaScript]

```
let xData = [1,2,3,4,5,6,7,8,9,10];
let yData = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];

let dataset = QCChartTS.SimpleDataset.newSimpleDataset("Sales", xData, yData);
let simpleScale = new QCChartTS.CartesianCoordinates();
simpleScale.autoScale(dataset);
```

You can control the “tightness” of the auto-scale values about the dataset values using other versions of the **CartesianCoordinates.autoScale** method that take rounding mode parameters.

Example of auto-scaling a CartesianCoordinates object using a single dataset and explicit rounding mode parameters

```
simpleScale.autoScale(dataset,  QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR)
```

You can auto-scale the bounds of the coordinate system using a dataset, and then explicitly modify the range the auto-scale selected. There are methods that set the minimum and maximum values of the x- and y-scales. This way you can use the auto-scale methods for the values of one scale (the y-scale in the example below), but explicitly set the values for the other scale (the x-scale in the example below).

Example of modifying the minimum and maximum values selected by an auto-scale method.

[TypeScript]

```
let xData: number[] = [1,2,3,4,5,6,7,8,9,10];
let yData: number[] = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];

let dataset: QCChartTS.SimpleDataset = QCChartTS.SimpleDataset.newSimpleDataset("Sales", xData,
yData);
let simpleScale: QCChartTS.CartesianCoordinates = new QCChartTS.CartesianCoordinates();


simpleScale.autoScale(dataset);
simpleScale.setScaleStopX(10);
simpleScale.setScaleStartX(1.0);
```

[JavaScript]

```
let xData: = [1,2,3,4,5,6,7,8,9,10];
let yData: = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];

let dataset = QCChartTS.SimpleDataset.newSimpleDataset("Sales", xData, yData);
let simpleScale = new QCChartTS.CartesianCoordinates();


simpleScale.autoScale(dataset);
simpleScale.setScaleStopX(10);
simpleScale.setScaleStartX(1.0);
```

The auto-scale methods that use an array of datasets to determine the proper range are very similar.

Example of auto-scaling a CartesianCoordinates object using the multiple datasets

[TypeScript]

```
let xData1: number[] = [1,2,3,4,5,6,7,8,9,10];
let yData1: number[] = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];
let xData2: number[] = [10,9,8,7,6,5,4,3,2,1];
let yData2: number[] = [20, 12, 43, 54, 15, 26, 63, 25, 24, 19];
let xData3: number[] = [5,6,7,6,5,4,5,6,7,8];
let yData3: number[] = [30, 52, 13, 64, 25, 76, 13, 35, 24, 19];

let dataset1: QCChartTS.SimpleDataset  =
QCChartTS.SimpleDataset.newSimpleDataset("Sales1",xData1,yData1);
let dataset2: QCChartTS.SimpleDataset  =
QCChartTS.SimpleDataset.newSimpleDataset("Sales2",xData2,yData2);
let dataset3: QCChartTS.SimpleDataset  =
QCChartTS.SimpleDataset.newSimpleDataset("Sales3",xData3,yData3);
let datasetsArray: QCChartTS.SimpleDataset []  = new Array<QCChartTS.SimpleDataset>(3);
datasetsArray[0] = dataset1;
datasetsArray[1] = dataset2;
datasetsArray[2] = dataset3;
let simpleScale: QCChartTS.CartesianCoordinates   = new QCChartTS.CartesianCoordinates();
simpleScale.autoScaleDatasets(datasetsArray);
```

[JavaScript]

```
let xData1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let yData1 = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];
let xData2 = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1];
let yData2 = [20, 12, 43, 54, 15, 26, 63, 25, 24, 19];
let xData3 = [5, 6, 7, 6, 5, 4, 5, 6, 7, 8];
let yData3 = [30, 52, 13, 64, 25, 76, 13, 35, 24, 19];
let dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("Sales1", xData1, yData1);
let dataset2 = QCChartTS.SimpleDataset.newSimpleDataset("Sales2", xData2, yData2);
let dataset3 = QCChartTS.SimpleDataset.newSimpleDataset("Sales3", xData3, yData3);
let datasetsArray = new Array(3);
datasetsArray[0] = dataset1;
datasetsArray[1] = dataset2;
datasetsArray[2] = dataset3;
let simpleScale = new QCChartTS.CartesianCoordinates();
simpleScale.autoScaleDatasets(datasetsArray);
```

There is a version of the multiple dataset auto-scale routine that also specifies rounding mode parameters.

```
simpleScale.autoScaleDatasetsRoundMode(datasetsArray,QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR)
```

Logarithmic Coordinate Scaling

The previous examples assume that both the x- and y- scales are linear. If the x and/or y scale are to be logarithmic, then use the **CartesianCoordinates** constructor that has scale mode parameters.

Example of explicit scaling of three different logarithmic CartesianCoordinates objects

[TypeScript]

```
let xMin: number = 1;
let xMax: number = 1000;
let yMin: number = 0.2;
let yMax: number =  2000;
let logYScale: QCChartTS.CartesianCoordinates  =

QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LOG_SCALE);
logYScale.setCoordinateBounds(xMin, yMin, xMax, yMax);

let logXScale: QCChartTS.CartesianCoordinates  =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LOG_SCALE,
QCChartTS.ChartConstants.LINEAR_SCALE);
logXScale.setCoordinateBounds(xMin, yMin, xMax, yMax);

let logXLogYScale:  QCChartTS.CartesianCoordinates  =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LOG_SCALE,
QCChartTS.ChartConstants.LOG_SCALE);
logXLogYScale.setCoordinateBounds(xMin, yMin, xMax, yMax);
```

[JavaScript]

```
let xData1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let yData1 = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];
let xData2 = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1];
let yData2 = [20, 12, 43, 54, 15, 26, 63, 25, 24, 19];
let xData3 = [5, 6, 7, 6, 5, 4, 5, 6, 7, 8];
let yData3 = [30, 52, 13, 64, 25, 76, 13, 35, 24, 19];
let dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("Sales1", xData1, yData1);
let dataset2 = QCChartTS.SimpleDataset.newSimpleDataset("Sales2", xData2, yData2);
let dataset3 = QCChartTS.SimpleDataset.newSimpleDataset("Sales3", xData3, yData3);
let datasetsArray = new Array(3);
datasetsArray[0] = dataset1;
datasetsArray[1] = dataset2;
datasetsArray[2] = dataset3;
let simpleScale = new QCChartTS.CartesianCoordinates();
simpleScale.autoScaleDatasetsRoundMode(datasetsArray);
```

Note: When you explicitly scale the minimum and maximum values for a scale set to logarithmic coordinates, make sure you use valid values, i.e. non-negative values greater than 0.0.

The auto-scale routines work for both linear and logarithmic scales. If you use the auto-scale methods, it is important that you call the auto-scale methods **after** you establish if a scale is linear or logarithmic. This is because the auto-scale routines will allow zero and negative values for the minimum and maximum of a linear scale, but not a logarithmic scale. If you call the auto-scale routines first, while the scales are set to the default linear scale mode, and change one or both of the scales to logarithmic mode, you can introduce invalid negative and zero values into the logarithmic coordinate system.

Example of auto-scaling a CartesianCoordinates object that has a logarithmic y-scale, using a single dataset

[TypeScript]

```
let xData: number[] = [2,3,4,5,6,7,8,9];
let yData: number[] = [ 2, 33, 440, 5554, 46123, 332322, 5435641, 64567551];

let dataset: QCChartTS.SimpleDataset  = QCChartTS.SimpleDataset.newSimpleDataset("Sales", xData,
yData);
let simpleScale: QCChartTS.CartesianCoordinates  =

QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LOG_SCALE);
simpleScale.autoScaleDataset(dataset);
simpleScale.setScaleStopX(10);
simpleScale.setScaleStartX(1.0);
```

[JavaScript]

```
let xData = [2, 3, 4, 5, 6, 7, 8, 9];
let yData = [2, 33, 440, 5554, 46123, 332322, 5435641, 64567551];
let dataset = QCChartTS.SimpleDataset.newSimpleDataset("Sales", xData, yData);
let simpleScale =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LOG_SCALE);
simpleScale.autoScaleDataset(dataset);
simpleScale.setScaleStopX(10);
simpleScale.setScaleStartX(1.0);
```

Coordinate Systems using times and dates

Many charting applications use data where the x- or y-coordinate values are in the form of time and date records. The creation of a powerful yet flexible time and date physical coordinate system for scaling charts is a major challenge. Important design goals include:

- The programmer scales the graph using objects of the **Date** class.
- The scale should support a continuous range of date/time scaling from milliseconds to hundreds of years.
- The scale should have a 5/7 day week option.
- A day does not have to be 24 hours long; instead, it can have a specific range, for example: 8:30 AM to 4:00 PM.
- Time and date axes, used to delineate a date/time scale, must be able to display and label tick marks for days, weeks, months and years, taking into account the non-uniformity of years, months, weeks and days in the Gregorian calendar.

The Date Class

The **Date** class represents time and date information in the format used by the majority of the world. It is a universal time class, capable of representing time with a resolution of milliseconds and a dynamic range of hundreds of years. It contains time and date fields that specify an exact moment in time, i.e. November 7, 2000 20:43:22.554. Since the **Date** class represents both time of day and calendar dates, is not necessary to use separate time of day and date classes to manage data collected every few seconds, yet spans a day or more. An example of this type of date/time range is a graph of experimental data, sampled every 15 seconds, starting on June 12, 2001 11:43:00 PM and continuing until June 15, 2001 1:03:45 AM.

The major application for date/time chart scaling on the Internet is the display of financial market data. Multiply the number of on-line investors using the charting tools of on-line brokerage firms and financial web sites by the number of individual stocks, bonds, options and futures contracts that they are able to chart and it becomes apparent that the number of potential charts is almost infinite. Many stocks have more than 50 years of hourly historical information associated with them. A chart comparing the relative performance of General Electric, IBM, Dupont, Ford and Kodak, compared to the Dow Jones Industrial average, since the dawn of the computer age in 1950, will quickly highlight the above average stock market gains that can be made by investing in the right stocks. A user may start by looking at a fifty-year window on a stock, then through successive zoom operations narrow the window to two years, one month, one day, one hour and perhaps even one minute.

5 vs. 7 Day Work Weeks

Historically, western cultures use a five-day workweek. Much of the data suitable for charting includes data for only the workdays of Monday through Friday, excluding weekends. Data associated with financial markets is the most common. Stocks trade on Monday through Friday, excluding weekends and holidays. In the display of financial market information, it is not useful, and in many cases misleading, to scale a graph, using a seven-day work week, and then only plot data using five of the seven days. The gaps left in the chart waste valuable chart space. For line plots, if a line is drawn from the last data point on Friday to the first data point on Monday, the result gives a visual impression that the trend from Friday to Monday lasted two days (Friday midnight to Sunday midnight), when in fact it may have only lasted minutes. The **QCChart2D for JavaScript/TypeScript** date/time scaling, axes and auto-axes classes support dropping weekends, as an option, from the scale. One minute after Friday midnight is 12:01 AM Monday morning.

Non-24 Hour Days

Similarly, a full day of time stamped data may not fill an entire 24 hour day. Financial markets have specific trading hours. For example, the NYSE is open from 9:30 AM EST to 4:00 PM EST. Stocks traded on the NYSE will have a time stamp in this time range. Plotting NYSE stock data for several days, using data points sampled every fifteen minutes, will result in large gaps in a traditional chart, where 2/3 of the day stock trading is inactive. It is possible to treat the unused portion of the day in a manner analogous to weekends. It is possible to eliminate unused hours and minutes of a day from the chart coordinate system. The **QCChart2D for JavaScript/TypeScript** coordinate system allows the programmer to specify a starting and ending time for a days worth of data. In the NYSE stock market example the starting time is 9:30 AM EST and the ending time is 4:00 PM EST. Any data outside of this range is invalid and not plotted. In terms of the resulting chart, one minute after 4:30 PM is 9:31 AM. Combine the starting and ending time parameters, with the option of deleting weekends from consideration, and one minute after 4:30 PM Friday is 9:31 AM Monday.

Non-Uniformity of Date/Time Tick Marks

There even more complications associated with date/time scales. The axis that delineates a date/time scale must take into account the non-uniform nature of date/time tick marks and tick mark labels. A month has a variable number of days. When months are used as the major tick mark interval, and days are the minor tick mark interval, the software must be capable of plotting 28, 29, 30 or 31minor tick marks (days) for every major tick mark (months), depending on the month. Another example is the use of months as the major tick mark interval, and weeks as the minor tick mark interval. Some months will have four minor tick marks (start of each new week) while others will have five. The software also needs to take into the variable number of days/year due to leap years. Date/time axis tick marks and labels become even more complicated when the 5-day/7-day option and the working hours/day options are used.

The **QCChart2D for JavaScript/TypeScript** library uses milliseconds as the underlying time base for all date/time coordinate system. When a scale is created using two **Date** dates as end points, the software calculates the number of milliseconds seconds between the starting date and the ending date. If the coordinate system is based on a 5-day week, the milliseconds associated with the missing weekends are not counted. If the coordinate system does not use 24 hours a day, the milliseconds associated with the missing part of the day are not counted. A linear coordinate system is scaled using the range of calculated milliseconds. Data points plotted in this coordinate system have their date/time value converted to milliseconds seconds by subtracting the starting date of the scale from the data point date, making sure to exclude the seconds associated with weekends and fractional days, if necessary. The data point is plotted in the milliseconds based linear coordinate system. Since the **QCChart2D for JavaScript/TypeScript** library uses milliseconds as the underlying time base, the minimum allowable displayable range is one millisecond. For ranges smaller than one second, the programmer needs to convert the Date values to seconds or milliseconds and use the **CartesianCoordinates** class to scale the chart. You loose the date/time formatting of the axis labels, but this should not matter if you are dealing in the sub millisecond realm.

Class TimeCoordinates

```
PhysicalCoordinates
        |
        +-- TimeCoordinates
```

The **TimeCoordinates** class scales the chart plot area for physical coordinate systems that use date/time scaling. The basic techniques are essentially the same as those used with linear and logarithmic scaling; only the **TimeCoordinates** class uses **Date** dates in place of numeric values for the x- or y-axis scale values. The minimum and maximum values of the x- and y-scales can be set explicitly, set using one of the auto-scale methods, or set using a combination of the two.

TimeCoordinates constructors

```
public static newTimeCoordinates(dstart: Date,
    y1: number,
    dstop: Date,
    y2: number): TimeCoordinates
    public static newTimeCoordinatesTime(dstart: Date,
        y1: number,
        dstop: Date,
        y2: number): TimeCoordinates

public static newTimeCoordinatesTimeY(  x1: number,
    dstart: Date,
    x2: number,
    dstop: Date): TimeCoordinates

public static newTimeCoordinatesDateWeekType(dstart: Date,
        y1: number,
        dstop: Date,
        y2: number,
        nweektype: number): TimeCoordinates

public static newTimeCoordinatesDateTimeAxisWeekType(dstart: Date,
        y1: number,
        dstop: Date,
        y2: number,
        ntimeaxis: number,
        nweektype: number): TimeCoordinates


public static newTimeCoordinatesXYScale(xscale: number, yscale: number): TimeCoordinates

public static newTimeCoordinatesDateStartTimeWeekType(dstart: Date,
        starttime: number,
        y1: number,
        dstop: Date,
        stoptime: number,
        y2: number,
        ntimeaxis: number,
        nweektype: number): TimeCoordinates
```

dstart	Sets the starting date value for the x-axis of the plotting area physical coordinate system.
starttime	Sets the start time used for all days in the plotting area.
y1	Sets the lower left y-value for the plotting area physical coordinate system.
dstop	Sets the ending date value for the x-axis of the plotting area physical coordinate system.

stoptime	Sets the stop time used for all days in the plotting area.
y2	Sets the upper right y-value for the plotting area physical coordinate system.
ntimeaxis	Specifies whether the time axis is the x-axis or the y-axis.
x1	Sets the lower left x-value for the plotting area physical coordinate system.
x2	Sets the upper right x-value for the plotting area physical coordinate system.
nweektype	Specifies the current week mode for calendar calculations. Use one of the time/date week constants: WEEK_7D or WEEK_5D.

The default coordinate system for the **TimeCoordinates** class is time for the x-scale and linear for the y-scale scale. The y-scale can be logarithmic and you can set that mode using the explicit scale mode version of the **TimeCoordinates** constructor. If you already know the range for x and y for the plot area, you can scale the plot area explicitly. In the example below a **TimeCoordinates** constructor initializes the coordinates to the proper values.

TimeCoordinates constructor with explicit scaling of a time based x-scale, and a numeric based y-scale.

[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);
```

[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);
```

TimeCoordinates constructor with explicit scaling of a numeric based x-scale, and a time based y-scale.

[TypeScript]

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

let simpleTimeScale:QCChartTS.TimeCoordinates  =
QCChartTS.TimeCoordinates.newTimeCoordinatesTimeY(xMin, yMin, xMax, yMax);
```

[JavaScript]

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

Another technique uses the default constructor and scales the coordinates using the **TimeCoordinates.setTimeCoordinateBounds** method.

Example of explicit scaling of a TimeCoordinates object (time x-scale and numeric y-scale) using the TimeCoordinates.setTimeCoordinateBounds method

[TypeScript]

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

simpleTimeScale.setTimeCoordinateBounds (xMin, yMin, xMax, yMax);
```

[JavaScript]

```
var xMin As ChartCalendar = new Date(1996, QCChartTS.ChartConstants.FEBRUARY,5)
var xMax As ChartCalendar = new Date(2002, QCChartTS.ChartConstants.JANUARY,5)
var yMin = 0
var yMax =  105

var simpleTimeScale As TimeCoordinates = New TimeCoordinates()
simpleTimeScale.setTimeCoordinateBounds (xMin, yMin, xMax, yMax)
```

Example of explicit scaling of a TimeCoordinates object (numeric x-scale and time y-scale) using the TimeCoordinates.setTimeCoordinateBounds method

[TypeScript]

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

let simpleTimeScale:QCChartTS.TimeCoordinates  = new QCChartTS.TimeCoordinates();
simpleTimeScale.setTimeCoordinateBoundsTimeY(xMin, yMin, xMax, yMax);
```

[JavaScript]

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

It is possible to scale the bounds of the coordinate system based on the data values in a time-based dataset, **TimeSimpleDataset** and **TimeGroupDataset**. There are constructors and methods that take a single dataset and others that take an array of datasets.

Example of auto-scaling a TimeCoordinates object using a single dataset

[TypeScript]

```
let xData: Date[] = [new Date(1996, QCChartTS.ChartConstants.FEBRUARY, 5),
new Date(1996, QCChartTS.ChartConstants.MARCH, 5),
new Date(1996, QCChartTS.ChartConstants.APRIL, 5),
new Date(1996, QCChartTS.ChartConstants.MAY, 5),
new Date(1996, QCChartTS.ChartConstants.JUNE, 5),
new Date(1996, QCChartTS.ChartConstants.JULY, 5),
new Date(1996, QCChartTS.ChartConstants.AUGUST, 5),
new Date(1996, QCChartTS.ChartConstants.SEPTEMBER, 5),
new Date(1996, QCChartTS.ChartConstants.OCTOBER, 5),
new Date(1996, QCChartTS.ChartConstants.NOVEMBER, 5)];
let yData: number[] = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];

let dataset: QCChartTS.TimeSimpleDataset =
QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Sales", xData, yData);
let simpleTimeScale: QCChartTS.TimeCoordinates = new QCChartTS.TimeCoordinates();
simpleTimeScale.autoScaleDataset(dataset);
```

[JavaScript]

```
let xData = [new Date(1996, QCChartTS.ChartConstants.FEBRUARY, 5),
    new Date(1996, QCChartTS.ChartConstants.MARCH, 5),
    new Date(1996, QCChartTS.ChartConstants.APRIL, 5),
    new Date(1996, QCChartTS.ChartConstants.MAY, 5),
    new Date(1996, QCChartTS.ChartConstants.JUNE, 5),
    new Date(1996, QCChartTS.ChartConstants.JULY, 5),
    new Date(1996, QCChartTS.ChartConstants.AUGUST, 5),
    new Date(1996, QCChartTS.ChartConstants.SEPTEMBER, 5),
    new Date(1996, QCChartTS.ChartConstants.OCTOBER, 5),
    new Date(1996, QCChartTS.ChartConstants.NOVEMBER, 5)];
let yData = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];
let dataset = QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Sales", xData, yData);
let simpleTimeScale = new QCChartTS.TimeCoordinates();
simpleTimeScale.autoScaleDataset(dataset);
```

Using similar logic you can define a **TimeSimpleDataset** with numeric x-values and **Date** y-values. If you auto-scale the coordinate system using that dataset you will end up with a numeric x-axis and a time y-axis.

You can control the “tightness” of the auto-scale values about the dataset values using other versions of the **TimeCoordinates.autoScale** method that take rounding mode parameters.

Example of auto-scaling a TimeCoordinates object using a single dataset and explicit rounding mode parameters

```
simpleTimeScale.autoScaleRoundMode(dataset,QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR)
```

You can auto-scale the coordinate bounds using a dataset, and then explicitly modify the range the auto-scale selected. There are methods for setting the minimum and maximum values of the x- and y-scales. This way you can use the auto-scale methods for the values of one scale (the y-scale in the example below), but explicitly set the values for the other scale (the x-scale in the example below).

Example of modifying the minimum and maximum values selected by an auto-scale method

[TypeScript]

```
let xData: Date[] = [new Date(1996, QCChartTS.ChartConstants.FEBRUARY, 5),
new Date(1996, QCChartTS.ChartConstants.MARCH, 5),
new Date(1996, QCChartTS.ChartConstants.APRIL, 5),
new Date(1996, QCChartTS.ChartConstants.MAY, 5),
new Date(1996, QCChartTS.ChartConstants.JUNE, 5),
new Date(1996, QCChartTS.ChartConstants.JULY, 5),
new Date(1996, QCChartTS.ChartConstants.AUGUST, 5),
new Date(1996, QCChartTS.ChartConstants.SEPTEMBER, 5),
new Date(1996, QCChartTS.ChartConstants.OCTOBER, 5),
new Date(1996, QCChartTS.ChartConstants.NOVEMBER, 5)];
let yData: number[] = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];

let dataset: QCChartTS.TimeSimpleDataset =
QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Sales", xData, yData);
let simpleTimeScale: QCChartTS.TimeCoordinates = new QCChartTS.TimeCoordinates();
simpleTimeScale.autoScaleDataset(dataset);

simpleTimeScale.setTimeScaleStart(new Date(1996, QCChartTS.ChartConstants.JANUARY,5));
simpleTimeScale.setTimeScaleStop(new Date(1997, QCChartTS.ChartConstants.JANUARY,5));
```

[JavaScript]

```
let xData = [new Date(1996, QCChartTS.ChartConstants.FEBRUARY, 5),
    new Date(1996, QCChartTS.ChartConstants.MARCH, 5),
    new Date(1996, QCChartTS.ChartConstants.APRIL, 5),
    new Date(1996, QCChartTS.ChartConstants.MAY, 5),
    new Date(1996, QCChartTS.ChartConstants.JUNE, 5),
    new Date(1996, QCChartTS.ChartConstants.JULY, 5),
    new Date(1996, QCChartTS.ChartConstants.AUGUST, 5),
    new Date(1996, QCChartTS.ChartConstants.SEPTEMBER, 5),
    new Date(1996, QCChartTS.ChartConstants.OCTOBER, 5),
    new Date(1996, QCChartTS.ChartConstants.NOVEMBER, 5)];
let yData = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];
let dataset = QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Sales", xData, yData);
let simpleTimeScale = new QCChartTS.TimeCoordinates();
simpleTimeScale.autoScaleDataset(dataset);
simpleTimeScale.setTimeScaleStart(new Date(1996, QCChartTS.ChartConstants.JANUARY,5))
simpleTimeScale.setTimeScaleStop(new Date(1997, QCChartTS.ChartConstants.JANUARY,5))
```

The auto-scale methods that use an array of datasets to determine the proper range are very similar.

Example of auto-scaling a TimeCoordinates object using the multiple datasets

[TypeScript]

```
let xData: Date[] = [new Date(1996, QCChartTS.ChartConstants.FEBRUARY, 5),
new Date(1996, QCChartTS.ChartConstants.MARCH, 5),
new Date(1996, QCChartTS.ChartConstants.APRIL, 5),
new Date(1996, QCChartTS.ChartConstants.MAY, 5),
new Date(1996, QCChartTS.ChartConstants.JUNE, 5),
new Date(1996, QCChartTS.ChartConstants.JULY, 5),
new Date(1996, QCChartTS.ChartConstants.AUGUST, 5),
new Date(1996, QCChartTS.ChartConstants.SEPTEMBER, 5),
new Date(1996, QCChartTS.ChartConstants.OCTOBER, 5),
new Date(1996, QCChartTS.ChartConstants.NOVEMBER, 5)];

let yData1: number[] = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];
let yData2: number[] = [20, 12, 43, 54, 15, 26, 63, 25, 24, 19];
let yData3: number[] = [30, 52, 13, 64, 25, 76, 13, 35, 24, 19];

// All of the datasets reference the same xData array of ChartCalendar
// dates, though this does not have to be the case.
   let dataset1: QCChartTS.TimeSimpleDataset  =
QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Sales1",xData,yData1);
   let dataset2: QCChartTS.TimeSimpleDataset  =
QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Sales2",xData,yData2);
   let dataset3: QCChartTS.TimeSimpleDataset  =
QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Sales3",xData,yData3);
   let datasetsArray: QCChartTS.TimeSimpleDataset []  = new
Array<QCChartTS.TimeSimpleDataset>(3);
datasetsArray[0] = dataset1;
datasetsArray[1] = dataset2;
datasetsArray[2] = dataset3;
let simpleTimeScale:  QCChartTS.TimeCoordinates  = new  QCChartTS.TimeCoordinates();
simpleTimeScale.autoScaleDatasets(datasetsArray);
```

[JavaScript]

```
let xData = [new Date(1996, QCChartTS.ChartConstants.FEBRUARY, 5),
    new Date(1996, QCChartTS.ChartConstants.MARCH, 5),
    new Date(1996, QCChartTS.ChartConstants.APRIL, 5),
    new Date(1996, QCChartTS.ChartConstants.MAY, 5),
    new Date(1996, QCChartTS.ChartConstants.JUNE, 5),
    new Date(1996, QCChartTS.ChartConstants.JULY, 5),
    new Date(1996, QCChartTS.ChartConstants.AUGUST, 5),
    new Date(1996, QCChartTS.ChartConstants.SEPTEMBER, 5),
    new Date(1996, QCChartTS.ChartConstants.OCTOBER, 5),
    new Date(1996, QCChartTS.ChartConstants.NOVEMBER, 5)];
let yData1 = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];
let yData2 = [20, 12, 43, 54, 15, 26, 63, 25, 24, 19];
let yData3 = [30, 52, 13, 64, 25, 76, 13, 35, 24, 19];
// All of the datasets reference the same xData array of ChartCalendar
// dates, though this does not have to be the case.
let dataset1 = QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Sales1", xData, yData1);
let dataset2 = QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Sales2", xData, yData2);
let dataset3 = QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Sales3", xData, yData3);
let datasetsArray = new Array(3);
datasetsArray[0] = dataset1;
datasetsArray[1] = dataset2;
datasetsArray[2] = dataset3;
let simpleTimeScale = new QCChartTS.TimeCoordinates();
simpleTimeScale.autoScaleDatasets(datasetsArray);
```

There is a version of the multiple dataset auto-scale routine that also specifies rounding mode parameters.

```
simpleTimeScale.autoScaleDatasetsRoundMode(datasetsArray,QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
```

The previous examples use the **TimeCoordinates** default 7 days/week, and 24-hours/day modes. Many of the methods in the software have a *weektype* parameter. The default value of this parameter is the constant **QCChartTS.ChartConstants.WEEK_7D**. If you want the coordinate system to ignore Saturdays and Sundays, use the constant **QCChartTS.ChartConstants.WEEK_5D** constant**.** Use the methods below to establish a 5-days/week coordinate system.

TimeCoordinates constructor

```
)

public static newTimeCoordinatesDateStartTimeWeekType(dstart: Date,
  starttime: number,
  y1: number,
  dstop: Date,
  stoptime: number,
  y2: number,
  ntimeaxis: number,
  nweektype: number): TimeCoordinates
);

public static newTimeCoordinatesDateTimeAxisWeekType(dstart: Date,
        y1: number,
        dstop: Date,
        y2: number,
        ntimeaxis: number,
        nweektype: number): TimeCoordinates
```

dstart	Sets the starting date value for the x-axis of the plotting area physical coordinate system.
starttime	Sets the start time used for all days in the plotting area.
y1	Sets the lower left y-value for the plotting area physical coordinate system.
dstop	Sets the ending date value for the x-axis of the plotting area physical coordinate system.
stoptime	Sets the stop time used for all days in the plotting area.
y2	Sets the upper right y-value for the plotting area physical coordinate system.
ntimeaxis	Specifies whether the time axis is the x-axis or the y-axis.

nweektype	Specifies the current week mode for calendar calculations. Use one of the time/date week constants: WEEK_7D or WEEK_5D.

setTimeCoordinateBounds method

```
public setTimeCoordinateBoundsDates(dstart: Date,
     y1: number,
     dstop: Date,
     y2: number,
     nweektype: number)
```

setWeekType method

```
public setWeekType(weektype: number)
```

If you use the auto-scale routines, set the week type before you call the **TimeCoordinates.autoScale** method because the auto-scale routines need to take into account the week type. For example:

[TypeScript]

```
let dataset: QCChartTS.TimeSimpleDataset  =
QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Sales", xData, yData);
let simpleTimeScale: QCChartTS.TimeCoordinates  = new QCChartTS.TimeCoordinates();
simpleTimeScale.setWeekType(QCChartTS.ChartConstants.WEEK_5D);
simpleTimeScale.autoScaleDataset(dataset);
```

[JavaScript]

```
let dataset = QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Sales", xData, yData);
let simpleTimeScale = new QCChartTS.TimeCoordinates();
simpleTimeScale.setWeekType(QCChartTS.ChartConstants.WEEK_5D);
simpleTimeScale.autoScaleDataset(dataset)
```

In addition to the week type, the other major way to customize a **TimeCoordinates** coordinate system is not to use a 24-hour day. There are methods that set the starting and ending time-of-day. For example, if you are interested in plotting stock market data trading during the regular trading day of 9:30 AM to 4:00 PM, you can setup the coordinate system to only include these hours, and to treat any data outside of these hours as invalid and not to be plotted. A day can have only one continuous range. You are not able to define a day to with a valid range of 9:30 AM to 4:00 PM and 6:00 PM to 9:00 PM. Only one of these ranges is valid, or a combined range of 9:30 AM to 9:00 PM where the 4:00 PM to 6:00 PM time segment is included in the range.

TimeCoordinates constructor with time-of-day parameters

```
 public static newTimeCoordinatesDateStartTimeWeekType(dstart: Date,
        starttime: number,
        y1: number,
        dstop: Date,
        stoptime: number,
        y2: number,
        ntimeaxis: number,
        nweektype: number): TimeCoordinates
```

TimeCoordinates constructor example for a 5 day/week and 9:30 AM to 4:00 PM time-of-day range

[TypeScript]

```
let  starttod: number = (9 * 60 + 30) * 60 * 1000; // msecs cooresponding to 9:30 AM
let  stoptod: number = 16 * 60 * 60 * 1000; // msecs cooresponding to 4:00 PM
let  dstart: Date = new Date(1996,QCChartTS.ChartConstants.FEBRUARY,5);
let  dstop: Date = new Date(1997, QCChartTS.ChartConstants.JANUARY,5);
let  y1: number = 0.0;
let  y2: number = 55.0;
let   stockTimeScale: QCChartTS.TimeCoordinates =
  QCChartTS.TimeCoordinates.newTimeCoordinatesDateStartTimeWeekType(dstart, starttod,  y1, dstop,
stoptod , y2,
    QCChartTS.ChartConstants.X_AXIS, QCChartTS.ChartConstants.WEEK_5D) ;
```

[JavaScript]

```
let starttod = (9 * 60 + 30) * 60 * 1000; // msecs cooresponding to 9:30 AM
let stoptod = 16 * 60 * 60 * 1000; // msecs cooresponding to 4:00 PM
let dstart = new Date(1996, QCChartTS.ChartConstants.FEBRUARY, 5);
let dstop = new Date(1997, QCChartTS.ChartConstants.JANUARY, 5);
let y1 = 0.0;
let y2 = 55.0;
let stockTimeScale = QCChartTS.TimeCoordinates.newTimeCoordinatesDateStartTimeWeekType(dstart,
starttod, y1, dstop, stoptod, y2, QCChartTS.ChartConstants.X_AXIS,
QCChartTS.ChartConstants.WEEK_5D);
```

Another technique uses the default constructor and scales the coordinates using the **TimeCoordinates.setTimeCoordinateBounds** method.

setTimeCoordinateBounds Method

```
public setTimeCoordinateBoundsTOD(dstart: Date,
        starttod: number,
        y1: number,
        dstop: Date,
        stoptod: number,
        y2: number,
        nweektype: number)
)

public setTimeCoordinateBoundsTimeYTOD(
        x1: number,
        dstart: Date,
        starttod: number,
        x2: number,
        dstop: Date,
        stoptod: number,
        nweektype: number)
```

dstart	Sets the starting time-of-day and date value for the x-axis of the plotting area physical coordinate system.
starttod	Sets the starting time (in msecs) for the range of hours to use for a day
y1	Sets the lower left y-value for the plotting area physical coordinate system.
dstop	Sets the ending date value for the x-axis of the plotting area physical coordinate system.
stoptod	Sets the ending time-of-day (in msecs) for the range of hours to use for a day
y2	Sets the upper right y-value for the plotting area physical coordinate system.
nweektype	Specifies the current week mode for calendar calculations. Use one of the time/date week constants: WEEK_7D or WEEK_5D.

setTimeCoordinateBounds example for a 5 day/week and 9:30 AM to 4:00 PM time-of-day range setWeekType method

[TypeScript]

```
long starttod = (9 * 60 + 30) * 60 * 1000; // msecs cooresponding to 9:30 AM
long stoptod = 16 * 60 * 60 * 1000; // msecs cooresponding to 4:00 PM
ChartCalendar dstart = new Date(1996, QCChartTS.ChartConstants.FEBRUARY,  5);
ChartCalendar dstop = new Date(1997, QCChartTS.ChartConstants.JANUARY,  5);
double y1 = 0.0;
double y2 = 55.0;
TimeCoordinates stockTimeScale;

stockTimeScale = new TimeCoordinates();
stockTimeScale.setTimeCoordinateBounds(dstart, starttod,  y1,
                                       dstop, stoptod , y2,
                                      QCChartTS.ChartConstants.WEEK_5D);
```

[JavaScript]

```
let starttod = (9 * 60 + 30) * 60 * 1000; // msecs cooresponding to 9:30 AM
let stoptod = 16 * 60 * 60 * 1000; // msecs cooresponding to 4:00 PM
let dstart = new Date(1996, QCChartTS.ChartConstants.FEBRUARY, 5);
let dstop = new Date(1997, QCChartTS.ChartConstants.JANUARY, 5);
let y1 = 0.0;
let y2 = 55.0;
let stockTimeScale = new QCChartTS.TimeCoordinates();
stockTimeScale.setTimeCoordinateBoundsTOD(dstart, starttod, y1, dstop, stoptod, y2,
QCChartTS.ChartConstants.WEEK_5D);
```

If you use the auto-scale routines, set the week type and the time-of-day range before you call the **TimeCoordinates.autoScale** method because the auto-scale routines need to take into account the number of seconds per day and the week type. For example:

[TypeScript]

```
let Dataset1: QCChartTS.TimeSimpleDataset  =
QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("First",tradingDay,stockPrice);

let startTime: number = (9 * 60 + 30) * 60 * 1000; // msecs cooresponding to 9:30 AM
let stopTime: number = 16 * 60 * 60 * 1000; // msecs cooresponding to 4:00 PM

let stockTimeScale: QCChartTS.TimeCoordinates  = new QCChartTS.TimeCoordinates();
stockTimeScale.setWeekType (QCChartTS.ChartConstants.WEEK_5D);
stockTimeScale.setScaleStartTOD(startTime);
stockTimeScale.setScaleStopTOD(stopTime);
stockTimeScale.autoScaleDatasetRoundMode(Dataset1,QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR)
```

[JavaScript]

```
let Dataset1 = QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("First", tradingDay, stockPrice);
let startTime = (9 * 60 + 30) * 60 * 1000; // msecs cooresponding to 9:30 AM
let stopTime = 16 * 60 * 60 * 1000; // msecs cooresponding to 4:00 PM
let stockTimeScale = new QCChartTS.TimeCoordinates();
stockTimeScale.setWeekType(QCChartTS.ChartConstants.WEEK_5D);
stockTimeScale.setScaleStartTOD(startTime);
stockTimeScale.setScaleStopTOD(stopTime);
stockTimeScale.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
```

Class ElapsedTimeCoordinates

```
PhysicalCoordinates
       |
       +-- CartesianCoordinates
              |
              +-- ElapsedTimeCoordinates
```

The **ElapsedTimeCoordinates** class scales the chart plot area for a physical coordinate system which uses an elapsed time scale in combination with a linear or logarithmic scaling. The elapsed time scale uses milliseconds as the time base, so all time values should be represented using their milliseconds equivalent value, i.e. a value of 30 seconds is represented by the value 30000. The axis labeling class, **ElapsedTimeAxisLabels**, converts the millisecond values to equivalent seconds values, i.e., the value 30000 milliseconds will be displayed as 30 seconds in the format “00:00:30”.

There are three main ways to scale the plot area:

- Scale the minimum and maximum x- and y- values explicitly. You can scale the elapse time axis using **ChartTimeSpan** objects, or using millisecond values, i.e. an elapsed time range of 0 – 30 seconds would be scaled for 0-30000.
- Use an auto-scale method to calculates appropriate minimum and maximum x- and y-values based on the x- and y-values in one or more datasets
- Use a combination of the first two methods. It is useful to be able to run an auto-scale function, and then change the minimum or maximum value of one or more coordinate endpoints.

ElapsedTime Coordinate Scaling

The default coordinate system for the **ElapsedTimeCoordinates** class is elapsed time for the x-dimension and linear for the y dimension. If you already know the range for x and y for the plot area, you can scale the plot area explicitly.

ElapsedTimeCoordinates Constructors

```
public static newElapsedTimeCoordinatesMinMax(rX1: number, rY1: number, rX2: number, rY2:
number): ElapsedTimeCoordinates

public static newElapsedTimeCoordinates(rX1: number, rY1: number, rX2: number, rY2: number):
ElapsedTimeCoordinates

public static newElapsedTimeCoordinatesTimeSpanX(rX1: ChartTimeSpan, rY1: number, rX2:
ChartTimeSpan, rY2: number): ElapsedTimeCoordinates

public static newElapsedTimeCoordinatesXYScale(xscale: number, yscale: number):
ElapsedTimeCoordinates {
```

rX1	Sets the lower left x-value for the plotting area physical coordinate system.
rY1	Sets the lower left y-value for the plotting area physical coordinate system.
rX2	Sets the upper right x-value for the plotting area physical coordinate system.
rY2	Sets the upper right y-value for the plotting area physical coordinate system.
xscale	Sets the x-coordinate system to either linear or logarithmic scaling. Use one of the scaling constants: LINEAR_SCALE or LOG_SCALE.
yscale	Sets the y-coordinate system to either linear or logarithmic scaling. Use one of the scaling constants: LINEAR_SCALE or LOG_SCALE.

The example below uses a **ElapsedTimeCoordinates** constructor to initialize the coordinates to the proper values.

Scale for elapsed time using the ElapsedTimeCoordinates constructor with explicit scaling for a range of 30 seconds.

[TypeScript]

```
let xMin: number = 0; // starting elapsed time is 0
let xMax: number = 30 * 1000; // ending elpase time is 30 seconds
let yMin: number = 0;
let yMax: number =  105;

let simpleScale: QCChartTS.ElapsedTimeCoordinates =
QCChartTS.ElapsedTimeCoordinates.newElapsedTimeCoordinates(xMin, yMin, xMax, yMax);

let xTSMin: QCChartTS.ChartTimeSpan  = QCChartTS.ChartTimeSpan.fromSeconds(0); // starting
elapsed time is 0
let xTSMax: QCChartTS.ChartTimeSpan  = QCChartTS.ChartTimeSpan.fromSeconds(30); // ending elpase
time is 30 seconds
```

```
simpleScale = QCChartTS.ElapsedTimeCoordinates.newElapsedTimeCoordinatesTimeSpanX(xTSMin, yMin,
xTSMax, yMax);
```

[JavaScript]

```
let xMin = 0; // starting elapsed time is 0
let xMax = 30 * 1000; // ending elpase time is 30 seconds
let yMin = 0;
let yMax = 105;
let simpleScale = QCChartTS.ElapsedTimeCoordinates.newElapsedTimeCoordinates(xMin, yMin, xMax,
yMax);
let xTSMin = QCChartTS.ChartTimeSpan.fromSeconds(0); // starting elapsed time is 0
let xTSMax = QCChartTS.ChartTimeSpan.fromSeconds(30); // ending elpase time is 30 seconds
simpleScale = QCChartTS.ElapsedTimeCoordinates.newElapsedTimeCoordinatesTimeSpanX(xTSMin, yMin,
xTSMax, yMax););
```

Another technique uses the default constructor and scales the coordinates using the **ElapsedTimeCoordinates .setCoordinateBounds** method.

Example of explicit scaling of a ElapsedTimeCoordinates object using the ElapsedTimeCoordinates.setCoordinateBounds method

[TypeScript]

```
let xMin: number = 0; // starting elapsed time is 0
let xMax: number = 30 * 1000; // ending elpase time is 30 seconds
let yMin: number = 0;
let yMax: number =  105;

let simpleScale: QCChartTS.ElapsedTimeCoordinates = new QCChartTS.ElapsedTimeCoordinates ();

simpleScale.setCoordinateBounds(xMin, yMin, xMax, yMax);
```

[JavaScript]

```
let xMin = 0; // starting elapsed time is 0
let xMax = 30 * 1000; // ending elpase time is 30 seconds
let yMin = 0;
let yMax = 105;
let simpleScale: QCChartTS.ElapsedTimeCoordinates = new QCChartTS.ElapsedTimeCoordinates ();

simpleScale.setCoordinateBounds(xMin, yMin, xMax, yMax);
```

It is possible to scale the bounds of the coordinate system based on the data values in a dataset. There are constructors and methods that take a single dataset and others that take an array of datasets.

Example of auto-scaling a ElapsedTimeCoordinates object using a single dataset

[TypeScript]

```
let xData: number[] = [1000,2000,3000,4000,5000,6000,7000,8000,9000,10000];
let yData: number[] = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];
let dataset: QCChartTS.ElapsedTimeSimpleDataset =
QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDatasetXNumber ("Sales", xData, yData);
let simpleScale: QCChartTS.ElapsedTimeCoordinates   = new QCChartTS.ElapsedTimeCoordinates ();
simpleScale.autoScaleDataset(dataset);
```

[JavaScript]

```
let xData = [1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000];
let yData = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];
let dataset = QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDatasetXNumber("Sales",
xData, yData);
let simpleScale = new QCChartTS.ElapsedTimeCoordinates();
simpleScale.autoScaleDataset(dataset);
```

You can control the “tightness” of the auto-scale values about the dataset values using other versions of the **ElapsedTimeCoordinates .autoScale** method that take rounding mode parameters.

```
public autoScale(nroundmodeX: number, nroundmodeY: number)

public autoScaleDatasetRoundModes(dataset: ChartDataset,nroundmodeX: number, nroundmodeY: number)
public autoScaleDataset(dataset: ChartDataset)

public autoScaleDatasetsRoundModes(datasets: ChartDataset[],nroundmodeX: number, nroundmodeY:
number)

public autoScaleDatasets(datasets: ChartDataset[])

private calcAutoScaleDatasetRoundModes(dataset: ChartDataset,nroundmodeX: number, nroundmodeY:
number)

private calcAutoScaleDatasetsRoundModes(datasets: ChartDataset[],nroundmodeX: number,nroundmodeY:
number)
```

nroundmodeX	Sets the auto-scale mode for the x-coordinate. Use one of the auto-scale rounding mode constants:AUTOAXES_FAR,AUTOAXES_NEAR,AUTOAXES_EXACT.
nroundmodeY	Sets the auto-scale mode for the y-coordinate. Use one of the auto-scale rounding mode constants:AUTOAXES_FAR,AUTOAXES_NEAR,AUTOAXES_EXACT.
dataset	The dataset used as the basis for the new coordinate system.
datasets	The array of dataset used as the basis for the new coordinate system.
nroundmodeX	Sets the auto-scale mode for the x-coordinate. Use one of the auto-scale rounding mode constants:AUTOAXES_FAR,AUTOAXES_NEAR,AUTOAXES_EXACT.
nroundmodeY	Sets the auto-scale mode for the y-coordinate. Use one of the auto-scale rounding mode constants: AUTOAXES_FAR,AUTOAXES_NEAR,AUTOAXES_EXACT.

Example of auto-scaling a ElapsedTimeCoordinates object using a single dataset and explicit rounding mode parameters

```
simpleScale.autoScaleDatasetRoundMode(dataset,  QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR)
```

You can auto-scale the bounds of the coordinate system using a dataset, and then explicitly modify the range the auto-scale selected. There are methods that set the minimum and maximum values of the x- and y-scales. This way you can use the auto-scale methods for the values of one scale (the y-scale in the example below), but explicitly set the values for the other scale (the x-scale in the example below).

Example of modifying the minimum and maximum values selected by an auto-scale method.

[TypeScript]

```
let xData: number[] = [1000,2000,3000,4000,5000,6000,7000,8000,9000,10000];
let yData: number[] = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];

let dataset: QCChartTS.ElapsedTimeSimpleDataset =
QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDatasetXNumber ("Sales", xData, yData);
let simpleScale: QCChartTS.ElapsedTimeCoordinates   = new QCChartTS.ElapsedTimeCoordinates ();
simpleScale.autoScaleDataset(dataset)
simpleScale.setScaleStartY(0);
simpleScale.setScaleStopY(100.0);
```

[JavaScript]

```
let xData = [1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000];
let yData = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];
let dataset = QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDatasetXNumber("Sales",
xData, yData);
let simpleScale = new QCChartTS.ElapsedTimeCoordinates();
simpleScale.autoScaleDataset(dataset);
simpleScale.setScaleStartY(0)
simpleScale.setScaleStopY(100.0)
```

The auto-scale methods that use an array of datasets to determine the proper range are very similar.

Example of auto-scaling a ElapsedTimeCoordinates object using the multiple datasets

[TypeScript]

```
let xData: number[]1 = [1000,2000,3000,4000,5000,6000,7000,8000,9000,10000];
let xData1: number[] = [1000,2000,3000,4000,5000,6000,7000,8000,9000,10000];
let yData1: number[] = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];
let xData2: number[] = [1000,2000,3000,4000,5000,6000,7000,8000,9000,10000];
let yData2: number[] = [20, 12, 43, 54, 15, 26, 63, 25, 24, 19];
let xData3: number[] = [1000,2000,3000,4000,5000,6000,7000,8000,9000,10000];
let yData3: number[] = [30, 52, 13, 64, 25, 76, 13, 35, 24, 19];

let dataset1: QCChartTS.ElapsedTimeSimpleDataset  =
QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDatasetXNumber ("Sales1",xData1,yData1);
let dataset2:  QCChartTS.ElapsedTimeSimpleDataset  =
QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDatasetXNumber("Sales2",xData2,yData2);
let dataset3: QCChartTS.ElapsedTimeSimpleDataset  =
QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDatasetXNumber("Sales3",xData3,yData3);
let datasetsArray : QCChartTS.ElapsedTimeSimpleDataset []  = new
Array<QCChartTS.ElapsedTimeSimpleDataset>(3);
datasetsArray[0] = dataset1;
datasetsArray[1] = dataset2;
```

```
datasetsArray[2] = dataset3;
let simpleScale: QCChartTS.ElapsedTimeCoordinates   = new QCChartTS.ElapsedTimeCoordinates ();
simpleScale.autoScaleDatasets(datasetsArray);
```

[JavaScript]

```
let xData1 = [1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000];
let yData1 = [10, 22, 33, 44, 55, 46, 33, 25, 14, 9];
let xData2 = [1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000];
let yData2 = [20, 12, 43, 54, 15, 26, 63, 25, 24, 19];
let xData3 = [1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000];
let yData3 = [30, 52, 13, 64, 25, 76, 13, 35, 24, 19];
let dataset1 = QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDatasetXNumber("Sales1",
xData1, yData1);
let dataset2 = QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDatasetXNumber("Sales2",
xData2, yData2);
let dataset3 = QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDatasetXNumber("Sales3",
xData3, yData3);
let datasetsArray = new Array(3);
datasetsArray[0] = dataset1;
datasetsArray[1] = dataset2;
datasetsArray[2] = dataset3;
let simpleScale = new QCChartTS.ElapsedTimeCoordinates();
simpleScale.autoScaleDatasets(datasetsArray);
```

There is a version of the multiple dataset auto-scale routine that also specifies rounding mode parameters.

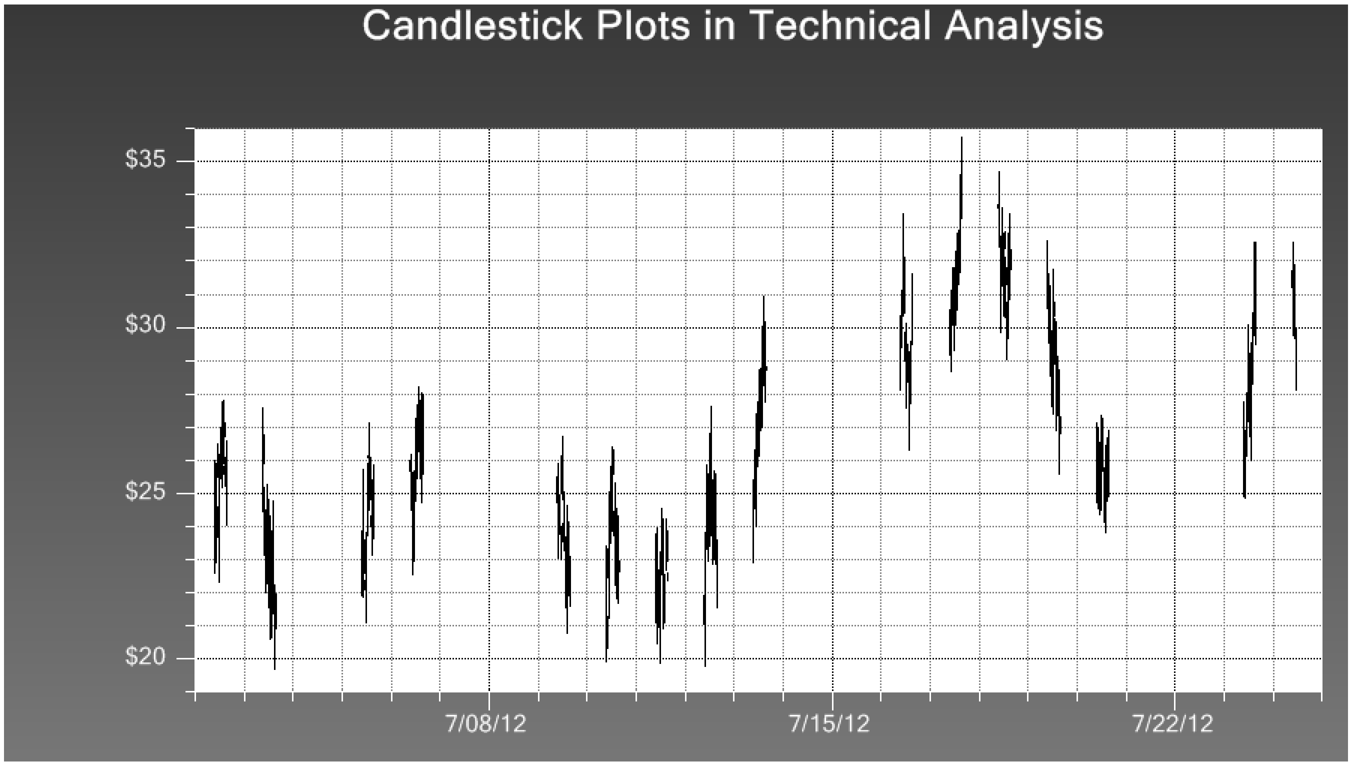
```
simpleScale.autoScaleDatasetsRoundMode(datasetsArray,QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR)
```

Class EventCoordinates

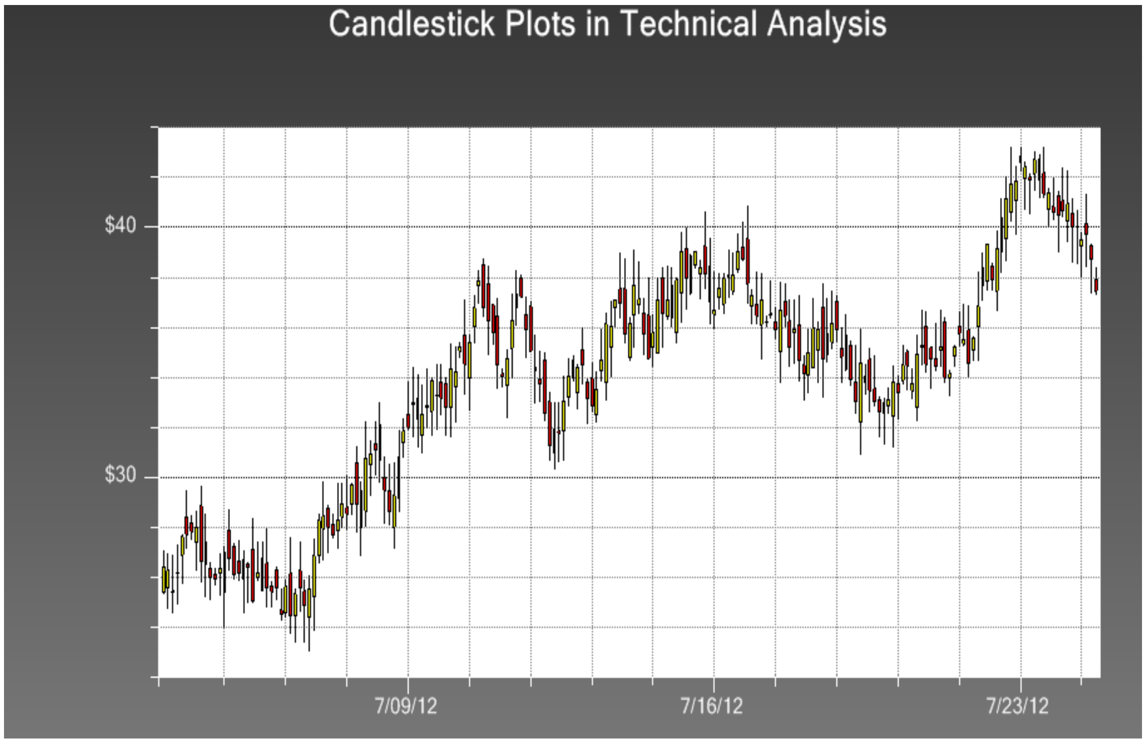
```
PhysicalCoordinates
    |
    +-- CartesianCoordinates
            |
            +-- EventCoordinates
```

The **EventCoordinates** class scales the chart plot area for a physical coordinate system which uses an event scale in combination with a linear or logarithmic scaling. The underlying event scale uses a simple linear scale, scaled from 0 to N-1, where N is the number of ChartEvents with a unique time stamp in the attached ChartEvent based datasets: EventSimpleDataset and EventGroupDataset. Unlike the other coordinate systems, an EventCoordinate object requires an event dataset as part of its definition.

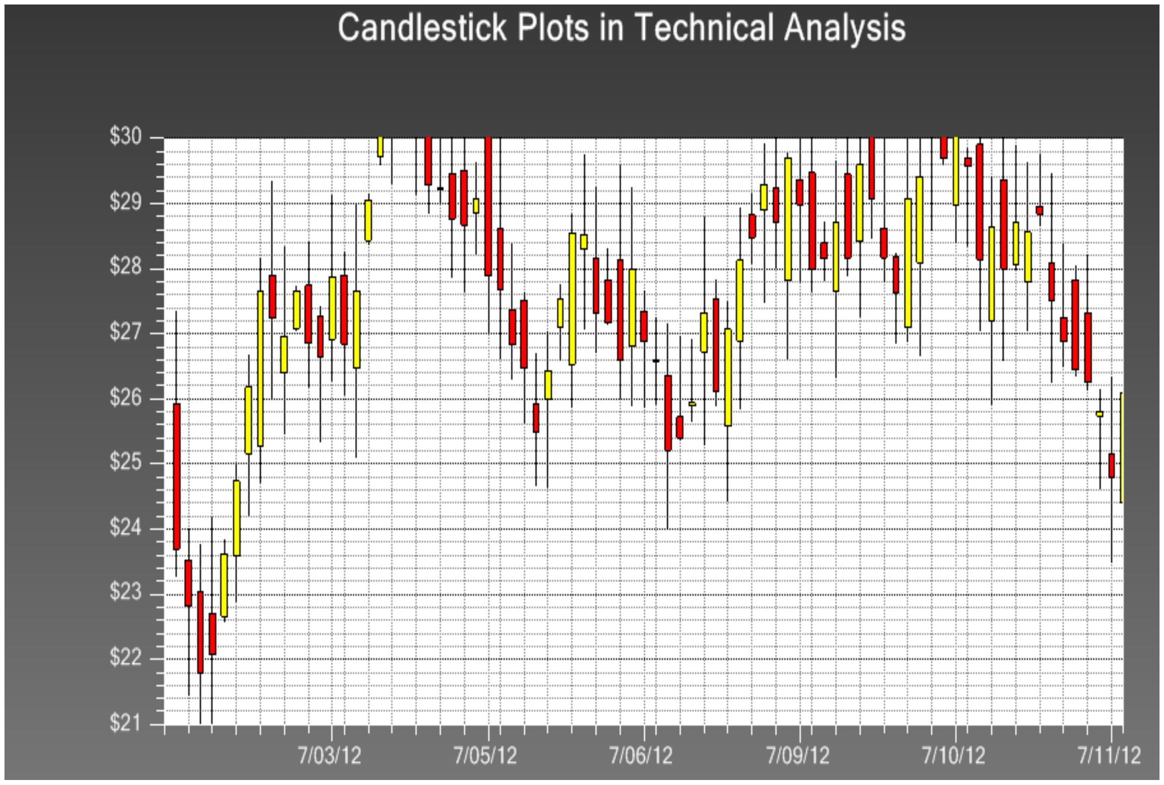
The **ChartEvent** class incorporates two x-value positioning properties, the Position and the TimeStamp, and one or more numeric y-values for each event. A single event therefore defines both the x-and y-values of the event in the underlying coordinate system. A collection, or array, of ChartEvent objects define the data for a plot, the same way as arrays of x- and y-values define a plot when using a simple dataset class with a Cartesian coordinate system. The critical element of the ChartEvent which permit it to be used for the plotting of discontinuous data is that the Position of the event in a chart is related, but, independent of the TimeStamp of the event. Event data can be positioned contiguously, and evenly spaced, in a chart, even if the time stamps of the events are not contiguous, or evenly spaced. Here is a simple example of a standard financial candlestick plot chart, using our TimeCoordinates class as the coordinate system, where the time/date data is not evenly spaced, and contains large gaps corresponding to weekends, and inactive hours of the day. The July 4th holiday is included in the range, and there is no data for that time interval either.



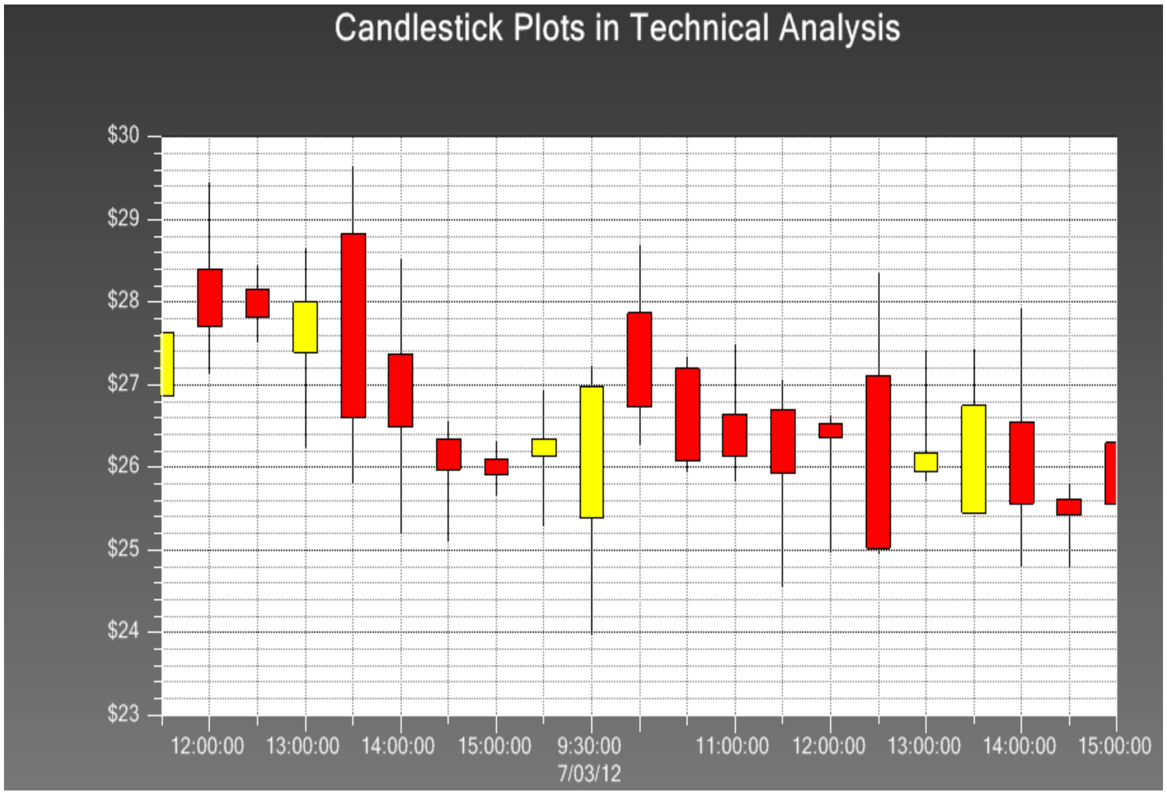
Contrast this to the similar data, using the same time range, plotted using the EventCoordinates class. Note how every event is evenly spaced with its neighbor. Gaps do not exist, since weekends, holidays, and unused hours are bridged over as if they do not exist.he same would be true for gaps due to holidays, and a varying number of work hours in a day.



Zooming in further, you can see the smooth transition across the July 4th holiday, and the following weekend.



Zoom in again, and you can see the smooth transition from one day to the next, even though the working hours are only a 9:30 to 16:00 subset of the 24 hours of a day.



This is accomplished because of the dual positioning values, Position and TimeStamp, of the ChartEvent class. Each element of a plot object (one of the candlestick objects in the plot above) is positioned in a simple linear coordinate system, starting at 0 and incrementing by 1 for each ChartEvent object. In the previous example, the first ChartEvent object is has a Position value of 0.0, and the last ChartEvent object has a position of 199, because there are 200 data points in the chart. This is what keeps the individual elements of a plot object evenly spaced, because the plot elements are positioned in the chart using the Position value, not the TimeStamp value. But, the associated x-axis (EventAxis) and x-axis labels objects (EventAxisLabels) look to the TimeStamp property for their values, not the Position property. What you end up with is the clean, evenly spaced look of a simple linear chart, with the axis tick marks and axis labeling of a dedicated time/date axis. The graph can be made to scroll (or pan) left to right, or re-scale along the y-axis, smoothly.

EventCoordinates constructors

```
public static newEventCoordinates(dataset: EventSimpleDataset): EventCoordinates

public static newEventCoordinatesMinMax(rX1: number, rY1: number, rX2: number, rY2: number):
EventCoordinates
```

```
public static newEventCoordinatesSimpleDataset(dataset: EventSimpleDataset): EventCoordinates

public static newEventCoordinatesSimpleDatasetSort(dataset: EventSimpleDataset, sort: boolean):
EventCoordinates

public static newEventCoordinatesSimpleDataetAxis(dataset: EventSimpleDataset, evaxis: number):
EventCoordinates

public static newEventCoordinatesSimpleDatasets(datasets: EventSimpleDataset[]): EventCoordinates

public static newEventCoordinatesSimpleDatasetsAxis(datasets: EventSimpleDataset[], evaxis:
number): EventCoordinates

public static newEventCoordinatesGroupDataset(dataset: EventGroupDataset): EventCoordinates

public static newEventCoordinatesGroupDatasetAxis(dataset: EventGroupDataset, evaxis: number):
EventCoordinates

public static newEventCoordinatesGroupDatasets(datasets: EventGroupDataset[]): EventCoordinates

public static newEventCoordinatesGroupDatasetsAxis(datasets: EventGroupDataset[], evaxis:
number): EventCoordinates
```

rX1	Sets the lower left x-value for the plotting area physical coordinate system.
rY1	Sets the lower left y-value for the plotting area physical coordinate system.
rX2	Sets the upper right x-value for the plotting area physical coordinate system.
rY2	Sets the upper right y-value for the plotting area physical coordinate system.
dataset	The dataset containing the coordinate system events.
sort	true and the dataset is sorted by time stamp.
evaxis	Specifies which axis (X_AXIS or Y_AXIS) is the event axis.
datasets	An array of datasets containing the coordinate system events.

Creating a chart using the event classes uses the same basic sequence as our other coordinate systems.

First, create the data – in this case an array of ChartEvent objects. Most of the code below is just the simulation of some raw data, taking into account a 9:30 to 16:00 working day, with no weekends. Note that each ChartEvent object is defined using both a TimeStamp value (xvalues[i]), and a Position value (i). The value of i controls the position of the ChartEvent object in the plot, and the value of the TimeStamp controls the x-axis tick marks and labels.

[TypeScript]

```
let nNumGroups: number = 4;
let nNumPnts: number = 50;
let weekmode: number = QCChartTS.ChartConstants.WEEK_5D;
let xValues: Date[] = new Array<Date>(nNumPnts);
let stockPriceData: number[] = new Array(nNumGroups);
let maxval: number = 0;
let minval: number = 0;
let i: number;
let currentdate: Date = new Date();
QCChartTS.ChartCalendar.setTOD(currentdate, 0, 0, 1);
theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
//  Make sure not to start on a weekend
xValues[0] = new Date(currentdate);
let eventArray: QCChartTS.ChartEvent[] = new Array<QCChartTS.ChartEvent>();
let currentEvent: QCChartTS.ChartEvent = new QCChartTS.ChartEvent();
```

```
currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
stockPriceData[3] = 25;
//  close
stockPriceData[0] = 25;
//  open
stockPriceData[1] = 26;
//  high
stockPriceData[2] = 24;
//  low
for (i = 0; (i < nNumPnts); i++) {
    let position: number = i + 1;
    xValues[i] = new Date(currentdate);
    if (i > 0) {
        stockPriceData[3] += 2 * (0.5 - QCChartTS.ChartSupport.getRandomDouble()); // close
        stockPriceData[0] += 2 * (0.5 - QCChartTS.ChartSupport.getRandomDouble()); // open
        minval = Math.min(stockPriceData[3], stockPriceData[0]);
        maxval = Math.max(stockPriceData[3], stockPriceData[0]);
        stockPriceData[1] = maxval + 1.5 * QCChartTS.ChartSupport.getRandomDouble();  // high
        stockPriceData[2] = minval - 1.5 * QCChartTS.ChartSupport.getRandomDouble();  // low
    }

    currentEvent = QCChartTS.ChartEvent.newChartEventDateTimeStampPosValues(xValues[i], position,
stockPriceData);
    currentEvent.AxisLabel = "XXX" + position.toString();
    currentEvent.ToolTip = "ToolTip" + position.toString();
    eventArray[i] = currentEvent;

    //  low
    currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
}
```

[JavaScript]

```
let nNumGroups = 4;
let nNumPnts = 50;
let weekmode = QCChartTS.ChartConstants.WEEK_5D;
let xValues = new Array(nNumPnts);
let stockPriceData = new Array(nNumGroups);
let maxval = 0;
let minval = 0;
let i;
let currentdate = new Date();
QCChartTS.ChartCalendar.setTOD(currentdate, 0, 0, 1);
theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
//  Make sure not to start on a weekend
xValues[0] = new Date(currentdate);
let eventArray = new Array();
let currentEvent = new QCChartTS.ChartEvent();
currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
stockPriceData[3] = 25;
//  close
stockPriceData[0] = 25;
//  open
stockPriceData[1] = 26;
//  high
stockPriceData[2] = 24;
//  low
for (i = 0; (i < nNumPnts); i++) {
    let position = i + 1;
    xValues[i] = new Date(currentdate);
    if (i > 0) {
        stockPriceData[3] += 2 * (0.5 - QCChartTS.ChartSupport.getRandomDouble()); // close
        stockPriceData[0] += 2 * (0.5 - QCChartTS.ChartSupport.getRandomDouble()); // open
        minval = Math.min(stockPriceData[3], stockPriceData[0]);
        maxval = Math.max(stockPriceData[3], stockPriceData[0]);
        stockPriceData[1] = maxval + 1.5 * QCChartTS.ChartSupport.getRandomDouble(); // high
        stockPriceData[2] = minval - 1.5 * QCChartTS.ChartSupport.getRandomDouble(); // low
    }
    currentEvent = QCChartTS.ChartEvent.newChartEventDateTimeStampPosValues(xValues[i], position,
stockPriceData);
    currentEvent.AxisLabel = "XXX" + position.toString();
    currentEvent.ToolTip = "ToolTip" + position.toString();
    eventArray[i] = currentEvent;
    //  low
    currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
}
```

Create an EventSimpleDataset, or EventGroupDataset using the source data.

[TypeScript]

```
let Dataset1: QCChartTS.EventGroupDataset =
QCChartTS.EventGroupDataset.newEventGroupDatasetNameArrayEventsGroups("Stock Data", eventArray,
4);
let Dataset2: QCChartTS.EventSimpleDataset = Dataset1.convertToEventSimpleDatasetIndex(1);
```

[JavaScript]

```
let Dataset1 = QCChartTS.EventGroupDataset.newEventGroupDatasetNameArrayEventsGroups("Stock
Data", eventArray, 4);
let Dataset2 = Dataset1.convertToEventSimpleDatasetIndex(1);
```

Create an EventCoordinateSystem, referencing the EventSimpleDataset as a parameter. The coordinate system is defined by the content of the EventSimpleDataset. It will auto-scale the coordinate system to the number of ChartEvents found in the source dataset.

[TypeScript]

```
let pTransform1: QCChartTS.EventCoordinates =
QCChartTS.EventCoordinates.newEventCoordinatesGroupDataset(Dataset1);

pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
pTransform1.setGraphBorderDiagonal(0.13, .15, .90, 0.8);
```

[JavaScript]

```
let pTransform1 = QCChartTS.EventCoordinates.newEventCoordinatesGroupDataset(Dataset1);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
pTransform1.setGraphBorderDiagonal(0.13, .15, .90, 0.8);
```

Define the x-axis as an EventAxis. The tick marks of the axis are defined using the TickRule property, in this case the TickRule is MINORCROSSOVEREVENT_MAJORCROSSOVEREVENT. This means a minor tick mark is placed every time the time rolls over a minor event, and a major tick mark is placed every time the time rolls over a major event. More on this later.

[TypeScript]

```
let xAxis1: QCChartTS.EventAxis = QCChartTS.EventAxis.newEventAxisCoords(pTransform1);
xAxis1.setColor(QCChartTS.ChartColor.WHITE);
xAxis1.TickRule = QCChartTS.ChartConstants.MINORCROSSOVEREVENT_MAJORCROSSOVEREVENT;

chartVu.addChartObject(xAxis1);
```

[JavaScript]

```
let xAxis1 = QCChartTS.EventAxis.newEventAxisCoords(pTransform1);
xAxis1.setColor(QCChartTS.ChartColor.WHITE);
xAxis1.TickRule = QCChartTS.ChartConstants.MINORCROSSOVEREVENT_MAJORCROSSOVEREVENT;

chartVu.addChartObject(xAxis1);
```

Last, define the x-axis labels using the EventAxisLabels class.

[TypeScript]

```
let xAxisLab1: QCChartTS.EventAxisLabels = QCChartTS.EventAxisLabels.newEventAxisLabels(xAxis1);
xAxisLab1.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(xAxisLab1);
```

[JavaScript]

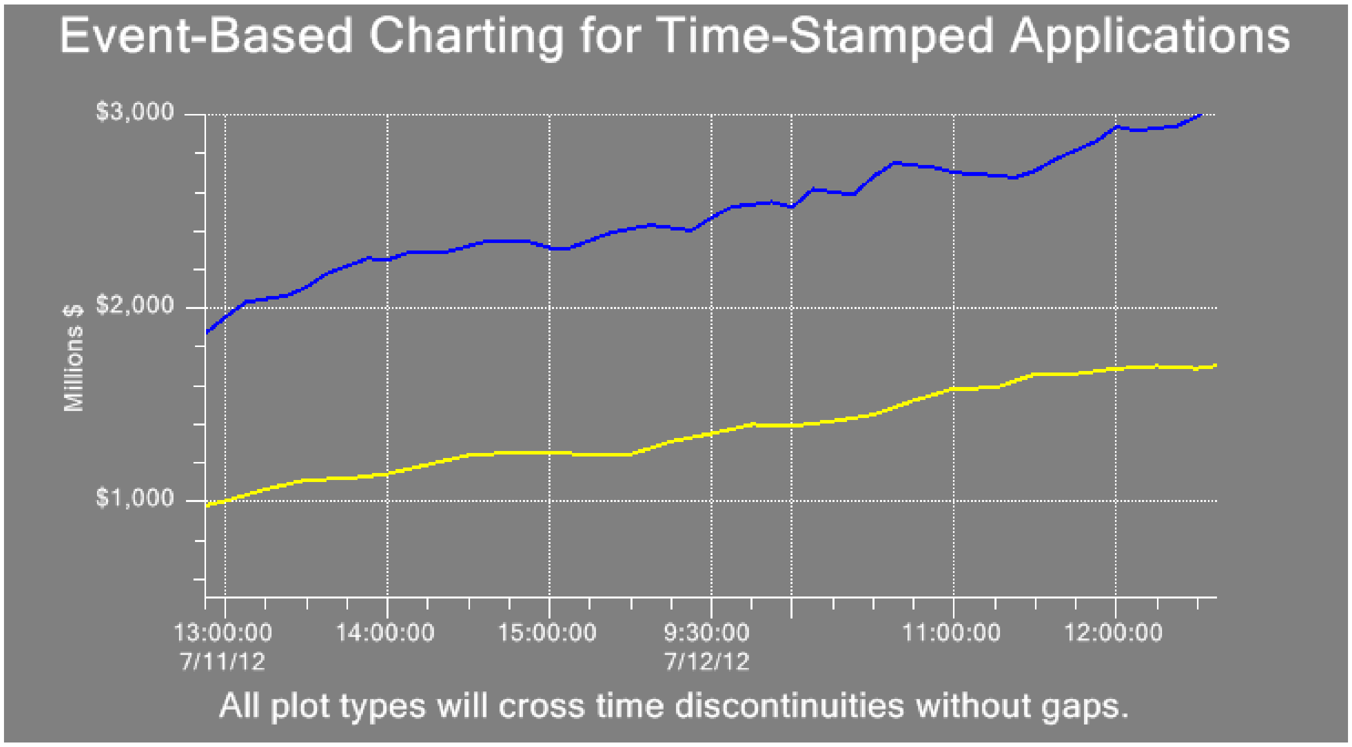
```
let xAxisLab1 = QCChartTS.EventAxisLabels.newEventAxisLabels(xAxis1);
xAxisLab1.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(xAxisLab1);
```

Everything else is the same as in our other charts.

Things are more complicated once you start plotting multiple, overlapping, datasets in the same coordinate system. Each ChartEvent object in the previous example had a unique Position and TimeStamp value. The software only needed to auto-scale the coordinate system for the range of ChartEvent Position values, and plot the data. If you want to plot a second dataset in the same chart, it is more complicated. If every ChartEvent object of the second dataset uses exactly the same Position and TimeStamp values as the first, and only has different y-values, you could plot it as is. You should be able to plot the second dataset directly on top of the first, and the TimeStamps of the ChartEvents objects would line up chronologically exactly as you would expect. The complication arises if the second dataset overlaps the first with respect to the TimeStamp values, but does NOT use exactly the same TimeStamps as the first set of data. For example, the first set of data is contains ChartEvent objects sampled at 10 minute intervals, starting at 8:30. The second dataset contains ChartEvents objects starting at 8:15 and sampled every 15 minutes. In this case some of the TimeStamp values would match (at every ½ hour), but in every case, the Position values would be out of sync. If these two datasets were plotted as is, the TimeStamp values for the two datasets would not align with respect to the x-axis.

When multiple datasets are attached to the same EventCoordinate system, the coordinate system needs to merge and sort the ChartEvent objects of every dataset. The ChartEvents are sorted by the TimeStamp value. It may be that many TimeStamps are duplicates, since different datasets may have used the same

TimeStamps in their event data. Other TimeStamps may be singular, having occurred in only one dataset. It really doesn't matter. Next the software runs through the ChartEvents of the sorted list, assigning a new Position value to every object in the list. If adjacent ChartEvent objects in the sorted list have the same TimeStamp, they are assigned the same Position value. If the next object in the sorted list has a different TimeStamp value, it is assigned a Position value one greater than the previous ChartEvent. The net effect is that the ChartEvents with the same TimeStamp value will align properly at the same x-position in the graph, regardless of how the initial Position values were set. The EventAxis and EventAxisLabels objects, when applied to the EventCoordinate system, will see every unique TimeStamp in the merged datasets, without double counting duplicate TimeStamp values across dataset.



Note how the plot still captures the crossover between 16:00 7/12/12 and 9:30 7/13/12 without a gap. The same would be true if there was a weekend, or holiday, or even a lunch break, between adjacent data points.

Below is an example of how multiple datasets are attached to an EventCoordinates system.

[TypeScript]

```
let Dataset1: QCChartTS.EventSimpleDataset =
QCChartTS.EventSimpleDataset.newEventSimpleDatasetNameArrayEvents("Actual Sales", cev1);
let Dataset2: QCChartTS.EventSimpleDataset =
QCChartTS.EventSimpleDataset.newEventSimpleDatasetNameArrayEvents("Forecast Sales", cev2);
let Dataset3: QCChartTS.EventSimpleDataset =
QCChartTS.EventSimpleDataset.newEventSimpleDatasetNameArrayEvents("Actual Sales", cev3);
let Dataset4: QCChartTS.EventSimpleDataset =
QCChartTS.EventSimpleDataset.newEventSimpleDatasetNameArrayEvents("Forecast Sales", cev4);
let DatasetArray: QCChartTS.EventSimpleDataset[] = [
    Dataset1,
    Dataset2,
    Dataset3,
    Dataset4];
let pTransform1: QCChartTS.EventCoordinates =
QCChartTS.EventCoordinates.newEventCoordinatesSimpleDatasets(DatasetArray);
```

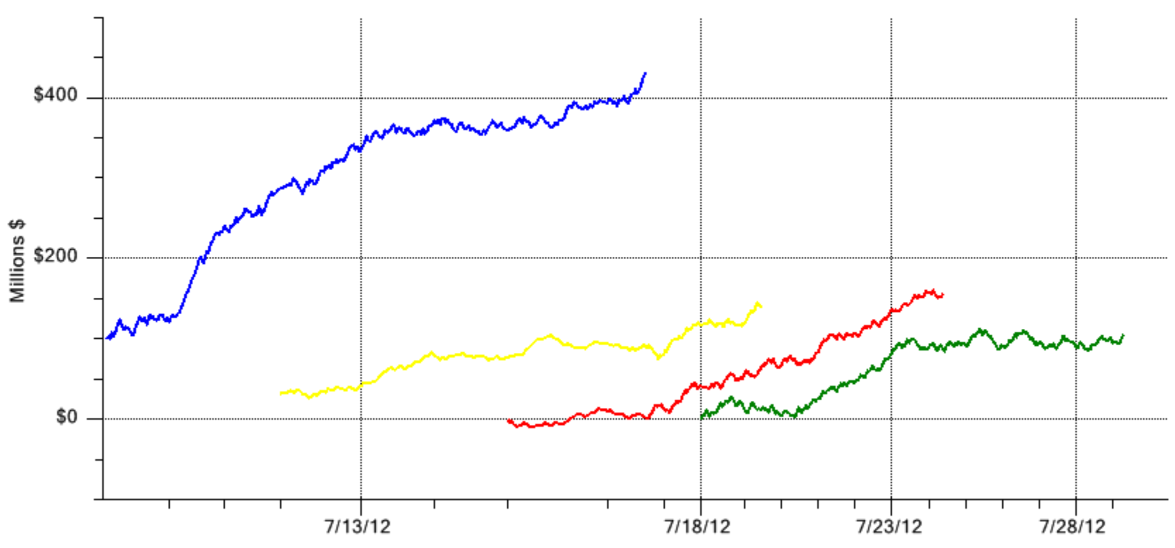
[JavaScript]

```
let Dataset1 = QCChartTS.EventSimpleDataset.newEventSimpleDatasetNameArrayEvents("Actual Sales",
cev1);
let Dataset2 = QCChartTS.EventSimpleDataset.newEventSimpleDatasetNameArrayEvents("Forecast
Sales", cev2);
let Dataset3 = QCChartTS.EventSimpleDataset.newEventSimpleDatasetNameArrayEvents("Actual Sales",
cev3);
let Dataset4 = QCChartTS.EventSimpleDataset.newEventSimpleDatasetNameArrayEvents("Forecast
Sales", cev4);
let DatasetArray = [
    Dataset1,
    Dataset2,
    Dataset3,
    Dataset4
];
```

This method relies on the ability to detect when the time stamps of an event are equal. In the case of pure time, equality depends on the granularity you want in the display. Events can be equal at the year, month, week, day, hour, minute second and millisecond level. So, if you want events, across multiple plots, to line up by the minute, not caring for differences in seconds or milliseconds, you can do that. Set the EventCoordinates property TimeStampResolution to QCChartTS.ChartConstants.MINUTE. The default value is QCChartTS.ChartConstants.SECOND, and if you want you can set the resolution to MILLISECOND, SECOND, MINUTE, HOUR, DAY_OF_YEAR, WEEK_OF_YEAR, MONTH, or YEAR. Make sure you set the resolution to a value below that what you want to see in your data. If your data is sampled at 6 second intervals, and you want to see each value at a unique position on the x-axis, set the TimeStampResolution to SECOND. If you set it to MINUTE, all of the samples within a minute interval will be grouped together at a single Position value.

Below is an example of a chart with multiple, overlapping datasets, sampled at different resolutions.

Event-Based Charting for Time-Stamped Applications



All plot types will cross time discontinuities without gaps.

When the merged datasets have the Position values of their ChartEvent objects modified to reflect the true position of the ChartEvent in the graph, the auto-positioning starts at 1 (not 0). If 0 was used, the first data point would exactly on the edge of the clipping window, and this would cut off half of a bar, scatter plot, candlestick, or OHLC symbol positioned in the first position. The default auto-scaling values scale the ChartCoordinates scale from 0 to N+1, where N is then number of unique ChartEvent Positon values. This way ChartEvent objects are not cut off on the left or the right.

There is a specialized axis class, EventAxis, and axis labels class, EventAxisLabels, to use with event data. Unlike our regular LinearAxis, and TimeAxis classes, which are independent of the data in the chart, the EventAxis is dependent on the underlying data. The tick marks of the EventAxis are placed at the x-position of the associated ChartEvent objects. If there are more than 10-20 ChartEvent objects in the graph, the tick mark labels would start to overlap, so we divide the tick marks into major tick marks, which are those that get a label, and minor tick marks which do not get a label. Further, if there are more than 100-200 ChartEvent objects in the graph, then the tick marks may start to overlap. So the software has the option of drawing a minor, or a major tick mark, every Nth event, to keep them from overlapping. Most of this is taken care of by the auto-axis routines. Though it is possible you do not like the results – you can't please everybody. So, once the axis is created you can modify the appearance by adjusting the following properties.

Regardless of how you initially setup the graph, if you use the zoom routines, or the RTScrollFrame routines in the QCRTGraph software, the look of the axes will always revert to our auto-scaling, not your modified setup. Because we cannot predict what you will do, and cannot scale a completely different range of values based on whatever logic you use.

TickRule – Controls the tick mark logic of the axis. User one of the TickRule constants found in the ChartConstants class:

NO_TICKS – do not display any tick marks. No tick marks means no axis labels.

MINOREVENT_MAJOREVENT – display a minor tick mark every AxisMinorNthTick event, and a major tick mark every AxisMinorTicksPerMajor.

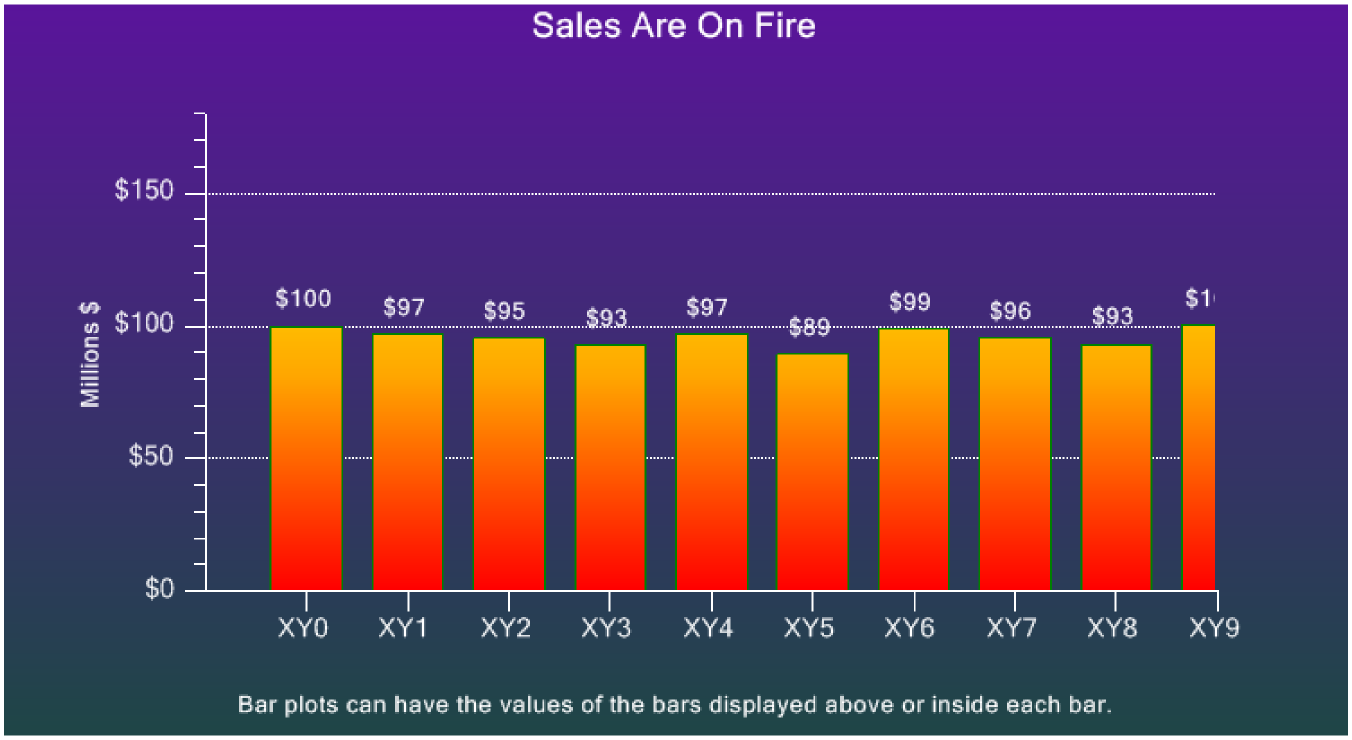
MAJOREVENT - display a major tick mark every AxisMajorNthTick event.

MINORCROSSOVEREVENT_MAJORCROSSOVEREVENT – display a minor tick mark every AxisMinorNthTick, minor crossover event, and a major tick mark every AxisMajorNthTick major crossover event. The minor and major crossover events are controlled by the MajorTickCrossoverEvent and MinorTickCrossoverEvent properties of the EventAxis.

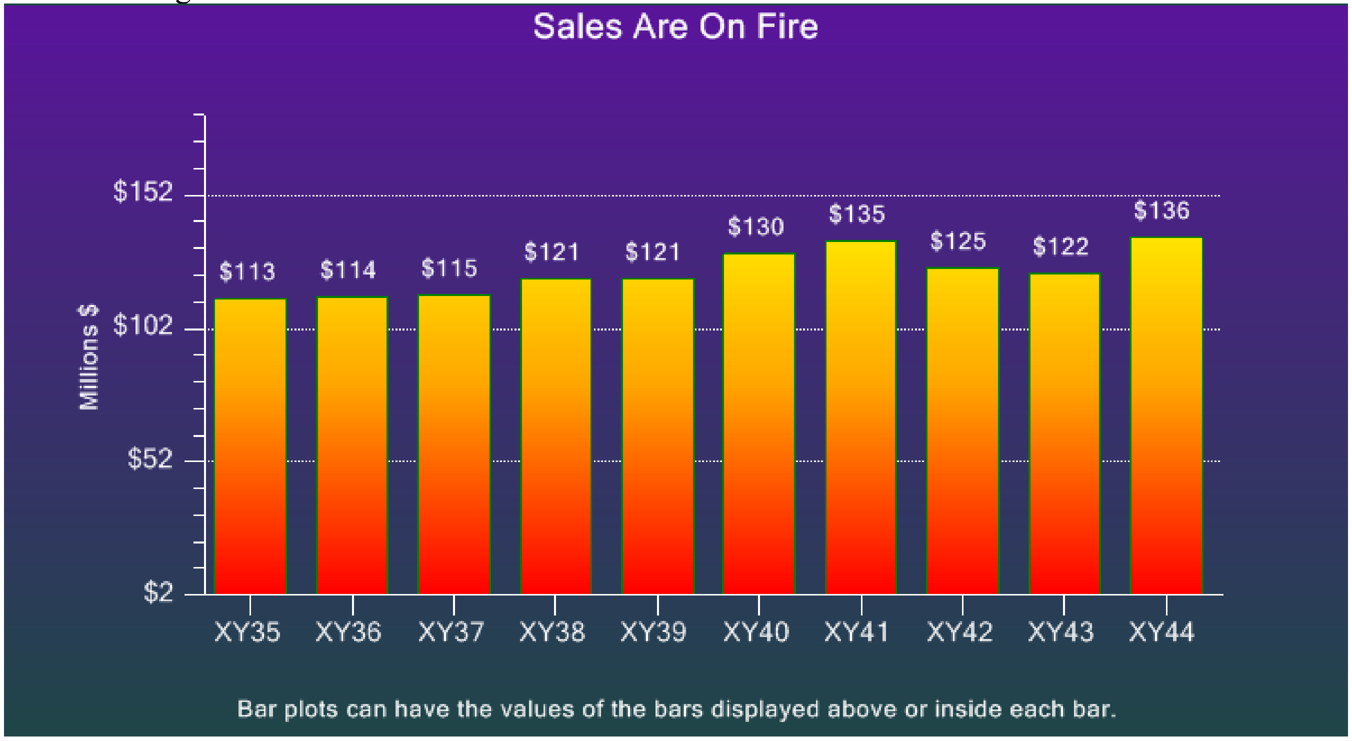
The default is QCChartTS.ChartConstants.MINORCROSSOVEREVENT_MAJORCROSSOVEREVENT.

The term crossover event means that a field of date/time timestamp changes. If you specify a MinorTickCrossoverEvent of QCChartTS.ChartConstants.SECOND, and an AxisMinorNthTick of 15, this will cause a minor tick mark to be displayed every 15th second, if an event falls within that range. So, if your events are spaced approximately 5 seconds apart, you will get a minor tick mark for approximately every three events. If you choose a MajorTickCrossoverEvent of QCChartTS.ChartConstants.MINUTE and an AxisMajorNthTick of 1, this will cause a major tick mark to be displayed every minute, if an event falls within that range. Tick marks only show up on an event, so if there are no events within the time interval, no tick mark will appear.

Every tick mark can have a custom label. So if you do not want to use the default time/date labels, but instead want label the event tick marks with a custom string, you can do that. The string will track the exact tick mark associated with a given event. This makes scrolling through the data easier, because the custom tick mark strings stick to the tick mark, and don't have to be recalculated.



After scrolling the data to the left:



If you plan to implement scrolling (panning) along the x-axis, using a scroll bar, or some other method, you need to know how to re-scale the x-scale EventCoordinate system. First, understand that the underlying coordinate system is a Cartesian coordinate system, with the x-axis scaled from 0 to the number of ChartEvent objects with unique time stamps. Because a simple linear scale is used for the x-axis, you can scale the x-axis using simple linear values (0 to number of ChartEvent objects). In that case you use the EventCoordinates.ScaleStartX and EventCoordinates.ScaleStopX properties. The code below is extracted from the **ChartEventExamples.BuildOHLCEvent** example program.

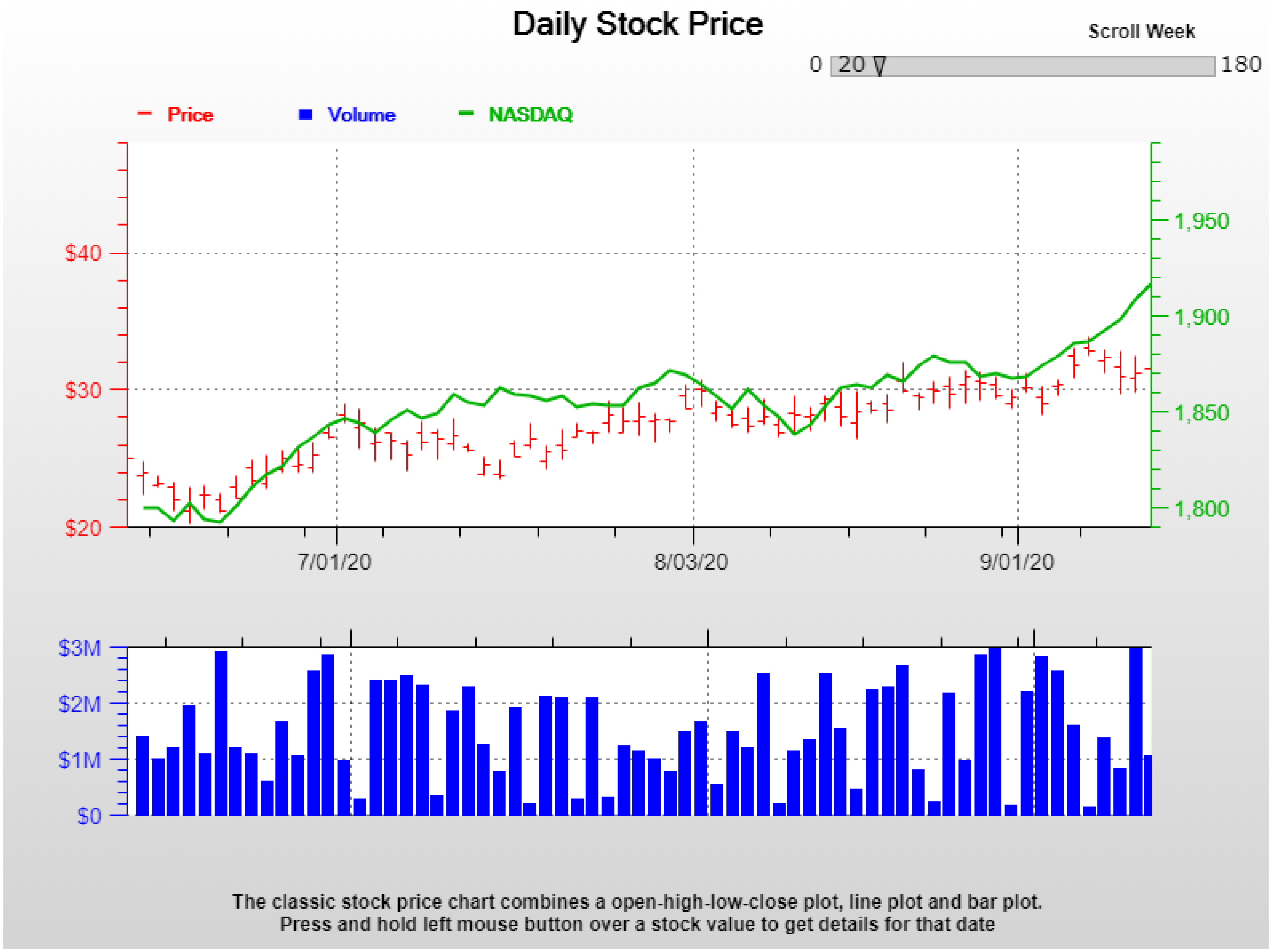
[TypeScript]

```
// in this example,all of the object with this in front are global to the class

public  UpdateScaleAndAxesUsingEventIndex( startindex: number)
 {

     this.pTransform1.ScaleStartX = startindex;
     this.pTransform1.ScaleStopX = startindex + numberEventsInView - 1;

     this.pTransform2.ScaleStartX = startindex;
     this.pTransform2.ScaleStopX = startindex + numberEventsInView - 1;

     this.pTransform3.ScaleStartX = startindex;
     this.pTransform3.ScaleStopX = startindex + numberEventsInView - 1;


     this.xAxis1.calcAutoAxis();
     this.yAxis1.calcAutoAxis();
     this.xAxisLab1.calcAutoAxisLabels();
     this.yAxisLab1.calcAutoAxisLabels();

     this.xAxis2.calcAutoAxis();
     this.xAxis2.setAxisIntercept(pTransform2.getStopY());
     this.xAxis2.setAxisTickDir(QCChartTS.ChartConstants.AXIS_MAX);

     this.yAxis2.calcAutoAxis();
     this.yAxisLab2.calcAutoAxisLabels();

     this.yAxis3.calcAutoAxis();
     this.yAxisLab3.calcAutoAxisLabels();
     this.yAxis3.setAxisIntercept(pTransform3.getStopX());
     this.yAxis3.setAxisTickDir(QCChartTS.ChartConstants.AXIS_MAX);

     this.chartVu.updateDraw();
 }
```

[JavaScript]

```
// in this example,all of the objects  are global to the module

UpdateScaleAndAxesUsingEventIndex( startindex )
{

    pTransform1.ScaleStartX = startindex;
     pTransform1.ScaleStopX = startindex + numberEventsInView - 1;

     pTransform2.ScaleStartX = startindex;
     pTransform2.ScaleStopX = startindex + numberEventsInView - 1;

     pTransform3.ScaleStartX = startindex;
     pTransform3.ScaleStopX = startindex + numberEventsInView - 1;


     xAxis1.calcAutoAxis();
```

```
        yAxis1.calcAutoAxis();
        xAxisLab1.calcAutoAxisLabels();
        yAxisLab1.calcAutoAxisLabels();

        xAxis2.calcAutoAxis();
        xAxis2.setAxisIntercept(pTransform2.getStopY());
        xAxis2.setAxisTickDir(QCChartTS.ChartConstants.AXIS_MAX);

        yAxis2.calcAutoAxis();
        yAxisLab2.calcAutoAxisLabels();

        yAxis3.calcAutoAxis();
        yAxisLab3.calcAutoAxisLabels();
        yAxis3.setAxisIntercept(pTransform3.getStopX());
        yAxis3.setAxisTickDir(QCChartTS.ChartConstants.AXIS_MAX);

        chartVu.updateDraw();

    }
```



It may be that you want to specify date/times for the starting and ending values of the x-axis, instead of a simple index. In that case you use the EventCoordinates.TimeScaleStart and EventCoordinates.TimeScaleStop properties. Those methods use a binary search algorithm to search for the ChartEvent closest to the desired date.time. It then uses the ChartEvent at that index to establish the x-axis scale. The code below is extracted from the **OHLCChart.BuildOHLCChart** example program.

[TypeScript]

```
public  UpdateScaleAndAxesUsingDates( startindex: number)
{
     let startdate:  Date   = new Date(datastartdate);
     let stopdate:  Date    = new Date();

     QCChartTS.ChartCalendar.add(startdate, QCChartTS.ChartConstants.DAY_OF_YEAR,startindex);
     stopdate = new Date(startdate);
     QCChartTS.ChartCalendar.add(stopdate, QCChartTS.ChartConstants.MONTH,1);

     this.pTransform1.TimeScaleStart = startdate;
     this.pTransform1.TimeScaleStop = stopdate;

     this.pTransform2.TimeScaleStart = startdate;
     this.pTransform2.TimeScaleStop = stopdate;

     this.pTransform3.TimeScaleStart = startdate;
     this.pTransform3.TimeScaleStop = stopdate;

     this.xAxis1.calcAutoAxis();
     this.yAxis1.calcAutoAxis();
     this.xAxisLab1.calcAutoAxisLabels();
     this.yAxisLab1.calcAutoAxisLabels();

     this.xAxis2.calcAutoAxis();
     this.xAxis2.setAxisIntercept(this.pTransform2.getStopY());
     this.xAxis2.setAxisTickDir(QCChartTS.ChartConstants.AXIS_MAX);

     this.yAxis2.calcAutoAxis();
     this.yAxisLab2.calcAutoAxisLabels();

     this.yAxis3.calcAutoAxis();
     this.yAxisLab3.calcAutoAxisLabels();
     this.yAxis3.setAxisIntercept(this.pTransform3.getStopX());
     this.yAxis3.setAxisTickDir(QCChartTS.ChartConstants.AXIS_MAX);

     this.chartVu.updateDraw();
}
```

[JavaScript]

```
UpdateScaleAndAxesUsingDates( startindex )
     let startdate  = new Date(datastartdate);
     let stopdate   = new Date();

     QCChartTS.ChartCalendar.add(startdate, QCChartTS.ChartConstants.DAY_OF_YEAR,startindex);
     stopdate = new Date(startdate);
     QCChartTS.ChartCalendar.add(stopdate, QCChartTS.ChartConstants.MONTH,1);

     pTransform1.TimeScaleStart = startdate;
     pTransform1.TimeScaleStop = stopdate;

     pTransform2.TimeScaleStart = startdate;
     pTransform2.TimeScaleStop = stopdate;

     pTransform3.TimeScaleStart = startdate;
     pTransform3.TimeScaleStop = stopdate;

     xAxis1.calcAutoAxis();
     yAxis1.calcAutoAxis();
     xAxisLab1.calcAutoAxisLabels();
     yAxisLab1.calcAutoAxisLabels();
```

```
    xAxis2.calcAutoAxis();
    xAxis2.setAxisIntercept(pTransform2.getStopY());
    xAxis2.setAxisTickDir(QCChartTS.ChartConstants.AXIS_MAX);

    yAxis2.calcAutoAxis();
    yAxisLab2.calcAutoAxisLabels();

    yAxis3.calcAutoAxis();
    yAxisLab3.calcAutoAxisLabels();
    yAxis3.setAxisIntercept(pTransform3.getStopX());
    yAxis3.setAxisTickDir(QCChartTS.ChartConstants.AXIS_MAX);

    chartVu.updateDraw();
}
```

If you specify a date/time value which is an exact match for one of the date/values of a ChartEvent, then the ChartEvent Position property becomes the scale value. If it is not an exact match, then the binary search for the closest date/time rounds down to the nearest ChartEvent for the TimeScaleStart property, and rounds up for the TimeScaleStop property.

When using the time/date values for scaling an EventCoordinate system, you cannot set a time/date value not bounded by the range of values found in the attached datasets. In other words, you cannot create datasets using ChartEvents that use time/date values in 2011, and try and scale the x-axis for a range of 2011 to 2013. The largest value you can scale the x-axis for is the time/date value of the ChartEvent with the largest (or latest) time stamp value. The only time/date values which exist in a EventCoordinates based coordinate system are the time stamps of the ChartEvents in the datasets attached to the coordinate systems. Other times and dates do not exist unless you add a ChartEvent containing the date/time to one of the attached datasets.

Polar Coordinate Systems

Class PolarCoordinates

```
PhysicalCoordinates
        |
        +-- CartesianCoordinates
                |
                +-- PolarCoordinates
```

The magnitude and the polar angle of a point define its position in a chart scaled for polar coordinates. The magnitude can have any value greater than 0.0 and the polar angle any positive or negative value. A polar angle range of 0 to 2 pi radians (0 to 360 degrees) sweeps a complete circle in polar coordinates.

A polar coordinate system uses a Cartesian coordinate system scaled for plus/minus the polar magnitude. The following equations convert from polar coordinates to Cartesian coordinates.

x	=	magnitude * cos (angle)
y	=	magnitude * sine (angle)

magnitude	Polar coordinate magnitude
angle	Plot coordinate angle
x	Cartesian x-coordinate
y	Cartesian y-coordinate

The **PolarCoordinates** class is an extension of the **CartesianCoordinates** class and it automatically handles these conversions. The only important parameter that needed for the creation of a **PolarCoordinates** object is the polar magnitude, since the polar angle always has a range 0 to 2 pi radians (0 to 360 degrees).

PolarCoordinates constructors

```
public static newPolarCoordinatesRadius(rR: number): PolarCoordinates

public static newPolarCoordinates(rR: number): PolarCoordinates

public static newPolarCoordinatesDataset(dataset:ChartDataset): PolarCoordinates

public static newPolarCoordinatesDatasets(datasets:ChartDataset[]): PolarCoordinates
```

rR	The radius of the polar scale.
dataset	The dataset of polar chart data.
datasets	An array of datasets of polar chart data.

The first way to create a **PolarCoordinates** object is to use the constructor that specifies the polar magnitude directly.

[TypeScript]

```
let polarmagnitude: number = 5.0;
let pPolarTransform: QCChartTS.PolarCoordinates =
QCChartTS.PolarCoordinates.newPolarCoordinatesRadius(polarmagnitude);
```

[JavaScript]

```
let polarmagnitude= 5.0;
let pPolarTransform =  QCChartTS.PolarCoordinates.newPolarCoordinatesRadius(polarmagnitude);
```

Or you can use an auto-scale routine to analyze a dataset and select the appropriate polar magnitude.

[TypeScript]

```
let angleData: number [] = [.20,.60,1.40,1.70,2.50,4.0,5.0, 6.0];   // In Radians
let magnitudeData [] = [ 20, 33, 44, 55, 46, 33, 54, 64];
let Dataset1: QCChartTS.SimpleDataset = QCChartTS.SimpleDataset.newSimpleDataset("First",
magnitudeData, angleData);
let pPolarTransform: QCChartTS.PolarCoordinates =
QCChartTS.PolarCoordinates.newPolarCoordinatesDataset(Dataset1);
```

[JavaScript]

```
let angleData = [.20,.60,1.40,1.70,2.50,4.0,5.0, 6.0];   // In Radians
let magnitudeData = [ 20, 33, 44, 55, 46, 33, 54, 64];
let Dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("First", magnitudeData, angleData);
let pPolarTransform =    QCChartTS.PolarCoordinates.newPolarCoordinatesDataset(Dataset1);
```

Antenna Coordinate Systems

Class AntennaCoordinates

```
PhysicalCoordinates
   |
   +-- CartesianCoordinates
          |
          +-- AntennaCoordinates
```

An antenna coordinate’s point is defined by its radial and angular values. The radial and angle values can be positive or negative. An angle range of 0 to 360 degrees clockwise, starting at 12:00, sweeps a complete circle in antenna coordinates.

AntennaCoordinates are defined by specifying a starting and ending value for the radius. Unlike a polar chart, these values can be positive or negative. Antenna coordinates always have an angular range 0 to 360 degrees.

AntennaCoordinates constructors

```
public static newAntennaCoordinatesMinMax(minvalue: number, maxvalue: number): AntennaCoordinates

public static newAntennaCoordinates(minvalue: number, maxvalue: number): AntennaCoordinates

public static newAntennaCoordinatesDataset(dataset: ChartDataset): AntennaCoordinates
public static newAntennaCoordinatesDatasets(datasets: ChartDataset[]): AntennaCoordinates
```

minvalue	The minimum value of the radius of the antenna scale.
maxvalue	The maximum value of the radius of the antenna scale.
dataset	The dataset used as the basis for the new coordinate system. The maximum x-value in the dataset controls the antenna coordinate scaling.
datasets	An array of datasets used as the basis for the new coordinate system. The maximum x-value in the datasets controls the antenna coordinate scaling.

The first way to create a **AntennaCoordinates** object is to use the constructor that specifies the minimum and maximum values of the radius: **AntennaCoordinates(minvalue, maxvalue).**

[TypeScript]

```
let minvalue: number  = -40;
let maxvalue: number  = 20;

let antennacoords: QCChartTS.AntennaCoordinates  =
QCChartTS.AntennaCoordinates.newAntennaCoordinates (minvalue, maxvalue);
```

[JavaScript]

```
let minvalue  = -40;
let maxvalue  = 20;

let antennacoords = QCChartTS.AntennaCoordinates.newAntennaCoordinates (minvalue, maxvalue);
```

Or you can use an auto-scale routine to analyze a dataset and select the appropriate antenna radius limits.

[TypeScript]

```
let angleData: number [] = [0, 30, 60, 90, 120, 150, 180];   // In degrees
let radiusData: number[]  = [ -35, -31, -5, 12, 14, -14, -30];
let dataset: QCChartTS.SimpleDataset  = QCChartTS.SimpleDataset.newSimpleDataset("Control",
angleData, radiusData);
let antennacoords: QCChartTS.AntennaCoordinates  = new  QCChartTS.AntennaCoordinates ();
antennacoords.autoScaleDatasetRoundMode(dataset, QCChartTS.ChartConstants.AUTOAXES_FAR); ;
```

[JavaScript]

```
let angleData = [0, 30, 60, 90, 120, 150, 180];   // In degrees
let radiusData  = [ -35, -31, -5, 12, 14, -14, -30];
let dataset: QCChartTS.SimpleDataset  = QCChartTS.SimpleDataset.newSimpleDataset("Control",
angleData, radiusData);
let antennacoords  = new  QCChartTS.AntennaCoordinates ();
```

```
antennacoords.autoScaleDatasetRoundMode(dataset, QCChartTS.ChartConstants.AUTOAXES_FAR);
```

Miscellaneous Coordinate System Topics

Inverted Coordinate Systems

Charts that use linear, logarithmic and time coordinate systems usually follow the convention that values increase as you move from left to right and from bottom to top. This is not always the case though. Many standard charts that users want to reproduce on the computer have the x-scale, the y-scale, or both, increase as you move from right to left and from top to bottom.

Invert the x- and/or y-scales by swapping the scale starting and ending vaues in the call to the **CartesianCoordinates** or the **TimeCoordinates** constructor.

Example of inverted x-scale using the CartesianCoordinates constructor

[TypeScript]

```
let xMin: number  = -5;
let xMax: number  = 15;
let yMin: number  = 0;
let yMax: number  =  15;

let simpleScale: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinates(xMax, yMin, xMin, yMax);
```

[JavaScript]

```
let xMin  = -5;
let xMax  = 15;
let yMin  = 0;
let yMax =  15;

let simpleScale = QCChartTS.CartesianCoordinates.newCartesianCoordinates(xMax, yMin, xMin, yMax);
```

Use the **CartesianCoordinates.setCoordinateBounds** method in the same manner. The example below inverts the y-scale.

```
simpleScale.setCoordinateBounds(xMin, yMax, xMax, yMin)
```

Invert the x- and y-scale of a **TimeCoordinates** object in an analogous fashion.

Example of inverted scaling using a TimeCoordinates constructor

[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  =  15;
let simpleTimeScale :  QCChartTS.TimeCoordinates  =
QCChartTS.TimeCoordinates.newTimeCoordinates(xMax, yMin, xMin, yMax);;
```

[JavaScript]

```
let xMin = new Date(1996, QCChartTS.ChartConstants.FEBRUARY,  5);
let xMax = new Date(2002, QCChartTS.ChartConstants.JANUARY,  5);
let yMin  = 0;
let yMax  =  15;

let simpleTimeScale  = QCChartTS.TimeCoordinates.newTimeCoordinates(xMax, yMin, xMin, yMax);
```

Use the **TimeCoordinates.setCoordinateBounds** method in the same manner. The example below inverts the y-scale.

[TypeScript]

```
let simpleTimeScale: QCChartTS.TimeCoordinates  = new QCChartTS.TimeCoordinates();
simpleTimeScale.setCoordinateBounds(xMin, yMax, xMax, yMin);
```

[JavaScript]

```
let simpleTimeScale  = new QCChartTS.TimeCoordinates();
simpleTimeScale.setCoordinateBounds(xMin, yMax, xMax, yMin);
```

The auto-scale functions always create scales that increase from left to right, and bottom to top. This does not exclude the use of the auto-scale functions when creating inverted axes. After an auto-scale function creates the initial x- and y-scales, either or both can be inverted by using the **CartesianCoordinates** or **TimeCoordinates invertScaleX** or **invertScaleY** methods.

Example of inverting a scale created using the auto-scale methods

[TypeScript]

```
let xData: number[] = [2,3,4,5,6,7,8,9];
let yData: number[] = [ 22, 33, 44, 55, 46, 33, 25, 14];

let dataset:  QCChartTS.SimpleDataset  = QCChartTS.SimpleDataset.newSimpleDataset("Sales", xData,
yData);
let simpleScale:  QCChartTS.CartesianCoordinates   = new QCChartTS.CartesianCoordinates();
simpleScale.autoScaleDataset(dataset);
simpleScale.invertScaleY();
```

[JavaScript]

```
let xData = [2,3,4,5,6,7,8,9];
let yData = [ 22, 33, 44, 55, 46, 33, 25, 14];

let dataset  = QCChartTS.SimpleDataset.newSimpleDataset("Sales", xData, yData);
let simpleScale = new QCChartTS.CartesianCoordinates();
simpleScale.autoScaleDataset(dataset);
simpleScale.invertScaleY();
```

5. The Chart View

ChartView

The starting point of a chart is the **ChartView** class. The **ChartView** class contains a collection of all the chart objects displayed in the chart and will automatically update all chart objects when the underlying window moves, resizes, or otherwise needs to redraw in response to a **redraw** event.

```
UserControl
    |
    +-- ChartView
```

The **ChartView** class has only one constructor with no arguments.

ChartView constructor

```
public static newChartView(canvas: Canvas)
```

canvas	HTML canvas to use for the chart.

Inside the software we define the Canvas type to be an HTMLCanvasElement

All chart objects that have a graphical representation, i.e. that consist of lines, bars, arcs, text, etc., are subclasses of the **GraphObj** abstract base class. This includes all of the axis classes, axis label classes, plot classes, text classes and legend classes among others. You must explicitly add objects of this type to the **ChartView** object, using the **ChartView.addChartObject** method, after they have been created and initialized. Otherwise, the object will not be included in the draw list of the **ChartView**. The example below adds an axis object to the **ChartView** draw list.

ChartView.addChartObject example (extracted from the example program ScatterPlot,BuildLabeledDatapoints)

[TypeScript]

```
public async  BuildLabeledDatapoints(canvasid: string) {

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

let chartVu: QCChartTS.ChartView = QCChartTS.ChartView.newChartView(htmlcanvas);
if (!chartVu) return;
chartVu.setPreferredSize(800, 600);
let theFont: QCChartTS.ChartFont;
let numPoints: number = 19;
let x1: number[] = new Array(numPoints);
let y1: number[] = new Array(numPoints);
let i: number;
for (i = 0; i < numPoints; i++) {
    x1[i] = (i + 1);
    y1[i] = 80.0 + 40.0 * (0.5 - QCChartTS.ChartSupport.getRandomDouble());
    if (y1[i] > 98) y1[i] -= 5;
}

theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
let Dataset1: QCChartTS.SimpleDataset = QCChartTS.SimpleDataset.newSimpleDataset("First", x1,
y1);
let pTransform1: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setGraphBorderDiagonal(0.15, .15, .90, 0.7);
let xAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.X_AXIS);
chartVu.addChartObject(xAxis);
```

[JavaScript]

```
 async BuildLabeledDatapoints(canvasid) {
let htmlcanvas = document.getElementById(canvasid);
let chartVu = QCChartTS.ChartView.newChartView(htmlcanvas);
if (!chartVu)
    return;
chartVu.setPreferredSize(800, 600);
let theFont;
let numPoints = 19;
let x1 = new Array(numPoints);
let y1 = new Array(numPoints);
let i;
for (i = 0; i < numPoints; i++) {
    x1[i] = (i + 1);
    y1[i] = 80.0 + 40.0 * (0.5 - QCChartTS.ChartSupport.getRandomDouble());
    if (y1[i] > 98)
      y1[i] -= 5;
}
theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
let Dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("First", x1, y1);
let pTransform1 =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setGraphBorderDiagonal(0.15, .15, .90, 0.7);
let xAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.X_AXIS);
chartVu.addChartObject(xAxis);
```

Rendering Order of GraphObj Objects

Each **GraphObj** object is added as an element to an **Array** object inside the **ChartView** class. When the chart view is rendered, it runs through the **GraphObj** objects stored in the list and renders them one by one to the current view. 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.

Each **GraphObj** object has a default z-order value, summarized below.

Base Class	**Default z-order value**	**Comments**
Background	10	Backgrounds are drawn first. A plot area background has a z-value of 10 and a graph area background has a z-value of 9, forcing graph area backgrounds to be drawn first.
Grid	40	A z-value of 40 places grids under most other graph objects. If you want grids on top change the z-value to 150.
GraphObj	50	The default value for graph objects if not explicitly changed in the subclass.
ChartText	50	The default value for text objects.
ChartPlot	50	The default value for plot objects which includes **SimplePlot**, **GroupPlot**, **PolarPlot,** and **AntennaPlot** objects.
Axis	100	Chart axes are drawn after data plots
AxisLabels	100	Axes labels are drawn with same priority as axes
Legend	150	Legend objects usually sit on top of all other graph objects and are drawn last

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.

The example below sets the z-order value of the x-axis to 30, changing the drawing order so that the x-axis draws before, and is therefore underneath, any **Grid** and **ChartPlot** objects in the view.

[TypeScript]

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

let chartVu: QCChartTS.ChartView = QCChartTS.ChartView.newChartView(htmlcanvas);
if (!chartVu) return;

let xAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.X_AXIS);
xAxis.setZOrder(30);
chartVu.addChartObject(xAxis);
```

[JavaScript]

```
let htmlcanvas = document.getElementById(canvasid);
let chartVu = QCChartTS.ChartView.newChartView(htmlcanvas);
if (!chartVu)
    return;
let xAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.X_AXIS);
xAxis.setZOrder(30);
chartVu.addChartObject(xAxis);
```

Dynamic or Real-Time Updates of Chart Objects

If you want to change the properties of one or more **GraphObj** derived objects displayed in the current graph, just go ahead and change them using the appropriate get and set methods. Once you change all of the properties that you want, call the **ChartView.updateDraw** method. This will force the **ChartView** object to update, redrawing every object in its draw list. See the example below.

[TypeScript]

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

let chartVu: QCChartTS.ChartView = QCChartTS.ChartView.newChartView(htmlcanvas);
if (!chartVu) return;

let xAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.X_AXIS);
xAxis.setZOrder(30);
chartVu.addChartObject(xAxis);
.
.
.
xAxis.setColor(QCChartTS.ChartColor.RED);
chartVu.updateDraw();
```

[JavaScript]

```
let htmlcanvas = document.getElementById(canvasid);
let chartVu = QCChartTS.ChartView.newChartView(htmlcanvas);
if (!chartVu)
    return;
let xAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.X_AXIS);
xAxis.setZOrder(30);
chartVu.addChartObject(xAxis);

.
.
.
xAxis.setColor(QCChartTS.ChartColor.RED);
chartVu.updateDraw();
```

You can change the values of a dataset, or even change the complete dataset of a chart plot object. Changing the values of the dataset will not show in the current graph until the **ChartView** repaints. Call the **ChartView.updateDraw** method to force a repaint. See the example programs DynamicCharts.BuildDynPieChart and DynamicCharts.BuildMixedPlot. The auto-scale methods are not automatically invoked if you change the values of dataset. If you want the graph to rescale taking into account the new data values you must call the appropriate autoscale methods of the coordinate system and of the related axes objects.

The chart classes that are NOT subclasses of **GraphObj** do not have a physical representation in a graph, so do not try to add them to the **ChartView** draw list. This includes the coordinate classes, the dataset classes and all of the utility classes. The **GraphObj** and **ChartView** classes use these classes for coordinate conversions, data storage, math calculations and I/O.

Delete a specific chart object from the **ChartView** draw list using the **ChartView.deleteChartObject** method. Clear the entire draw list using the **ChartView.resetChartObjectList** method. You can leave an object in the **ChartView** draw list but disable its display by calling that objects **GraphObj.setChartObjEnable** method. If you disable an object, you will still need to call the **ChartView.updateDraw** method to redraw the chart without that object.

Placing Multiple Charts in a ChartView

One way to create multiple charts is ti use multiple Canvas elements in your web page and then put a different ChartView in each one.

Or, you can have a single Canvas, combine it with a single ChartView, and combine multiple charts positioned within the ChartView canvas. The trick to doing this is to create separate coordinate system objects (**CartesianCoordinates**, **TimeCoordinates, PolarCoordinates,** or **AntennaCoordinates**) 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.BuildGroupBars, Bargraphs.BuildDoubleBarPlot, FinancialExamples.BuildOHLCChart, FinancialExamples.BuildFinOptions, DynamicCharts.BuildDynPieChart, PieCharts.BuildPieAndLineChart and PieCharts.BuildPieAndBarChart.

Or, you can have a single Canvas, and place multiple overlapping ChartViews within the single Canvas. Only the most recently drawn ChartView will be displayed. So you will have to create some sort of ChartView selection system, which is what all of our demo programs do, using the HTML button elements at the top of the page to select which chart is drawn

Multiple charts in a ChartView example (extracted from the example program FinancialExamples.BuildOHLCChart)

[TypeScript]

```
pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.setWeekType(weekmode);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
pTransform1.setGraphBorderDiagonal(0.1, .15, .90, 0.55);


pTransform2 = new QCChartTS.TimeCoordinates();
pTransform2.setWeekType(weekmode);
pTransform2.autoScaleDatasetRoundMode(Dataset2, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
this.SetInitialDates(new Date(), pTransform2);
pTransform2.setGraphBorderDiagonal(0.1, .675, .90, 0.85);
```

[JavaScript]

```
pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.setWeekType(weekmode);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
pTransform1.setGraphBorderDiagonal(0.1, .15, .90, 0.55);


pTransform2 = new QCChartTS.TimeCoordinates();
pTransform2.setWeekType(weekmode);
pTransform2.autoScaleDatasetRoundMode(Dataset2, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
this.SetInitialDates(new Date(), pTransform2);
pTransform2.setGraphBorderDiagonal(0.1, .675, .90, 0.85);
```

Multiple Coordinate Systems in the Same Chart

Often a chart needs more than one coordinate system to in order to support multiple x- and y-axes, each with different scales. As in the preceding section, this involves creating multiple coordinate systems. The plot areas for the coordinate systems can occupy separate space in the chart view, or they can overlap at the exact same position. As in the previous section, the position of each coordinate systems plot area in the chart view is set using one of the coordinate systems **setGraphBorder**… methods. Many of the examples use this technique, including FinancialExamples.BuildOHLCChart, MultipeAxis.BuildMultiAxes, ChartAxis.BuildLinearAxes, ChartAxis.BuildLogAxes and ChartAxis.BuildDateAxes1 and ChartAxis.BuildDateAxes2.

Multiple coordinate systems in a ChartView example

[TypeScript]

```
let xMin1: number = -5;
let xMax1: number = 15;
let yMin1: number = 0;
let yMax1: number =  105;
let pTransform1: QCChartTS.CartesianCoordinates  =
QCChartTS.CartesianCoordinates.newCartesianCoordinates(xMin1, yMin1, xMax1, yMax1);
pTransform1.setGraphBorderDiagonal(0.1, .15, .90, 0.6) ;

let xMin2: number = -50;
let xMax2: number = 150;
let yMin2: number = 0;
let yMax2: number =  1050;
let pTransform2: QCChartTS.CartesianCoordinates  =
QCChartTS.CartesianCoordinates.newCartesianCoordinates(xMin2, yMin2, xMax2, yMax2);
pTransform2.setGraphBorderDiagonal(0.1, .15, .90, 0.6) ;
```

[JavaScript]

```
let xMin1 = -5;
let xMax1 = 15;
let yMin1 = 0;
let yMax1 = 105;
let pTransform1 = QCChartTS.CartesianCoordinates.newCartesianCoordinates(xMin1, yMin1, xMax1,
yMax1);
pTransform1.setGraphBorderDiagonal(0.1, .15, .90, 0.6);
let xMin2 = -50;
let xMax2 = 150;
let yMin2 = 0;
let yMax2 = 1050;
let pTransform2 = QCChartTS.CartesianCoordinates.newCartesianCoordinates(xMin2, yMin2, xMax2,
yMax2);
pTransform2.setGraphBorderDiagonal(0.1, .15, .90, 0.6);
```

ChartView Object Resize Modes

Every **GraphObj** object has absolute size properties, such as font size or line thickness. Resize a window and these absolute size parameters are NOT changed. No matter how you resize a chart, if you set a text object to a font size of 10, the text object will always return a font size of 10, *regardless if the text now appears larger or smaller*. Instead, the value of the **resizeMultiplier** adjusts to represent the proportional change in the window size. In calculating the font size and the line thickness, the current size properties are multiplied by the **resizeMultiplier**. The initial value of the **resizeMultiplier** is 1.0 and the objects size properties correspond exactly to the initial settings. Shrink the **ChartView** window and the **resizeMultiplier** for each object is set to a value that is less than 1.0. Enlarge the window and the **resizeMultiplier** is set to a value greater than 1.0. The **ChartView** class has three resize modes that it can use to resize graph objects placed in the graph.

NO_RESIZE_OBJECTS	The **resizeMultiplier** stays fixed at 1.0. Resizing the graph window does not affect the size and thickness of the charts graph objects. Text will stay the same size and lines will stay the same thickness. The overall chart shrinks; it is just that size of the chart text and the thickness of the chart lines do not change. Resize the window small enough and the chart text will overlap and the lines used to draw the chart will look thick when compared to the chart size.
AUTO_RESIZE_OBJECTS	Resizing the graph window causes the size and thickness of the charts graph objects to resize. The auto-resize algorithm looks at which dimension changed the most (x or y) and uses the larger of the two changes to calculate new sizes for lines and text. Text and lines will shrink the same percentage. Resize the chart window with a minimum change in the charts aspect ratio, the change in the chart size will be very close to the change in the font size and line thickness. If the charts aspect ratio changes

drastically, the font size and line thickness will resize to reflect the dimension that was *reduced* the most. This minimizes the condition where text overlaps, though it may make the text unreadable if the chart size changes from large to a small with a large aspect ratio change.

MANUAL_RESIZE_OBJECTS

The **resizeMultiplier** for each object has an initial value of 1.0. Unlike the NO_RESIZE_OBJECTS mode, the value can be changed. The programmer must explicitly set the **resizeMultiplier** for any objects requiring a size change, using the **GraphObj.setResizeMultiplier** method.

The example program PieCharts demonstrates who to resize a chart in response to a resize of the parent HTML page. There are a couple of parts. The first is to add an event listener in the HTML page which traps the page resize event. There you use the resize information to define how you want to resize the internal HTMLCanvasElement which the chart is placed in. In this example we just set it to 0.8 of the inner width and inner height of the parent page.

Extracted from PieCharts.html

```
// in response to any window resize, this will force the canvas for every ChartView tied to the
// the chartCanvas1 canvas to update to the new size.
 window.addEventListener('resize', () =>
{
    var htmlcanvas = document.getElementById("chartCanvas1");
    if (htmlcanvas)
    {
    // This is just a simple brute force method to resize canvas to a percentage of the
    // height and width of the parent window. You may have your own algorithm.
    htmlcanvas.width = window.innerWidth * 0.8;
    htmlcanvas.height =  window.innerHeight * 0.8;

    // Force all charts tied to the canvas to resize internal objects
      PieCharts.ForceCanvasResize(htmlcanvas, true);

    }
});
```

The ForceCanvasResize function is a static function placed in the PieCharts.ts (PieCharts.js) file. It looks like:

```
public static  ForceCanvasResize(canvas: HTMLCanvasElement, update: boolean): boolean
{
   let result: boolean = QCChartTS.ChartView.updateChartsToCanvas(canvas,  update);
   return result;
}
```

where it forces all of the charts assigned to a Canvas to update using the current Canvas size parameters.

ChartView View Modes

A **ChartView** window can interact with a parent container, creating a couple of interesting view modes. The standard mode is to have a fixed size Canvas placed in a variable size HTML page. Size the Canvas larger than the HTML page and the HTML page acts as a "porthole" through which the **ChartView** window is viewed. Resizing the HTML page has no affect on the size of the Canvas and **ChartView** object within. Scroll bars will automatically appear on the sides of the HTML page to permit scrolling (panning) of the HTML view area around the Canvas. Most all of our example are setup this way, since most HTML pages do not resize their contents when the page is resized.

If you want the Canvas to always fit within the current size of the browser HTML page, use the resize techniqute descripted in the previous section. The example PieCharts demonstrates this technique.

Finding Chart Objects

The **ChartView** class is the central container class of the chart library. It keeps track of all of the objects in the chart. It includes a routine that can compare a test point against all of the objects in the chart and return an instance of an object that intersects the test point. The search can be restricted to a class and all subclasses of the specified class.

FindObj method

```
public findObj(testpoint: ChartPoint2D, classname: string): GraphObj | null
```

Parameters

If the graph has multiple overlapping objects of the same type, you can return the nth object intersecting the test point.

```
 public findObjNthHit(testpoint: ChartPoint2D, classname: string, nthhit: number): GraphObj |
null
```

testpoint	The current position of the mouse in Canvas device coordinates.
classname	The class name of the base class that is used to filter the desired class objects. The string "ChartPlot" would cause the routine to return only objects derived from the **ChartPlot** class.

nthhit	Specifies to return the nth object that intersects the test point. A value of 0 signifies that the first object found is returned, a value of 1 specifies that the second item found is returned, and so on.

The function returns a reference to the found object, or null if unsuccessful.

6. Colors, Gradients and Backgrounds

Class ChartAttribute

```
ChartObj
    |
    +-- ChartAttribute
```

All graphical object derived from our abstract **GraphObj** class include an instance of the **ChartAttribute** class. This class encapsulates common graphical line and fill style characteristics into a single class. If a graphical object is line based, it can have a line color, line style and a line thickness. Line based graphical objects include line-based plots (SimpleLinePlot, MultiLinePlot, OHLCPlot) all types of axes, and all types of text. If an object is area based, it can have a solid fill color, or a gradient of fill colors. The fill color fills the interior of the area object. Most area fill objects also use line attributes to define the color and line thickness of the outline of the area object. A bar can have an outline color different from the interior fill color. All of the bar graph plot types, scatter plot types, and pie charts are example of graphical objects which use the area fill solid color

ChartAttribute constructors

Use the constructor below for simple line and fill attributes. There are similar constructors with fewer parameters if all you want to do is set a line color, or a line color with a line thickness.

```
public static newChartAttributeGraphObj(source: GraphObj): ChartAttribute

public static newChartAttributePlotGroup(source: ChartPlot, ngroup: number): ChartAttribute

public static newChartAttributeColorWidthStyleFill(rgbcolor: number, rlinewidth: number, nlinestyle: number,
rgbfillcolor: number): ChartAttribute

public static newChartAttribute4(rgbcolor: number, rlinewidth: number, nlinestyle: number,
rgbfillcolor: number): ChartAttribute

public static newChartAttribute(rgbcolor: number, rlinewidth: number, nlinestyle: number,
rgbfillcolor: number): ChartAttribute

public static newChartAttributeColorWidthStyle(rgbcolor: number, rlinewidth: number, nlinestyle: number):
ChartAttribute
```

```
public static newChartAttribute3(rgbcolor: number, rlinewidth: number, nlinestyle: number):
ChartAttribute

public static newChartAttributeColorWidth(rgbcolor: number, rlinewidth: number): ChartAttribute

public static newChartAttribute2(rgbcolor: number, rlinewidth: number): ChartAttribute

public static newChartAttributeColor(rgbcolor: number): ChartAttribute

public static newChartAttributeAttrib(source: ChartAttribute): ChartAttribute
```

rgbcolor	The primary line and text color.
rlinewidth	The line width for all lines
nlinestyle	The line style for all lines.
rgbfillcolor	The fill color for solid objects
source	A ChartAttribute object to copy

[TypeScript]

```
let attrib1: QCChartTS.ChartAttribute  = QCChartTS.ChartAttribute.newChartAttribute
(QCChartTS.ChartColor.BLUE, 3,QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.BLUE);
let thePlot1: QCChartTS.SimpleLinePlot  =
QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, Dataset1, attrib1);
thePlot1.setLineStyle(QCChartTS.ChartConstants.LS_DASH_DOT);
chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
let attrib1 = QCChartTS.ChartAttribute.newChartAttribute (QCChartTS.ChartColor.BLUE,
3,QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.BLUE);
let thePlot1  =
QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, Dataset1, attrib1);
thePlot1.setLineStyle(QCChartTS.ChartConstants.LS_DASH_DOT);
chartVu.addChartObject(thePlot1);
```

All of the **ChartAttribute** constructors assume you are using a solid area fill color. If you want to use a gradient, you need to attach a **ChartGradient** object to the **ChartAttribute**. The **ChartGradient** object will specify the defining range of colors for the gradient, the breakpoints for the gradient colors, and the mapping mode for mapping the breakpoints to the current chart.

Class ChartGradient

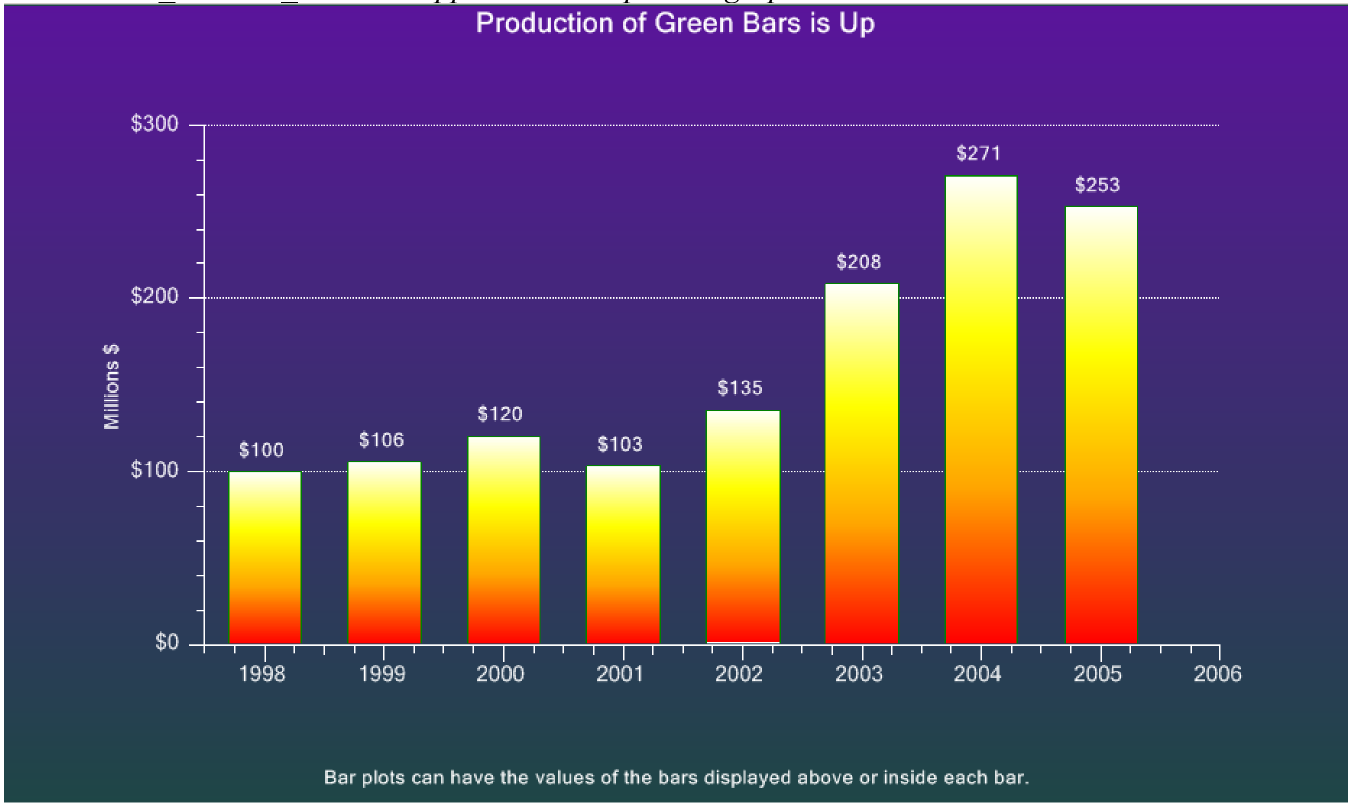
ChartObj
        |

+-- ChartGradient

All **ChartAttribute** objects include a reference to a **ChartGradient** object. Normally this reference is null, signifying that line and area fills work exactly the same as before. If the **ChartGradient** reference is not null, the color definitions in the **ChartGradient** take precedence over the fill color of the **ChartAttribute**. Any area fill object can have a gradient of two or more colors mapped to it. The colors are mapped to the area fill object using an array of breakpoints, one for each color, that define the transition points for one color to the next. The values of the breakpoints are interpreted according to one of four different mapping modes:

GRADIENT_MAPTO_OBJECT
In this mapping mode, the breakpoints are expected to be in the range of 0.0 to 1.0. The break points are applied as percentages to the area fill object. The value 0.0 corresponds to the start of the area fill object and the value 1.0 corresponds to the end of the area fill object. It does not matter how large or small the area fill object is, all of the gradient colors will map to that object. This mapping mode would normally used with just two colors, though it will work with an unlimited number of colors.

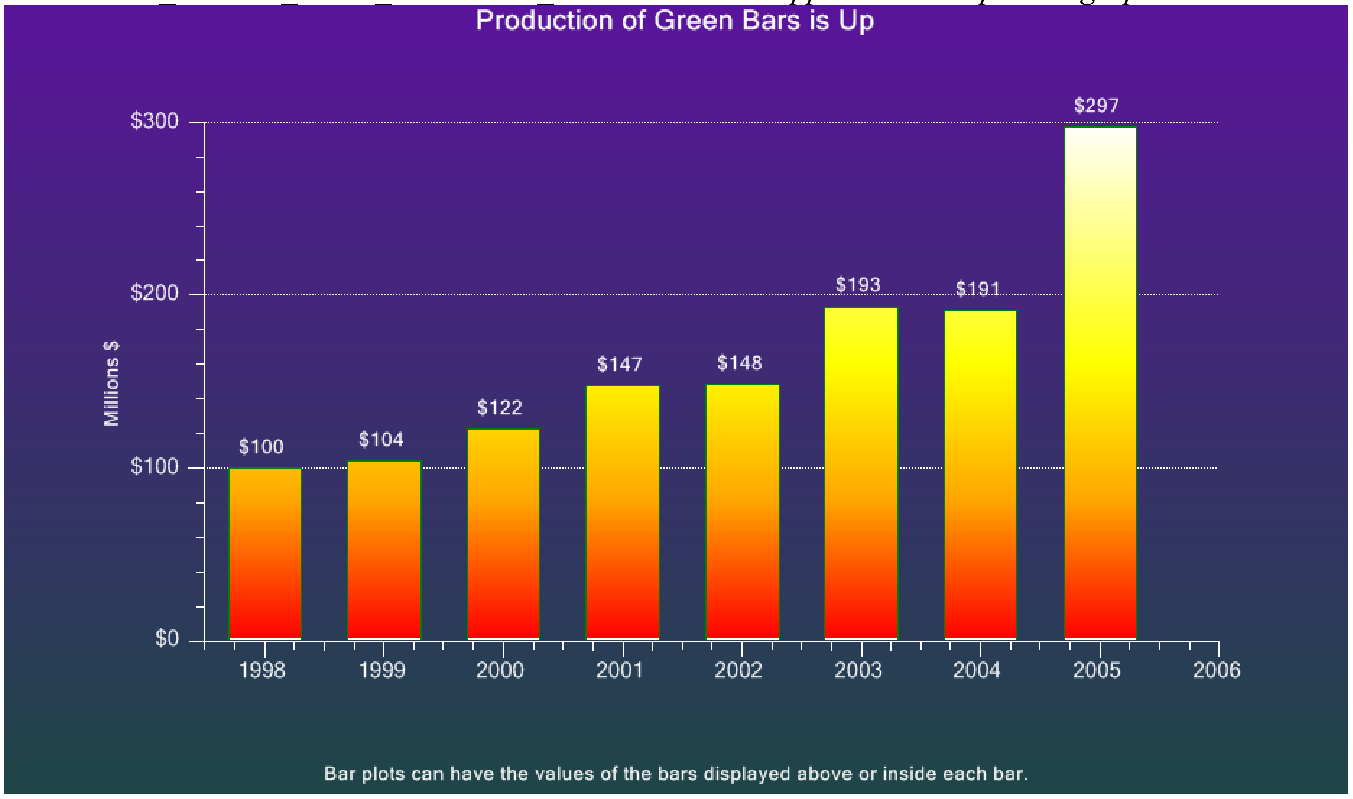
GRADIENT_MAPTO_OBJECT applied to a simple bar graph



Note how in this example, each bar displays the full range of colors (red, orange, yellow, and white), regardless of the bar height.

GRADIENT_MAPTO_PLOT_PHYSICAL_COORDINATES
In this mapping mode, the breakpoints are expected to be in the range of the physical coordinate system of the plot area of the chart. If the y-scale of the coordinate system has been scaled for 0 – 50,000, then the breakpoints are expected to be in the range of 0-50,000. This allows for the most interesting gradient effects. If you define your gradient breakpoints as extending from 0-50,000, and you plot a bar that is only 10,000 high, only the lower 20% of the gradient will be visible. This way, you can have bars change color as they increase in value. The best analogy would be if you were plotting temperature in a bar graph. As the temperature increases, and the height of the bar, the bar would display as a color gradient. The color gradient would transition from a dull red color, through orange, yellow and finally white, representing the highest color breakpoint.

GRADIENT_MAPTO_PLOT_PHYSICAL_COORDINATES applied to a simple bar graph



Note how in this example, the range of colors in each bar (red, orange, yellow, and white), depends on the bar height.

GRADIENT_MAPTO_PLOT_NORMALIZED_COORDINATES
In this mapping mode, the breakpoints are expected to be in the range of 0.0 to 1.0. The break points are applied as percentages to the plot area. The value 0.0 corresponds to the start of the plot area and the value 1.0 corresponds to the end of the plot area. Unlike the GRADIENT_MAPTO_OBJECT mapping mode, a small area fill object will not show all of the colors of the gradient. Only an area fill object the size of the plot area would show all of the colors. Otherwise, it can be used much the same as the

GRADIENT_MAPTO_PLOT_PHYSICAL_COORDINATES, mapping mode, except in this case you are using normalized coordinates (0.0 – 1.0) instead of physical coordinates.

GRADIENT_MAPTO_GRAPH_NORMALIZED_COORDINATES
Much the same as the GRADIENT_MAPTO_PLOT_PHYSICAL_COORDINATES, except that in this case the breakpoints are applied to the entire graph area, not the plot area.

ChartGradient constructors

Use the constructor below for simple line and fill attributes. There are similar constructors with fewer parameters if all you want to do is set a line color, or a line color with a line thickness.

```
public static newChartGradientTransform(transform: PhysicalCoordinates, startcolor: number,
endcolor: number, graddir: number): ChartGradient

public static newChartGradientTransformMode(transform: PhysicalCoordinates, gradmode: number,
startcolor: number, endcolor: number, graddir: number): ChartGradient

public static newChartGradientArray(transform: PhysicalCoordinates, gradmode: number, gradcolors:
number[], gradbreak: number[], graddir: number): ChartGradient
```

gradcolors	An array of colors used to define the gradient.
gradbreak	An array of gradient breakpoints (one for each color). The range of values depends on the mapping mode. For the GRADIENT_MAPTO_OBJECT, GRADIENT_MAPTO_PLOT_NORMALIZED_COORDINATES, and GRADIENT_MAPTO_GRAPH_NORMALIZED_COORDINATES modes, the first value of the array should always be 0.0 and the last value should always be 1.0.
graddir	The direction of the gradient in degrees. At 0 degrees, the direction is 3:00. Positive degrees rotate clockwise. When used with the *GRADIENT_MAPTO_PLOT_PHYSICAL_COORDINATES* mapping mode, always make degrees even divisible by 90.
gradmode	*The mapping mode of the breakpoints to the gradient area.* Use one of the gradient mode constants: GRADIENT_MAPTO_OBJECT, GRADIENT_MAPTO_PLOT_PHYSICAL_COORDINATES, GRADIENT_MAPTO_PLOT_NORMALIZED_COORDINATES, or GRADIENT_MAPTO_GRAPH_NORMALIZED_COORDINATES.
transform	The physical coordinate system of the graph object. The coordinate system is for the GRADIENT_MAPTO_PLOT_PHYSICAL_COORDINATES, GRADIENT_MAPTO_PLOT_NORMALIZED_COORDINATES, and GRADIENT_MAPTO_GRAPH_NORMALIZED_COORDINATES mapping modes.

The example below uses the GRADIENT_MAPTO_PLOT_PHYSICAL_COORDINATES mapping mode to map four colors to the physical coordinates of the plot area.

[TypeScript]

```
let attrib1:  QCChartTS.ChartAttribute  = QCChartTS.ChartAttribute.newChartAttribute
(QCChartTS.ChartColor.GREEN, 0,QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
let barcolors:  number []   = [QCChartTS.ChartColor.RED, QCChartTS.ChartColor.ORANGE,
QCChartTS.ChartColor.YELLOW, QCChartTS.ChartColor.WHITE];
let barbreakpoints:  number []  = [0.0, 80, 160, 240];

let gradmode: number  = QCChartTS.ChartConstants.GRADIENT_MAPTO_PLOT_PHYSICAL_COORDINATES;
let cg:  QCChartTS.ChartGradient  = QCChartTS.ChartGradient.newChartGradientArray(pTransform1,
gradmode, barcolors, barbreakpoints, -90 );
attrib1.Gradient = cg;
```

[JavaScript]

```
let attrib1 = QCChartTS.ChartAttribute.newChartAttribute(QCChartTS.ChartColor.GREEN, 0,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
let barcolors = [QCChartTS.ChartColor.RED, QCChartTS.ChartColor.ORANGE,
QCChartTS.ChartColor.YELLOW, QCChartTS.ChartColor.WHITE];
let barbreakpoints = [0.0, 80, 160, 240];
let gradmode = QCChartTS.ChartConstants.GRADIENT_MAPTO_PLOT_PHYSICAL_COORDINATES;
let cg = QCChartTS.ChartGradient.newChartGradientArray(pTransform1, gradmode, barcolors,
barbreakpoints, -90);
attrib1.Gradient = cg
```

The example below uses the GRADIENT_MAPTO_OBJECT mapping mode to map four colors to each bar of the bar plot, regardless of the bar size.

[TypeScript]

```
let attrib1: QCChartTS.ChartAttribute  = QCChartTS.ChartAttribute.newChartAttribute
(QCChartTS.ChartColor.GREEN, 0,QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
let barcolors: number []   = [QCChartTS.ChartColor.RED, QCChartTS.ChartColor.ORANGE,
QCChartTS.ChartColor.YELLOW, QCChartTS.ChartColor.WHITE];
let barbreakpoints : number []  = [0.0, 0.33, 0.66, 1.0];

let gradmode: number  = QCChartTS.ChartConstants. GRADIENT_MAPTO_OBJECT;
let cg: QCChartTS.ChartGradient  = QCChartTS.ChartGradient.newChartGradientArray(pTransform1,
gradmode, barcolors, barbreakpoints, -90 );
 attrib1.Gradient = cg;
```

[JavaScript]

```
let attrib1 = QCChartTS.ChartAttribute.newChartAttribute(QCChartTS.ChartColor.GREEN, 0,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
let barcolors = [QCChartTS.ChartColor.RED, QCChartTS.ChartColor.ORANGE,
QCChartTS.ChartColor.YELLOW, QCChartTS.ChartColor.WHITE];
let barbreakpoints = [0.0, 0.33, 0.66, 1.0];
let gradmode = QCChartTS.ChartConstants.GRADIENT_MAPTO_OBJECT;
let cg = QCChartTS.ChartGradient.newChartGradientArray(pTransform1, gradmode, barcolors,
barbreakpoints, -90);
attrib1.Gradient = cg
```

Class Background

Two rectangular areas act as a backdrop for the other graphical elements of a chart. The first is the rectangle formed by the **ChartView** class. This rectangle is the *graph area* and all elements of the chart (axes, labels, plots, titles, etc.) are within its bounds. The second area is the plot area. That area is within the graph area. Its position within the graph area is set using one of the **PhysicalCoordinates.setGraphBorder…** methods: **setGraphBorderDiagonal**, **setGraphBorderFrame** or **setGraphBorderInsets**. Typically the chart plot objects (line plots, bar plots, scatter plots, etc.) are clipped to the plot area. Other chart objects (axes, axes labels, titles, etc.) need to reside outside of the plot area and these objects clip to the graph area. The plot area can have a background different from that of the graph background. Often a contrast between the graph area background and the plot area background produces a more visually pleasing chart.

```
GraphObj
        |
        +-- Background
```

The **Background** class paints the graph area background or the plot area background. One instance of the class can only paint one area, either the graph area or the plot area. If you want unique fill properties for both, you need to create two instances of the class. The **Background** class uses one of the following techniques to fill the background:

- solid color
- simple color gradient defined as the linear interpolation of two RGB colors, in either the vertical or horizontal orientation.

Background constructors

```
public static newBackground(transform: PhysicalCoordinates, bgtype: number, bgcolor: number):
Background

public static newBackgroundCoordsTypeAttrib(transform: PhysicalCoordinates, bgtype: number,
attrib: ChartAttribute): Background

public static newBackgroundCoordsTypeCanvasGradient(transform: PhysicalCoordinates, bgtype:
number, gradient: CanvasGradient): Background

public static newBackgroundCoordsTypeColorGradient(transform: PhysicalCoordinates, bgtype:
number, startcolor: number, stopcolor: number, dir: number): Background

public static newBackgroundCoordsTypeStriped(transform: PhysicalCoordinates, bgtype: number,
color1: number, color2: number, barwidth: number, dir: number): Background
```

Use this constructor to fill the background with a single color.

```
public static newBackground(transform: PhysicalCoordinates, bgtype: number, bgcolor: number):
Background
```

Use this constructor to fill the background with the gradient defined using the *startcolor* and *stopcolor* arguments.

```
public static newBackgroundCoordsTypeColorGradient(transform: PhysicalCoordinates, bgtype:
number, startcolor: number, stopcolor: number, dir: number): Background
```

Use this constructor to fill the background with a user-defined gradient.

```
public static newBackgroundCoordsTypeCanvasGradient(transform: PhysicalCoordinates, bgtype:
number, gradient: CanvasGradient): Background
```

transform	The coordinate system associated with the chart background. The *transform* defines where the plot area fits in the graph area.
bgtype	The chart background type. Use one of the chart background type constants: PLOT_BACKGROUND or GRAPH_BACKGROUND. Specifying the PLOT_BACKGROUND type fills the plot area of the chart, while specifying the GRAPH_BACKGROUND type fills the entire graph area of the chart.
gradient	The user defined background gradient.
startcolor	Specifies the starting color value of the gradient.
stopcolor	Specifies the ending color value of the gradient.
dir	Specifies the direction of the gradient.

Should you want to use some sort of image as a background for the chart, use the **ChartImage** class and size it to fill the entire view.

The example below defines a simple linear gradient for the graph background area (extracted from the example program SimpleLinePlots.BuildLineFill).

[TypeScript]

```
pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.autoScaleDatasetsRoundMode(DatasetArray, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setGraphBorderDiagonal(0.15, .1, .92, 0.75);
```

```
let background: QCChartTS.Background =
QCChartTS.Background.newBackgroundCoordsTypeColorGradient(pTransform1,
QCChartTS.ChartConstants.GRAPH_BACKGROUND,
    QCChartTS.ChartColor.fromRgb(100, 50, 255), QCChartTS.ChartColor.fromRgb(40, 25, 120),
QCChartTS.ChartConstants.Y_AXIS);
chartVu.addChartObject(background);
let plotbackground: QCChartTS.Background = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.PLOT_BACKGROUND, QCChartTS.ChartColor.BLACK);
chartVu.addChartObject(plotbackground);
```

[JavaScript]

```
pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.autoScaleDatasetsRoundMode(DatasetArray, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setGraphBorderDiagonal(0.15, .1, .92, 0.75);
let background = QCChartTS.Background.newBackgroundCoordsTypeColorGradient(pTransform1,
QCChartTS.ChartConstants.GRAPH_BACKGROUND, QCChartTS.ChartColor.fromRgb(100, 50, 255),
QCChartTS.ChartColor.fromRgb(40, 25, 120), QCChartTS.ChartConstants.Y_AXIS);
chartVu.addChartObject(background);
let plotbackground = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.PLOT_BACKGROUND, QCChartTS.ChartColor.BLACK);
chartVu.addChartObject(plotbackground);
```

7. Axes

Axis
- **LinearAxis**
  - **ElapsedTimeAxis**
  - **PolarAxes**
  - **AntennaAxes**
  - **EventAxis**
- **LogAxis**
- **TimeAxis**

Chart axes describe for the viewer the physical coordinate system used to scale the plot area of a chart. A well-defined, visually appealing chart will display one or more axes with the following characteristics:

- Minimum and maximum values for axes endpoints that are appropriate for the displayed data
- Appropriately spaced axis tick marks that permit the user to easily interpolate by simple inspection data values located between labeled tick marks
- Axis tick mark labels that fall on logical, even intervals
- Flexible axis placement, inside or outside the plot area
- Axes for linear,date/time, elapsed time, event, logarithmic, polar and antenna, physical coordinate systems

The programmer can explicitly set these characteristics, or they can be calculated automatically based on an analysis of the associated chart data.

The axes of a chart do not define the physical coordinate system of the chart. Rather, the axes provide a visual key to the physical coordinate system. Define the physical coordinate system first using one of the classes derived from **PhysicalCoordinates**. Next, create the axes that reside in the physical coordinate. It is possible to define a physical coordinate system scaled using a xy range of (0-100, 0-100) and create an axis, residing in that coordinate system, that has minimum and maximum values of (0-25). The axis in that case takes up 25% of the chart plot area of the chart. Define the same axis with minimum and maximum values of (0-100) and the axes will span 100% of chart plot area.

A chart axis consists of at least two and usually three parts: the axis line, the axis tick marks, and the axis labels. The axis line extends from the minimum value to the maximum value of the axis. Major tick marks perpendicular to the axis line divide the axis line into sub ranges suitable for labeling. Minor tick marks, also perpendicular to the axis line, further subdivide the space between the major tick marks into even smaller intervals. Axis labels are optional. On one side of a chart there may be an axis with labels and on the other side an axis without labels.

Chart Axes

There are seven concrete axis types supported by the **QCChart2D for JavaScript/TypeScript** library:

Axis Type	**Class**
Linear	**LinearAxis**
Logarithmic	**LogAxis**
Date/time	**TimeAxis**
ElapsedTime	**ElapsedTimeAxis**
Event	**EventAxis**
Polar	**PolarAxes**
Antenna	**AntennaAxes**

The seven axis types derive directly or indirectly from the **Axis** abstract base class that provides a core set of properties and methods.

All axis objects use the same set of methods, found in the base **GraphObj** class, to set the drawing properties of the lines used to draw the axis line and tick marks. The default values use a black solid line of thickness 1.0. Change the default values using the **GraphObj** methods below.

setColor method

```
public setColor( rgbcolor: number);
```

setLineWidth method

```
public setLineWidth( linewidth: number);
```

setLineStyle method

```
Public setLineStyle(linestyle: number)
```

rgbcolor	Sets the primary line color for the chart object.
linewidth	Sets the line width, in device coordinates, for the chart object.

linestyle	Sets the line style for the chart object. Use one of the ChartConstants enumerated line style constants: LS_SOLID, LS_DASH_8_4, LS_DASH_4_4 , LS_DASH_4_2, LS_DASH_2_2, LS_DOT_1_1, LS_DOT_1_2, LS_DOT_1_4, LS_DOT_1_8, LS_DASH_DOT.

.

Linear Axes

Class LinearAxis

```
GraphObj
        |
        +--Axis
                |
                +-- LinearAxis
```

Linear Axis Minimum and Maximum

The axes minimum and maximum are the physical coordinate values that define the starting and ending points of the axis line. It is a mistake to try to invert the axis, i.e. an axis where the scale decreases from left to right, or bottom to top, by setting axis minimum to a value greater than the axis maximum. The software swaps the values if this happens. Create an inverted axis by first defining an inverted physical coordinate system using one of the **PhysicalCoordinates** derived classes. Place the axis in the inverted coordinate system.

The minimum and maximum of a linear axis can assume any numeric values. This differentiates the linear axis from logarithmic, time, polar, and antenna, axes which have specific, valid numeric ranges.

Linear Minor and Major Tick Mark Intervals

Major tick marks perpendicular to the axis line divide the line into sub ranges suitable for labeling. Minor tick marks, also perpendicular to the axis line, further subdivide the space between the major tick marks into even smaller intervals.

The major tick mark interval for a linear axis is set equal to a specified integer number of minor tick intervals, forcing major tick marks to always fall on a minor tick mark.

It is important that the tick mark intervals fall on rounded values appropriate to the physical scale of the chart. It is not appropriate to look at the range (maximum value – minimum value) and divide by some integer. For example, an axis with endpoints –5 to 30 should have a major tick mark interval of 5 or 10, and a minor tick mark interval 1.0. Dividing the axis range (30 – (-5) = 35.0) by 10 will result in a tick interval of 3.5, which is inappropriate for either major or minor tick intervals. The programmer can calculate the proper tick mark intervals using custom algorithms, or use the automatic methods that are used by default in the axis constructors.

Linear Axis Intercept

An axis resides in a 2-dimensional physical coordinate system. The minimum and maximum values for the axis provide coordinate information for only one dimension; x-coordinates in the case of an x-axis, and y-coordinates in the case of the y-axis. The missing coordinate needed to position the axis is the axis intercept. The axis intercept specifies the y-coordinate position for the x-axis, and the x-coordinate position for the y-axis.

Linear Axis Tick Mark Origin

The axis major and minor tick mark intervals specify the space between adjacent tick marks. A minor tick mark interval of 1.0, and a major tick mark interval of 5.0, may result in major tick marks at 0.0, 5.0, 10.0, 15.0, 20.0, etc. It can also result in major tick marks at –0.88769, 4.11231, 9.11231, 14.11231, 19.11231, etc. Obviously, the first example is the desired tick mark placement. The difference between the two examples is the tick mark starting point, or origin. In the first example, the tick mark origin is 0.0 and in the second case the tick mark origin is –0.88769. The tick mark origin is an important property because often it should not be the minimum value of the axis, but rather some intermediate value between the minimum and maximum value of the axis. In the example above, the data may range from –0.88769 to 19.9 and the chart is to have exactly that range. It is still appropriate that the tick mark origin be set to 0.0, rather than the axis minimum value of –0.88769.

The tick mark origin should reside in the bounds defined by the axis minimum and maximum, inclusive of the endpoints. It does not need to be near an endpoint however. For example, an axis with endpoints –16 to +19 should use a minor tick mark interval of 1.0 or 2.0, a major tick mark interval of 5.0 or 10.0, and a tick mark origin of 0.0. Usually, if the axis minimum and maximum bracket 0.0, i.e. the axis minimum is less than or equal to 0.0 and the axis maximum is greater than or equal to 0.0, the best tick mark origin to use is 0.0.

Creating a Linear Axis

There are two main constructors for **LinearAxis** objects. The first **LinearAxis** constructor assumes that the axis extents match the extents of the underlying coordinate system, *transform*. The second **LinearAxis** constructor sets the axis extents to the specified minimum and maximum values, regardless of the underlying coordinate system.

LinearAxis constructors

```
public static newLinearAxis(transform: PhysicalCoordinates, axtype: number): LinearAxis

public static newLinearAxisCoordsType(transform: PhysicalCoordinates, axtype: number): LinearAxis

public static newLinearAxisTypeMinMax(transform: PhysicalCoordinates, axtype: number, minval:
number, maxval: number): LinearAxis
```

transform	Places the axes in the coordinate system defined by transform.
axtype	Specifies if the axis is an x-axis (X_AXIS), or a y-axis (Y_AXIS).

minval	Sets the minimum value for the axis.
maxval	Sets the maximum value for the axis.

Other axis properties: minor tick mark spacing, number of minor tick marks per major tick mark, axis intercept, tick mark lengths, tick mark direction and axis tick mark origin are automatically calculated using an auto-axis method. Set these properties explicitly if you need to override the automatically calculated values.

setAxisIntercept method

```
public  setAxisIntercept(intercept: number);
```

setAxisTicks method

```
public setAxisTicks(tickorigin: number, tickspace: number,  ntickspermajor: number );

public setAxisTicksFull(tickorigin: number, tickspace: number, nminortickspermajor: number,
   minorticklength: number, majorticklength: number, tickdir);
```

intercept	Sets the intercept of this axis with the perpendicular axis in physical coordinates.
tickorigin	The tick marks start at this value.
tickspace	Specifies the spacing between minor tick marks.
ntickspermajor	Specifies the number of minor tick marks per major tick mark.
minorticklength	The length of minor tick marks, in Canvas device coordinates.
majorticklength	The length of major tick marks, in Canvas device coordinates.
tickdir	The direction of the tick marks. Use one of the tick mark direction constants: AXIS_MIN, AXIS_CENTER, or AXIS_MAX.

Use the **setLineWidth**, **setLineStyle** and **setColor** methods to customize the drawing properties of the lines used to draw the axis line and tick marks.

Simple linear axis example

[TypeScript]

```
// Define the coordinate system
let xMin: number = -5;
let xMax: number = 15;
let yMin: number = 0;
let yMax: number = 105;
let simpleScale: QCChartTS.CartesianCoordinates =
    QCChartTS.CartesianCoordinates.newCartesianCoordinates(xMin, yMin, xMax, yMax);

// Create the x- and y-axes
let xAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(simpleScale,
QCChartTS.ChartConstants.X_AXIS);
let yAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(simpleScale,
QCChartTS.ChartConstants.Y_AXIS);

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

[JavaScript]

```
// Define the coordinate system
let xMin = -5;
let xMax = 15;
let yMin = 0;
let yMax = 105;
let simpleScale = QCChartTS.CartesianCoordinates.newCartesianCoordinates(xMin, yMin, xMax, yMax);
// Create the x- and y-axes
let xAxis = QCChartTS.LinearAxis.newLinearAxis(simpleScale, QCChartTS.ChartConstants.X_AXIS);
let yAxis = QCChartTS.LinearAxis.newLinearAxis(simpleScale, QCChartTS.ChartConstants.Y_AXIS);
// Add the x- and y-axes to the chartVu object
chartVu.addChartObject(xAxis);
chartVu.addChartObject(yAxis);
```

Customize the axis by adding the following lines after the creation of the *xAxis* object:

Custom linear axis example

[TypeScript]

```
let  xAxisIntercept: number= -5;
let  xAxisOrigin: number = 0.0;
let  xAxisMinorTickSpace: number = 1.0;
let  xAxisMinorTicksPerMajor: number = 5;
let  xAxisMinorTickLength: number = 5;
let  xAxisMajorTickLength: number = 10;
let  xAxisTickDirection = QCChartTS.ChartConstants.AXIS_MIN;

xAxis.setAxisIntercept(xAxisIntercept);
xAxis.setAxisTicks(xAxisOrigin, xAxisMinorTickSpace,
                   xAxisMinorTicksPerMajor, xAxisMinorTickLength,
```

[JavaScript]

```
let xAxisIntercept = -5;
let xAxisOrigin = 0.0;
let xAxisMinorTickSpace = 1.0;
let xAxisMinorTicksPerMajor = 5;
let xAxisMinorTickLength = 5;
let xAxisMajorTickLength = 10;
let xAxisTickDirection = QCChartTS.ChartConstants.AXIS_MIN;

xAxis.setAxisIntercept(xAxisIntercept);
xAxis.setAxisTicks(xAxisOrigin, xAxisMinorTickSpace,
                   xAxisMinorTicksPerMajor, xAxisMinorTickLength,
```

Logarithmic Axes

Scientific, engineering and financial applications often require the use of logarithmic axe. Logarithmic axes are useful for the display of data that either has a wide dynamic range and/or data that is exponential in nature. Two common examples that use logarithmic scales are hi-fi speaker charts (db vs. log frequency) and stock market charts.

Class LogAxis

```
GraphObj
   |
   +-- Axis
          |
          +-- LogAxis
```

The **LogAxis** class is a concrete subclass of the **Axis** class. Use the **LogAxis** class to create a logarithmic axis with logarithmic spacing between the major tick marks (1, 10, 100…), and linear spacing (2, 3, 4, 5…) between the minor tick marks.

Logarithmic Axis Minimum and Maximum

The minimum and maximum values for a logarithmic axis can have any positive value, as long as the maximum is greater than the minimum. Create an inverted axis by first defining an inverted physical coordinate system using one of the **PhysicalCoordinates** derived classes. The axis minimum and maximum do not have to fall on decade intervals, i.e. 0.1 to 10,000 and can assume any positive range, i.e. 0.23 to 13,100 is valid.

Logarithmic Minor and Major Tick Mark Intervals

The major tick marks for a logarithmic axis use an exponential interval. The exponential interval in physical coordinates transforms to a linear interval in the working coordinate system. Below are examples of the major tick mark locations for a logarithmic axis.

Axis Minimum and Maximum	**Axis Major Tick Mark Locations**
0.1 to 100.0	0.1, 1.0, 10.0, 100.0
20 to 50,000	20, 200, 2000, 20000
10^{-4} to 1.0	10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1.0

The minor tick marks for a logarithmic axis use a linear interval between the tick marks. For example, a major tick mark interval has endpoints of 10 to 100, a logarithmic interval. The minor ticks in-between the 10 and the 100 use a linear interval of 10 and fall at 20, 30, 40, 50, 60, 70, 80, and 90. For the next major tick mark interval, 100 to 1000, the minor tick mark interval becomes 100 and minor tick marks fall at 200, 300, 400, 500, 600, 700, 800, and 900. The minor tick mark intervals are set equal to the value of the preceding major tick mark interval. If the major tick mark interval uses a non-decade range, for example 3, 30, 300, 30000, the minor tick marks will track the major tick marks. The major tick mark interval of 3 to 30 will use a minor tick mark range of 3, with minor tick marks at 6, 9, 12, 15, 18, 21, 24, and 27.

Logarithmic Axis Intercept

A logarithmic axis has an intercept value, the same as a linear axis. Since the intercept value is specified using the scale of the perpendicular axis, if the perpendicular axis is linear, as in the case of semi-log graphs, the intercept value can be positive, negative, or 0.0. If the perpendicular axis is logarithmic, the intercept value is restricted to a positive range.

Logarithmic Axis Tick Mark Origin

The starting value for the major tick marks does not need to fall at the end of the axis range. For example, the axis may have a range of 0.175 to 195. It would not make sense to start the major tick mark placement at 0.175. The major tick marks would end up placed at 0.175, 1.75, 17.5 and 175. The minor tick marks would make even less sense. A better major tick mark placement is 0.2, 2, 20, and 200. The minor tick marks will also fall on even values.

The logarithmic axis tick mark origin controls the placement of the first major tick mark. The other major and minor tick mark positions are automatically calculated based on the initial position of the first major tick mark.

The tick mark origin must reside in the bounds defined by the axis minimum and maximum, inclusive of the endpoints. It does not need to be near an endpoint however.

LogAxis Constructors

There are two constructors for **LogAxis** objects.

```
public static newLogAxis(transform: PhysicalCoordinates, axtype: number): LogAxis

public static newLogAxisType(transform: PhysicalCoordinates, axtype: number): LogAxis

public static newLogAxisTypeMinMax(transform: PhysicalCoordinates, axtype: number, minval:
number, maxval: number): LogAxis
```

transform	Places the axes in the coordinate system defined by transform.
axtype	Specifies if the axis is an x-axis (X_AXIS), or a y-axis (Y_AXIS).

minval	Sets the minimum value for the axis.
maxval	Sets the maximum value for the axis.

The first **LogAxis** constructor assumes that the axis extents match the extents of the underlying coordinate system, *transform*. The second **LogAxis** constructor sets the axis extents to the specified minimum and maximum values, regardless of the underlying coordinate system.

Other axis properties: axis intercept, tick mark lengths, tick mark direction and axis tick mark origin are automatically calculated using an auto-axis method. These properties can be explicitly set if you need to override the automatically calculated values.

setAxisIntercept method

```
public  setAxisIntercept(intercept: number);
```

setAxisTicks method

```
 public setAxisTicksOriginFormat(tickorigin: number, nlogtickformat: number)

 public setAxisTicks5(origin: number, nlogtickformat: number,
    minorticklength: number,
    majorticklength: number, tickdir: number)

 public setAxisTicks(origin: number, nlogtickformat: number,
    minorticklength: number,
    majorticklength: number, tickdir: number)
```

intercept Sets the intercept of this axis with the perpendicular axis in physical coordinates.

nlogtickformat This parameter specifies which minor tick marks are flagged for labels. Logarithmic axis minor tick mark labels can become very crowded. It is possible to choose values that may overlap or not display. Valid *nlogtickformat* values are:

0	No minor tick mark labels
1	Place a label at tick mark 0 in each decade.
2	Place a label at minor tick marks 1, 3 and 5 in each decade.
3	Place a label at minor tick marks 0, 1, 2, 3 and 5 in each decade.
4	Place a label at minor tick marks 0, 1, 2, 3, 4 and 5 in each decade.
5	Place a label at minor tick marks 0, 1, 2, 3, 4, 5 and 6 in each decade.
6	Place a label at minor tick marks 0, 1, 2, 3, 4, 5, 6 and 7 in each decade.

	7	Place a label at minor tick marks 0, 1, 2, 3, 4, 5, 6, 7 and 8 in each decade.
	8	Place a label at minor tick marks 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 in each decade.
minorticklength	The length of minor tick marks, in Canvas device coordinates.	
majorticklength	The length of major tick marks, in Canvas device coordinates.	
ticdir	The direction of the tick marks. Use one of the tick mark direction constants: AXIS_MIN, AXIS_CENTER, or AXIS_MAX.	

The **setLineWidth**, **setLineStyle** and **setColor** methods are used to customize the drawing properties of the lines used to draw the axis line and tick marks.

Simple log axis example

[TypeScript]

```
let xMin: number = 0;
let xMax: number = 1000;
let yMin: number = 0.2;
let yMax: number =  2000;
let  logYScale:  QCChartTS.CartesianCoordinates  =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE,  QCChartTS.ChartConstants.LOG_SCALE);
logYScale.setCoordinateBounds(xMin, yMin, xMax, yMax);

// Create a linear x-axis and a logarithmic y-axis
let xAxis: QCChartTS.LinearAxis  = QCChartTS.LinearAxis.newLinearAxis (logYScale,
QCChartTS.ChartConstants.X_AXIS);
let yAxis: QCChartTS.LogAxis  =  QCChartTS.LogAxis.newLogAxis (logYScale,
QCChartTS.ChartConstants.Y_AXIS);


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

[JavaScript]

```
let xMin = 0;
let xMax = 1000;
let yMin = 0.2;
let yMax = 2000;
let logYScale =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LOG_SCALE);
logYScale.setCoordinateBounds(xMin, yMin, xMax, yMax);
// Create a linear x-axis and a logarithmic y-axis
let xAxis = QCChartTS.LinearAxis.newLinearAxis(logYScale, QCChartTS.ChartConstants.X_AXIS);
let yAxis = QCChartTS.LogAxis.newLogAxis(logYScale, QCChartTS.ChartConstants.Y_AXIS);
// Add the x- and y-axes to the chartVu object
chartVu.addChartObject(xAxis);
chartVu.addChartObject(yAxis);)
```

Should want to customize the axis you can add the following lines after the *yAxis* object is created:

Custom logarithmic axis example

[TypeScript]

```
let yAxisIntercept: number = 1000;
// Major tick marks at 0.2, 2, 20, 200 and 2000
let yAxisOrigin : number= 0.2;
// In addition to major tick marks, labels flagged for some minor tick marks
let yAxisLogFormat: number = 1;
let yAxisMinorTickLength: number = 5;
let yAxisMajorTickLength: number = 10;
let yAxisTickDirection: number =  QCChartTS.ChartConstants.AXIS_MAX;

yAxis.setAxisIntercept(yAxisIntercept);
yAxis.setAxisTicks(yAxisOrigin, yAxisLogFormat, yAxisMinorTickLength,
   yAxisMajorTickLength, yAxisTickDirection);
```

[JavaScript]

```
let yAxisIntercept = 1000;
// Major tick marks at 0.2, 2, 20, 200 and 2000
let yAxisOrigin = 0.2;
// In addition to major tick marks, labels flagged for some minor tick marks
let yAxisLogFormat = 1;
let yAxisMinorTickLength = 5;
let yAxisMajorTickLength = 10;
let yAxisTickDirection = QCChartTS.ChartConstants.AXIS_MAX;
yAxis.setAxisIntercept(yAxisIntercept);
yAxis.setAxisTicks(yAxisOrigin, yAxisLogFormat, yAxisMinorTickLength, yAxisMajorTickLength,
yAxisTickDirection);
```

Date/Time Axes

The date/time axis is used in combination with a **TimeCoordinates** physical coordinate system. The axis major and minor tick marks correspond to the common date/time divisions of second, minute, hour, day, week, month and year. The date/time axes supported with this software are very complex, because they take into account the varying number of days in months and years. The axes also take into account non-continuous date/time scales where a 5-day week is used, or where a full day consists of a specific time interval that can be something less than a 24-hour day.

Note – The **TimeAxis** class is **not** used to create an axis to display elapsed time. Use a **ElapsedTimeCoordinates** system in combination with a **ElapsedTimeAxis** to create an elapsed time axis. It is the **ElapsedTimeAxisLabels** that give the elapsed time axis its time axis look, i.e. 10:30:22.

Class TimeAxis

GraphObj

|

```
+--Axis
        |
        +-- TimeAxis
```

The **TimeAxis** class creates an axis with date/time specific spacing between minor and major tick marks, not necessarily uniform as with a **LinearAxis**. The **TimeAxis** extends the **Axis** class.

Date/Time Axis Minimum and Maximum

The minimum and maximum values for a date/time axis can have any valid date/time value, specified using the class **Date**. The axis maximum value should be later in time than the minimum. Create an inverted axis by first defining an inverted physical coordinate system using the **TimeCoordinates** class. The axis minimum and maximum do not have to fall on even time or date intervals and can assume any date compatible with the **Date** class. For example:

Starting Date and Time	Ending Date and Time	Range
1/1/1972 00:00:00	1/1/1999 00:00:00	27 years
11/04/1997 8:30:00	11/04/1997 16:00:00	7 hours 30 minutes
11/28/2000 8:31:22	1/14/2001 15:14:33	48 days 6 hours 43 minutes 11 sec

Date/Time Minor and Major Tick Mark Intervals

The predefined date/time axis tick mark constants listed below specify both major and minor tick mark spacing.

Date/Time Axis Tick Mark Constants	Description
TIMEAXIS_50YEAR10YEAR	50 year major tick mark spacing, 10 year minor tick mark spacing
TIMEAXIS_20YEAR5YEAR	20 year major tick mark spacing, 5 year minor tick mark spacing
TIMEAXIS_10YEARYEAR	10 year major tick mark spacing, 1 year minor tick mark spacing
TIMEAXIS_5YEARYEAR	5 year major tick mark spacing, 1 year minor tick mark spacing

TIMEAXIS_YEAR	1 year major tick mark spacing
TIMEAXIS_YEARQUARTER	1 year major tick mark spacing, 1 quarter minor tick mark spacing
TIMEAXIS_YEARMONTH	1 year major tick mark spacing, 1 month minor tick mark spacing
TIMEAXIS_QUARTER	1 quarter major tick mark spacing
TIMEAXIS_QUARTERMONTH	1 quarter major tick mark spacing, 1 month minor tick mark spacing
TIMEAXIS_MONTH	1 month major tick mark spacing
TIMEAXIS_MONTHWEEK	1 month major tick mark spacing, 1 week minor tick mark spacing
TIMEAXIS_MONTHDAY	1 month major tick mark spacing, 1 day minor tick mark spacing
TIMEAXIS_WEEK	1 week major tick mark spacing
TIMEAXIS_WEEKDAY	1 week major tick mark spacing, 1 day minor tick mark spacing
TIMEAXIS_DAY	1 day major tick mark spacing
TIMEAXIS_DAY12HOUR	1 day major tick mark spacing, 12 hour minor tick mark spacing
TIMEAXIS_DAY8HOUR	1 day major tick mark spacing, 8 hour minor tick mark spacing
TIMEAXIS_DAY4HOUR	1 day major tick mark spacing, 4 hour minor tick mark spacing
TIMEAXIS_DAY2HOUR	1 day major tick mark spacing, 2 hour minor tick mark spacing
TIMEAXIS_DAYHOUR	1 day major tick mark spacing, 1 hour minor tick mark spacing
TIMEAXIS_12HOURHOUR	12 hour major tick mark spacing, 1 hour minor tick mark spacing

TIMEAXIS_8HOURHOUR	8 hour major tick mark spacing, 1 hour minor tick mark spacing
TIMEAXIS_4HOURHOUR	4 hour major tick mark spacing, 1 hour minor tick mark spacing
TIMEAXIS_2HOURHOUR	2 hour major tick mark spacing, 1 hour minor tick mark spacing
TIMEAXIS_HOUR	1 hour major tick mark spacing
TIMEAXIS_HOUR30MINUTE	1 hour major tick mark spacing, 30 minute minor tick mark spacing
TIMEAXIS_HOUR15MINUTE	1 hour major tick mark spacing, 15 minute minor tick mark spacing
TIMEAXIS_HOUR10MINUTE	1 hour major tick mark spacing, 10 minute minor tick mark spacing
TIMEAXIS_HOUR5MINUTE	1 hour major tick mark spacing, 5 minute minor tick mark spacing
TIMEAXIS_HOUR2MINUTE	1 hour major tick mark spacing, 2 minute minor tick mark spacing
TIMEAXIS_HOURMINUTE	1 hour major tick mark spacing, 1 minute minor tick mark spacing
TIMEAXIS_30MINUTEMINUTE	30 minute major tick mark spacing, 1 minute minor tick mark spacing
TIMEAXIS_15MINUTEMINUTE	15 minute major tick mark spacing, 1 minute minor tick mark spacing
TIMEAXIS_10MINUTEMINUTE	10 minute major tick mark spacing, 1 minute minor tick mark spacing
TIMEAXIS_5MINUTEMINUTE	5 minute major tick mark spacing, 1 minute minor tick mark spacing
TIMEAXIS_2MINUTEMINUTE	2 minute major tick mark spacing, 1 minute minor tick mark spacing
TIMEAXIS_MINUTE	1 minute major tick mark spacing
TIMEAXIS_MINUTE30SECOND	1 minute major tick mark spacing, 30 second minor tick mark spacing

TIMEAXIS_MINUTE15SECOND	1 minute major tick mark spacing, 15 second minor tick mark spacing
TIMEAXIS_MINUTE10SECOND	1 minute major tick mark spacing, 10 second minor tick mark spacing
TIMEAXIS_MINUTE5SECOND	1 minute major tick mark spacing, 5 second minor tick mark spacing
TIMEAXIS_MINUTE2SECOND	1 minute major tick mark spacing, 2 second minor tick mark spacing
TIMEAXIS_MINUTESECOND	1 minute major tick mark spacing, 1 second minor tick mark spacing
TIMEAXIS_30SECONDSECOND	30 second major tick mark spacing, 1 second minor tick mark spacing
TIMEAXIS_15SECONDSECOND	15 second major tick mark spacing, 1 second minor tick mark spacing
TIMEAXIS_10SECONDSECOND	10 second major tick mark spacing, 1 second minor tick mark spacing
TIMEAXIS_5SECONDSECOND	5 second major tick mark spacing, 1 second minor tick mark spacing
TIMEAXIS_2SECONDSECOND	2 second major tick mark spacing, 1 second minor tick mark spacing
TIMEAXIS_SECOND	1 second major tick mark spacing

Sunday is the first day of the week for 7-day weeks, while Monday is the first day of the week for 5-day weeks.

It may not be immediately obvious, but the major tick marks for date/time scales do not necessarily fall on equal intervals. If the tick marks are set to the TIMEAXIS_MONTHDAY value, there may be 28, 29, 30 or 31 days between each months major tick mark. In most cases, the major tick mark coincides with a minor tick mark. For example, if the TIMEAXIS_MONTHDAY setting is used, the major tick mark for a month falls at the first day of the month, coinciding with the minor tick mark for that day. If the TIMEAXIS_WEEKDAY setting is used, the major tick mark for a WEEK falls at the first day of the week, coinciding with the minor tick mark for that day. Situations where the major and minor tick marks do not coincide involve weeks as minor tick marks. If the TIMEAXIS_MONTHWEEK setting is used, the first day of the month may or may not correspond to the first day of the week (Sunday or Monday). Major tick marks fall on the first day of each month, and minor tick marks fall on the first day of each week.

Combine these irregularities with a 5- or 7-day workweek option, and the non-24 hour day option and you can see that a generalized algorithm to define the positions of tick marks is a difficult task. For example: you are a stock trader and you want to track the price/volume characteristics of a stock from the last 5 minutes of the regular trading day, to the first 5 minutes of the next regular trading day, specifically from 3:55 PM Friday, Sept 29, 2000 to 9:35 AM Monday, Oct 2, 2000. The time range under consideration is only 10 minutes, since the market closes Friday at 4:00 PM and opens Monday at 9:30 PM. The resulting axis should reflect this 10-minute range, even though the actual time elapsed is 3,930 minutes. You should be able to specify the axis minimum and maximum values using the actual dates and times. You also need to set a 5-day workweek mode and establish a day that has a time range of 9:30 AM to 4:00 PM.

Date/Time Axis Intercept

A date/time axis has an intercept value, the same as a linear axis. Since the intercept value is specified using the scale of the perpendicular axis, if the perpendicular axis is linear, the intercept value can be positive, negative, or 0.0. If the perpendicular axis is logarithmic, the intercept value is restricted to a positive range.

There are three main constructors for **TimeAxis** objects. The first two **TimeAxis** constructors assume that the axis extents match the extents of the underlying coordinate system, *transform*. The third **TimeAxis** constructor sets the axis extents to the specified minimum and maximum values, regardless of the underlying coordinate system.

TimeAxis constructors

```
 public static newTimeAxis(transform: TimeCoordinates): TimeAxis

 public static newTimeAxisMinMax(transform: TimeCoordinates, dstart: Date,
        dstop: Date): TimeAxis

 public static newTimeAxisTypeTimeBaseTick(transform: TimeCoordinates,
        axtype: number,
        ntickmarkbase: number,
        nminornthtick: number): TimeAxis

 public static newTimeAxisTypeTimeBase(transform: TimeCoordinates,
        axtype: number,
        ntickmarkbase: number): TimeAxis

 public static newTimeAxisType(transform: TimeCoordinates,
        axtype: number): TimeAxis
)
```

If the constructor does not explicitly specify whether the axis is for the x- or y-axis, then the software checks the underlying TimeCoordinates system (*transform*), and creates an axis for the dimension of the coordinate system that is time based.

transform	The time coordinate system the axis is placed in. If the starting and ending dates of the axis are not explicitly set, the axis uses the starting and ending dates of the *transform* time-scale.
axtype	The axis types. Use one of the axis type constants: X_AXIS or Y_AXIS..
dstart	The starting date value for the axis.
dstop	The ending date value for the axis
ntickmarkbase	This field defines the major and minor tick mark spacing for a time axis. Use one of the Date/time axis tick mark mode constants: TIMEAXIS_YEARMONTH, TIMEAXIS_DAYHOUR for example.

Other axis properties: axis intercept, tick mark lengths, tick mark direction are automatically calculated using an auto-axis method. These properties can be explicitly set if you need to override the automatically calculated values.

setAxisIntercept method

```
public  setAxisIntercept(intercept: number);
```

setAxisTicksAttributes method

```
public setAxisTicksAttributes(minorticklength: number,
        majorticklength: number, tickdir: number)
```

intercept	Sets the intercept of this axis with the perpendicular axis in physical coordinates.
minorticklength	Specifies the length of a minor tick mark in Canvas device coordinates.
majorticklength	Specifies the length of a major tick mark in Canvas device coordinates.
tickdir	Specifies the direction of the tick marks with respect to axis line. Use one of the following tick direction constants: AXIS_MIN, AXIS_CENTER, AXIS_MAX.

Customize the line and tick mark drawing properties of the axis using the **setLineWidth**, **setLineStyle** and **setColor** methods.

Simple time axis example

[TypeScript]

```
// Define a Time coordinate system
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);
```

[JavaScript]

```
// Define a Time coordinate system
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);
```

Custom time axis example

[TypeScript]

```
// Define a Time coordinate system
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 xAxisTickMarkFormat = QCChartTS.ChartConstants.TIMEAXIS_MONTHWEEK;
let xAxisMinorTickLength = 5;
let xAxisMajorTickLength = 10;
let xAxisTickDirection = QCChartTS.ChartConstants.AXIS_MIN;
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.newTimeAxisType(simpleTimeScale,
xAxisTickMarkFormat);
xAxis.setAxisTicksAttributes(xAxisMinorTickLength, xAxisMajorTickLength, xAxisTickDirection);

// 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]

```
// Define a Time coordinate system
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 xAxisTickMarkFormat = QCChartTS.ChartConstants.TIMEAXIS_MONTHWEEK;
let xAxisMinorTickLength = 5;
let xAxisMajorTickLength = 10;
let xAxisTickDirection = QCChartTS.ChartConstants.AXIS_MIN;
let simpleTimeScale = QCChartTS.TimeCoordinates.newTimeCoordinates(xMin, yMin, xMax, yMax);
// Create the time axis (x-axis is assumed)
let xAxis = QCChartTS.TimeAxis.newTimeAxisType(simpleTimeScale, xAxisTickMarkFormat);
xAxis.setAxisTicksAttributes(xAxisMinorTickLength, xAxisMajorTickLength, xAxisTickDirection);
// 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);
```

Elapsed Time Axes

Class ElapsedTimeAxis

```
GraphObj
   |
   +--Axis
        |
        +-- LinearAxis
              |
              +-- ElapsedTimeAxis
```

The **ElapsedTimeAxis** is subclassed from the **LinearAxis** class and has much in common with it. The only difference between the two is the way in which major and minor tick marks are calculated in the **event** method. The **ElapsedTimeAxis** takes into account the base 60 of seconds per minute and minutes per hour, and the base 24 of hours per day. Read the sections:

- **Linear Axis Minimum and Maximum**
- **Linear Minor and Major Tick Mark Intervals**
- **Linear Axis Intercept**
- **Linear Axis Tick Mark Origin**

under the discussion of **LinearAxis**.

Creating a Elapsed Time Axis

There are two main constructors for **ElapsedTimeAxis** objects. The first **ElapsedTimeAxis** constructor assumes that the axis extents match the extents of the underlying coordinate system, *transform*. The second **ElapsedTimeAxis** constructor sets the axis extents to the specified minimum and maximum values, regardless of the underlying coordinate system.

ElapsedTimeAxis constructors

```
public static newElapsedTimeAxis(transform: PhysicalCoordinates): ElapsedTimeAxis

public static newElapsedTimeAxisType(transform: PhysicalCoordinates, axtype: number):
ElapsedTimeAxis

public newElapsedTimeAxisAxisMinMax(transform: PhysicalCoordinates, axtype: number, minval:
number, maxval: number) : ElapsedTimeAxis

public newElapsedTimeAxisTypeTimeSpanMinTimeSpanMax(transform: PhysicalCoordinates, axtype:
number, minval: ChartTimeSpan, maxval: ChartTimeSpan): ElapsedTimeAxis
```

transform	Places the axes in the coordinate system defined by transform.
axtype	Specifies if the axis is an x-axis (X_AXIS), or a y-axis (Y_AXIS).
minval	Sets the minimum value for the axis. This can be either the numeric value in milliseconds, or a value represented by a ChartTiueSpan object.
maxval	Sets the maximum value for the axis. This can be either the numeric value in milliseconds, or a value represented by a ChartTiueSpan object.

Other axis properties: minor tick mark spacing, number of minor tick marks per major tick mark, axis intercept, tick mark lengths, tick mark direction and axis tick mark origin are automatically calculated using an auto-axis method. Set these properties explicitly if you need to override the automatically calculated values.

setAxisIntercept method

```
Public setAxisIntercept (intercept: number)
```

setAxisTicks method

```
public setAxisTicks(tickorigin: number, tickspace: number,
        nticksspermajor: number)

public setAxisTicksFull(tickorigin: number, tickspace: number,
        nminorticksspermajor: number, minorticklength: number,
        majorticklength: number, tickdir: number)
```

intercept	Sets the intercept of this axis with the perpendicular axis in physical coordinates.
tickorigin	The tick marks start at this value.
tickspace	Specifies the spacing between minor tick marks.
ntickspermajor	Specifies the number of minor tick marks per major tick mark.
minorticklength	The length of minor tick marks, in Canvas device coordinates.
majorticklength	The length of major tick marks, in Canvas device coordinates.
tickdir	The direction of the tick marks. Use one of the tick mark direction constants: AXIS_MIN, AXIS_CENTER, or AXIS_MAX.

Use the **setLineWidth**, **setLineStyle** and **setColor** methods to customize the drawing properties of the lines used to draw the axis line and tick marks.

Simple elapsed time axis example

[TypeScript]

```
// Define the coordinate system
let xMin = QCChartTS.ChartTimeSpan.fromSeconds(0);
let xMax = QCChartTS.ChartTimeSpan.fromSeconds(15);
let yMin = 0;
let yMax = 105;
let simpleScale = QCChartTS.ElapsedTimeCoordinates.newElapsedTimeCoordinatesTimeSpanX(xMin, yMin,
xMax, yMax);
// Create the x- and y-axes
let xAxis = QCChartTS.ElapsedTimeAxis.newElapsedTimeAxisType(simpleScale,
QCChartTS.ChartConstants.X_AXIS);
let yAxis = QCChartTS.LinearAxis.newLinearAxis(simpleScale, QCChartTS.ChartConstants.Y_AXIS);
// Add the x- and y-axes to the chartVu object
chartVu.addChartObject(xAxis);
chartVu.addChartObject(yAxis);
```

Event Axes

Class EventAxis

```
GraphObj
       |
       +--Axis
              |
              +-- LinearAxis
                      |
                      +--EventAxes
```

The **EventAxis** is subclassed from the **LinearAxis** class and has much in common with it. The major difference between the two is the way in which major and minor tick marks are calculated in the **event** method. The **EventAxis** places tick marks at the x-positions of the ChartEvent objects attached to the common EventCoordinate system. It places major tick marks at the ChartEvent objects which correspond to where an axis label is expected to go, and minor tick marks at events which fall between the major tick marks. If the tick marks start to overlap, tick marks are skipped in order to maintain the look of the chart.

Tick Rules

The TickRule property controls the tick mark logic of the axis. Use one of theTickRule constants found in the ChartConstants class:

NO_TICKS – do not display any tick marks. No tick marks means no axis labels.

MINOREVENT_MAJOREVENT – display a minor tick mark every AxisMinorNthTick event, and a major tick mark every AxisMinorTicksPerMajor.

MAJOREVENT - display a major tick mark every AxisMajorNthTick event.

MINORCROSSOVEREVENT_MAJORCROSSOVEREVENT – display a minor tick mark every AxisMinorNthTick, minor crossover event, and a major tick mark every AxisMajorNthTick major crossover event. The minor and major crossover events are controlled by the MajorTickCrossoverEvent and MinorTickCrossoverEvent properties of the EventAxis.

The default is MINORCROSSOVEREVENT_MAJORCROSSOVEREVENT.

The term crossover event means that an element of date/time timestamp changes. If you specify a MinorTickCrossoverEvent of QCChartTS.ChartConstants.SECOND, and an AxisMinorNthTick of 15, this will cause a minor tick mark to be displayed every 15th second, if an event falls within that range. So, if your events are spaced approximately 5 seconds apart, you will get a minor tick mark for approximately every three events. If you choose a MajorTickCrossoverEvent of QCChartTS.ChartConstants.MINUTE and an AxisMajorNthTick of 1, this will cause a major tick mark to be displayed every minute, if an event falls within that range. Tick marks only show up on an event, so if there are no events within the time interval, no tick mark will appear.

Creating a Event Axis

There are two main constructors for **EventAxis** objects. The first **EventAxis** constructor assumes that the axis extents match the extents of the underlying coordinate system, *transform*. The second **EventAxis** constructor sets the axis extents to the specified minimum and maximum values, regardless of the underlying coordinate system.

EventAxis constructors

```
public static newEventAxisCoordsTickRuleType(transform: EventCoordinates, tickrule: number,
axtype: number): EventAxis

public static newEventAxisCoordsType(transform: EventCoordinates, axtype: number): EventAxis

public static newEventAxisCoordsTickRuleTypeMinMax(transform: EventCoordinates, tickrule: number,
axtype: number, minval: number, maxval: number): EventAxis

public static newEventAxisCoordsTickRuleMinMax(transform: EventCoordinates, tickrule: number,
minval: number, maxval: number): EventAxis

public static newEventAxisCoords(transform: EventCoordinates): EventAxis
public static newEventAxis(transform: EventCoordinates): EventAxis
```

transform	Places the axes in the coordinate system defined by transform.
tickrule	Specifies the tick rule used to calculated the tick marks: NO_TICKS,MINOREVENT_MAJOREVENT, MAJOREVENT, MINORCROSSOVEREVENT_MAJORCROSSOVEREVENT, MAJORCROSSOVEREVENT.
axtype	Specifies if the axis is an x-axis (X_AXIS), or a y-axis (Y_AXIS).
minval	Sets the minimum value for the axis.
maxval	Sets the maximum value for the axis.

Other axis properties: minor tick mark spacing, number of minor tick marks per major tick mark, axis intercept, tick mark lengths, tick mark direction and axis tick mark origin are automatically calculated using an auto-axis method. Set these properties explicitly if you need to override the automatically calculated values.

setAxisIntercept method

```
Public setAxisIntercept (intercept: number)
```

setAxisTicks method

```
public setAxisTicks(tickorigin: number, tickspace: number,
        ntickspermajor: number)

public setAxisTicksFull(tickorigin: number, tickspace: number,
        nminorticksspermajor: number, minorticklength: number,
        majorticklength: number, tickdir: number)
```

intercept	Sets the intercept of this axis with the perpendicular axis in physical coordinates.

tickorigin	The tick marks start at this value.
tickspace	Not used in event axis.
ntickspermajor	Specifies the number of minor tick marks per major tick mark.
minorticklength	The length of minor tick marks, in Canvas device coordinates.
majorticklength	The length of major tick marks, in Canvas device coordinates.
tickdir	The direction of the tick marks. Use one of the tick mark direction constants: AXIS_MIN, AXIS_CENTER, or AXIS_MAX.

Use the **setLineWidth**, **setLineStyle** and **setColor** methods to customize the drawing properties of the lines used to draw the axis line and tick marks.

Simple event axis example

[TypeScript]

```
let Dataset1: QCChartTS.EventSimpleDataset  =
QCChartTS.EventSimpleDataset.newEventSimpleDataset("Actual Sales", chartevents);
let pTransform1:  QCChartTS.EventCoordinates  =
QCChartTS.EventCoordinates.newEventCoordinates(Dataset1);
pTransform1.setScaleStartY(0);
pTransform1.setGraphBorderDiagonal(0.15, .15, .9, 0.8);
let background:  QCChartTS.Background  =
QCChartTS.Background.newBackgroundCoordsTypeColorGradient(pTransform1,
QCChartTS.ChartConstants.GRAPH_BACKGROUND,
    QCChartTS.ChartColor.fromRgb(30, 70, 70), QCChartTS.ChartColor.fromRgb(90, 20, 155),
QCChartTS.ChartConstants.Y_AXIS);
chartVu.addChartObject(background);

let xAxis:  QCChartTS.EventAxis  =
QCChartTS.EventAxis.newEventAxisCoordsTickRuleType(pTransform1,
QCChartTS.ChartConstants.MAJOREVENT, QCChartTS.ChartConstants.X_AXIS);
xAxis.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(xAxis);

let yAxis:  QCChartTS.LinearAxis  = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
yAxis.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(yAxis);
```

[JavaScript]

```
let Dataset1 = QCChartTS.EventSimpleDataset.newEventSimpleDataset("Actual Sales", chartevents);
let pTransform1 = QCChartTS.EventCoordinates.newEventCoordinates(Dataset1);
pTransform1.setScaleStartY(0);
pTransform1.setGraphBorderDiagonal(0.15, .15, .9, 0.8);
```

```
let background = QCChartTS.Background.newBackgroundCoordsTypeColorGradient(pTransform1,
QCChartTS.ChartConstants.GRAPH_BACKGROUND, QCChartTS.ChartColor.fromRgb(30, 70, 70),
QCChartTS.ChartColor.fromRgb(90, 20, 155), QCChartTS.ChartConstants.Y_AXIS);
chartVu.addChartObject(background);
let xAxis = QCChartTS.EventAxis.newEventAxisCoordsTickRuleType(pTransform1,
QCChartTS.ChartConstants.MAJOREVENT, QCChartTS.ChartConstants.X_AXIS);
xAxis.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(xAxis);
let yAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.Y_AXIS);
yAxis.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(yAxis);
```

Polar Axes

Polar axes provide the visual scale needed to compare data values that use polar coordinates. A polar axis consists of two parts. The first part is a pair of linear x- and y-axes intersecting at the center, Cartesian coordinate (0, 0). The second part of a polar axis is a circle with radius R, centered on the origin.

Class PolarAxes

```
GraphObj
   |
   +--Axis
        |
        +-- LinearAxis
              |
              +-- PolarAxes
```

The **PolarAxes** class creates a polar axes object that combines linear x- and y-axes for measurement of the polar magnitude, and a circular axis for measurement of the polar angle. The **PolarAxes** class extends the **LinearAxis** class. This is useful because the **LinearAxis** already has member variables that define properties and draw the tick marks for the circular axis. The **PolarAxes** class also includes uses two additional **LinearAxis** objects in support of the x and y linear axes used in the drawing of the polar magnitude axes.

Polar Axis Minimum and Maximum

Polar axes have only one scaling value, the maximum value of the polar magnitude, designated R. The minimum value is always 0.0. The limits of the x- and y-axis are set to +-R with the intercept for each axis set to 0.0. The polar angle scale is always 360 degrees, corresponding to a full circle.

Polar Axis Minor and Major Tick Mark Intervals

Polar axes use two sets of tick mark properties; one set for the x- and y-axes and the other set for the circular axis. The x- and y-axes use the same values for the major and minor tick mark spacing, partitioning the axes between +-R endpoints. The circular axis also uses major and minor tick marks, analogous to the hour and minute marks of a clock.

PolarAxes constructor

There is only one constructor for **PolarAxes** objects.

```
public static newPolarAxes(transform: PolarCoordinates): PolarAxes
```

transform	The *transform* coordinate system defines the axis extents of the polar axes.

The **PolarAxes** constructor assumes that the axis extents match the extents of the underlying coordinate system, *transform*.

Other axis properties: minor tick mark spacing, number of minor tick marks per major tick mark, tick mark direction and tick mark lengths are automatically calculated using an auto-axis method. These properties can be explicitly set if you need to override the automatically calculated values.

```
public setPolarAxesTicks(axestickspace: number, axesntickspermajor: number,
        angletickspace: number, anglentickspermajor: number)

public setPolarAxesTicksFull(axestickspace: number, axesntickspermajor: number,
        angletickspace: number, anglentickspermajor: number,
        minorticklength: number, majorticklength: number,
        tickdir: number)
```

axestickspace	Specifies the spacing between minor tick marks for the x- and y-axes.
axesntickspermajor	Specifies the number of minor tick marks per major tick mark for the x- and y-axes.
angletickspace	Specifies the spacing, in degrees, between minor tick marks for the radial axis.
anglentickspermajor	Specifies the number of minor tick marks per major tick mark for the radial axis.
minorticlength	The length of minor tick marks, in Canvas device coordinates.
majorticlength	The length of major tick marks, in Canvas device coordinates.
tickdir	The direction of the tick marks. Use one of the tick mark direction constants: AXIS_MIN, AXIS_CENTER, or AXIS_MAX.

Use the **setLineWidth**, **setLineStyle** and **setColor** methods to customize the drawing properties of the lines used to draw the axes lines and tick marks.

Simple polar axes example

[TypeScript]

```
let polarmagnitude: number = 5;
let polarscale:  QCChartTS.PolarCoordinates =
QCChartTS.PolarCoordinates.newPolarCoordinates(polarmagnitude);
let polarAxes:  QCChartTS.PolarAxes = QCChartTS.PolarAxes.newPolarAxes(polarscale);

//  Add the polar axes to the chartVu object
chartVu.addChartObject(polarAxes);
```

[JavaScript]

```
let polarmagnitude = 5;
let polarscale = QCChartTS.PolarCoordinates.newPolarCoordinates(polarmagnitude);
let polarAxes = QCChartTS.PolarAxes.newPolarAxes(polarscale);
//  Add the polar axes to the chartVu object
chartVu.addChartObject(polarAxes);
```

Custom polar axes example

[TypeScript]

```
let polarmagnitude: number = 15;
let polarscale: QCChartTS.PolarCoordinates =
QCChartTS.PolarCoordinates.newPolarCoordinates(polarmagnitude);
let polarAxes: QCChartTS.PolarAxes =  QCChartTS.PolarAxes.newPolarAxes(polarscale);
let axesstickspace: number = 1;
let axesnticksspermajor: number = 5;
let angletickspace: number = 50;
let anglenticksspermajor: number = 6;
let minorticlength: number = 5;
let majorticlength: number = 10;
let tickdir: number = QCChartTS.ChartConstants.AXIS_CENTER;
polarAxes.setPolarAxesTicksFull(axesstickspace, axesnticksspermajor, angletickspace,
anglenticksspermajor, minorticlength, majorticlength, tickdir);
  //  Add the polar axes to the chartVu object
chartVu.addChartObject(polarAxes);
```

[JavaScript]

```
 let polarmagnitude = 15;
let polarscale = QCChartTS.PolarCoordinates.newPolarCoordinates(polarmagnitude);
let polarAxes = QCChartTS.PolarAxes.newPolarAxes(polarscale);
let axesstickspace = 1;
let axesnticksspermajor = 5;
let angletickspace = 50;
let anglenticksspermajor = 6;
let minorticlength = 5;
let majorticlength = 10;
let tickdir = QCChartTS.ChartConstants.AXIS_CENTER;
```

```
polarAxes.setPolarAxesTicksFull(axestickspace, axesnticksspermajor, angletickspace,
anglenticksspermajor, minorticlength, majorticlength, tickdir);
//  Add the polar axes to the chartVu object
chartVu.addChartObject(polarAxes);
```

Antenna Axes

Antenna axes provide the visual scale needed to compare data values that use antenna coordinates. An antenna axis consists of two parts. The first part is a linear y-axis extending from the origin to the outer edge of the radial scale. The second part of an antenna axis is a circle with a radius equal to the range of the radial scale, centered on the origin.

Class AntennaAxes

```
GraphObj
    |
    +--Axis
        |
        +-- LinearAxis
            |
            +-- AntennaAxes
```

The **AntennaAxes** class creates an antenna axes object that combines a linear y-axis for measurement of the radial values, and a circular axis for the measurement of the angular values. The **AntennaAxes** class extends the **LinearAxis** class. This is useful because the **LinearAxis** already has member variables that define properties and draw the tick marks for the circular axis.

Antenna Axis Minimum and Maximum

Antenna axes use two scaling values, a minimum and maximum radius value. The radius minimum value is set at the origin of the coordinate system, and the radius maximum value at the outer edge. The radius starting and ending values can be positive or negative. The maximum value should always be greater than the minimum value though. The a.ngular scale is always 360 degrees, corresponding to a full circle. The angular scale starts with 0 degrees at 12:00 and increases clockwise.

Antenna Axis Minor and Major Tick Mark Intervals

Antenna axes use two sets of tick mark properties; one set for the y-axis and the other set for the circular axis. The y-axis has major and minor tick mark properties, as does the circular axis.

AntennaAxes constructor

There is only one constructor for **AntennaAxes** objects.

```
public static newAntennaAxes(transform: AntennaCoordinates): AntennaAxes
```

transform	The *transform* coordinate system defines the axis extents of the antenna axes.

The **AntennaAxes** constructor assumes that the axis extents match the extents of the underlying coordinate system, *transform*.

Other axis properties: minor tick mark spacing, number of minor tick marks per major tick mark, tick mark direction and tick mark lengths are automatically calculated using an auto-axis method. These properties can be explicitly set if you need to override the automatically calculated values.

```
public setAntennaAxesTicks(axestickspace: number, axesntickspermajor: number,
angletickspace: number, anglentickspermajor: number)

public setAntennaAxesTicksFull(axestickspace: number, axesntickspermajor: number,
   angletickspace: number, anglentickspermajor: number,
   minorticklength: number, majorticklength: number,
   tickdir: number)
```

axestickspace	Specifies the spacing between minor tick marks for the x- and y-axes.
axesntickspermajor	Specifies the number of minor tick marks per major tick mark for the x- and y-axes.
angletickspace	Specifies the spacing, in degrees, between minor tick marks for the radial axis.
anglentickspermajor	Specifies the number of minor tick marks per major tick mark for the radial axis.
minorticlength	The length of minor tick marks, in Canvas device coordinates.
majorticlength	The length of major tick marks, in Canvas device coordinates.
tickdir	The direction of the tick marks. Use one of the tick mark direction constants: AXIS_MIN, AXIS_CENTER, or AXIS_MAX.

Use the **setLineWidth**, **setLineStyle** and **setColor** methods to customize the drawing properties of the lines used to draw the axes lines and tick marks.

Simple antenna axes example

[TypeScript]

```
let minvalue: number = -40;
let maxvalue: number = 20;
let antennascale: QCChartTS.AntennaCoordinates =
QCChartTS.AntennaCoordinates.newAntennaCoordinates(minvalue, maxvalue);
let antennaAxes: QCChartTS.AntennaAxes =  QCChartTS.AntennaAxes.newAntennaAxes(antennascale);
//  Add the Antenna axes to the chartVu object
chartVu.addChartObject(antennaAxes);
```

[JavaScript]

```
let minvalue = -40;
let maxvalue = 20;
let antennascale = QCChartTS.AntennaCoordinates.newAntennaCoordinates(minvalue, maxvalue);
let antennaAxes = QCChartTS.AntennaAxes.newAntennaAxes(antennascale);
//  Add the Antenna axes to the chartVu object
chartVu.addChartObject(antennaAxes);
```

Custom antenna axes example

[TypeScript]

```
let minvalue: number = -40;
let maxvalue: number = 20;
let antennascale: QCChartTS.AntennaCoordinates =
QCChartTS.AntennaCoordinates.newAntennaCoordinates(minvalue, maxvalue);
let antennaAxes: QCChartTS.AntennaAxes = QCChartTS.AntennaAxes.newAntennaAxes(antennascale);
let axestickspace: number = 1;
let axesntickspermajor: number = 5;
let angletickspace: number = 5;
let anglentickspermajor: number = 6;
let minorticlength: number = 5;
let majorticlength: number = 10;
let tickdir: number = QCChartTS.ChartConstants.AXIS_CENTER;
antennaAxes.setAntennaAxesTicksFull(axestickspace, axesntickspermajor, angletickspace,
anglentickspermajor, minorticlength, majorticlength, tickdir);
//  Add the Antenna axes to the chartVu object
chartVu.addChartObject(antennaAxes);
```

[JavaScript]

```
let minvalue = -40;
let maxvalue = 20;
let antennascale = QCChartTS.AntennaCoordinates.newAntennaCoordinates(minvalue, maxvalue);
let antennaAxes = QCChartTS.AntennaAxes.newAntennaAxes(antennascale);
let axestickspace = 1;
let axesntickspermajor = 5;
let angletickspace = 5;
let anglentickspermajor = 6;
let minorticlength = 5;
let majorticlength = 10;
let tickdir = QCChartTS.ChartConstants.AXIS_CENTER;
antennaAxes.setAntennaAxesTicksFull(axestickspace, axesntickspermajor, angletickspace,
anglentickspermajor, minorticlength, majorticlength, tickdir);
//  Add the Antenna axes to the chartVu object
chartVu.addChartObject(antennaAxes);
```

8. Axis Labels

AxisLabels

  NumericAxisLabels
  TimeAxisLabels
  ElapsedTimeAxisLabels
  EventAxisLabels
  StringAxisLabels
  PolarAxesLabels
  AntennaAxesLabels

Axis Labels

Axis labels are numeric or text strings placed next to axis tick marks, indicating the scale of the axis. Axis labels are a separate class from the axis classes. An axis class, i.e. any class derived from **Axis**, can exist independent of axis labels. Many graphs use axes that do not have labels. For example, the y-axis on the left side of a graph may have labels, while an identical axis on the right hand side of the graph may not. The axis label classes require a valid reference axis class and cannot exist independently.

There are seven axis labels classes: the **AxisLabels** abstract base class and concrete subclasses **NumericAxisLabels**, **TimeAxisLabels, ElapsedTimeAxisLabels, EventAxisLabels, StringAxisLabels**, **PolarAxesLabels** and **AntennaAxesLabels.**.

Label Formats

An axis label can take many forms. The various axis label formats are divided between the axis labels classes in the following manner.

NumericAxisLabels

The **LinearAxis** and **LogAxis** axis types use this class.

- Full decimal conversion (0.000015)
- Scientific notation (1.5e-5)
- Exponent notation (1.5x10^{-5})
- Percent format (76%)
- Business format where B, M and K are used to represent billions, millions and thousands (1043M, 11.0K)
- Currency format ($123432)
- Business currency format – The business format combined with the currency format ($123K)

StringAxisLabels

The **LinearAxis** and **LogAxis** axis types use this class.

Arbitrary strings (“MA”, “PA”, “JEFF”,”Sales”) are used to label the major tick marks of the axis.

TimeAxisLabels

The **TimeAxis** axis types use this class.

- Time formats (hh:mm:ss, hh:mm, mm:ss, etc.)
- Date formats (mm/dd/yy, dd/mm/yy, mm/yy, etc.)
- Time/Date formats (mmm/ddd/yyy hh:mm:ss)

ElapsedTimeAxisLabels

The **ElapsedTimeAxisLabels** are used in combination with the **ElapsedTimeAxis** type**.**

Elapsed time formats (hh:mm:ss, hh:mm, mm:ss, mm:ss.fff, etc.)

EventAxisLabels

The EventeAxisLabels are used in combination with the **EventAxis** type**.**

- Time formats (hh:mm:ss, hh:mm, mm:ss, etc.)
- Date formats (mm/dd/yy, dd/mm/yy, mm/yy, etc.)
- Time/Date formats (mmm/ddd/yyy hh:mm:ss)
- Elapsed time formats (hh:mm:ss, hh:mm, mm:ss, etc.)
- Numeric formats (1.234)

PolarAxesLabels

This class displays numeric labels exclusively for the **PolarAxes** class. It uses labels in the same format as the **NumericAxisLabels**.

AntennaAxesLabels

This class displays numeric labels exclusively for the **AntennaAxes** class. It uses labels in the same format as the **NumericAxisLabels**.

Class AxisLabels

GraphObj

|

```
+-- ChartText
        |
        +-- AxisLabels
```

The **AxisLabels** class is the abstract base class for all axis label objects. It contains the properties and methods common to subclasses implementing more specialized axis labels.

Axis Label Text

The **AxisLabels** class includes a reference to a ChartFont object. If a valid font is not supplied a default font is created and used. Every axis labels object can have a unique font associated with it. The **Font** object defines the font typeface, size, and style. The font for any of the axis labels can be set using the **AxisLabels.setTextFont** method. The **AxisLabels** class manages other text attributes not directly associated with the font. These include the text foreground color, the text background color and the rotation of the text if it is different from the normal horizontal orientation. It is common to rotate x-axis labels 90 degrees, so that they are vertical rather than horizontal, in order to squeeze more tick mark labels in along the x-axis.

Axis Labels Positioning

The **AxisLabels** class manages the placement of the axis labels with respect to the underlying axis tick marks. Labels can be place above or below the tick marks of a horizontal x-axis, and to the left or right of the tick marks for a vertical y-axis. The axis label justification constants AXIS_MIN and AXIS_MAX are used for this purpose. The AXIS_MIN constant places the text label on the side of the tick mark that is in the direction of the perpendicular axis coordinate system minimum. The AXIS_MAX is much the same, except that it places the label on the side that is in the direction of the perpendicular axis coordinate system maximum. Axis labels should not actually touch the tick marks, so x and y offsets are factored in. The programmer can modify these offsets.

ChartFont

The standard text font handling of HTML5 does not match that of other platforms we support: .Net, WPF, and Java. So we create a wrapper class (ChartFont) for HTML5 font properties (name, style and size) to make the font handling similar to these other languages. So text objects will always contain a reference to a ChartFont object to use when drawing the text.

Class ChartFont

```
public static newChartFont(family: string, style: number, size: number): ChartFont
public static newChartFont3(family: string, style: number, size: number): ChartFont
public static newChartFont1(family: string): ChartFont
public static newChartFont2(family: string, size: number): ChartFont
```

family	the font name
style	the font style (ChartFont.NORMAL, BOLD, PLAIN, ITALIC, BOLD_ITALIC)
size	the font size
src	the source font to copy

Note: ChartFont argument order – The argument order for the default ChartFont constructor is (*name, style, size)*. This is similar to the Java Font class, but different than the .Net Font class, which is *(name, size, style)*. Because the last two arguments (style and size) are both of type number, it is easy to swap their order. If you do that the text will not display correctly, with the spacing between lines especially bad. But correcting the argument order will correct it.

Numeric Axis Labels

Class NumericAxisLabels

```
GraphObj
   |
   +-- ChartText
          |
          +-- AxisLabels
                 |
                 +-- NumericAxisLabels
```

The **NumericAxisLabels** class extends the **AxisLabels** class, adding extensive numeric formatting capability. It labels axes created using the **LinearAxis** and **LogAxis** classes.

Label formats

An axis label can take many forms. Variations on these forms include:

- Full decimal conversion (0.000015)
- Scientific notation (1.5e-5)
- Exponent notation (1.5x10^{-5})
- Percent format (76%)
- Business format where B, M and K are used to represent billions, millions and thousands (1043M, 11.0K)
- Currency format ($123432)

- Business currency format – The business format combined with the currency format ($123K)

Depending on the scaling of the associated axis, the numeric values of the axis labels may be very large or very small numbers requiring a great deal of space to display. Various techniques are used to abbreviate the numeric value, reducing the space requirements. Expressing a numeric value in scientific notation can reduce the amount of space a label requires, if the numeric value requires eight or more digits. If the label values end in a lot of zeros (10000000, 9000000, 8000000…), a major reduction in space is achieved by using the business format which replaces all of the zeros with a letter (10M, 9M, 8M, …).

The axis numeric labels constants are listed below:

Numeric Format Constant	**Description**
DECIMALFORMAT	Decimal format, i.e. 1234.563
SCIENTIFICFORMAT	Scientific or exponential format: 1.23e3
BUSINESSFORMAT	Business format where the letters K, M, B or T are appended on the end a truncated numeric value, i.e. 1.23, 14K, 44M, 32B, 3.0T
ENGINEERINGFORMAT	If the absolute value of the label is greater than 1.0e6, or less than 1.0e-6, the scientific format is used, else the decimal format is used.
PERCENTFORMAT	The value of a label is multiplied by 100 and the character '%' is appended on the end of the label.
EXPONENTFORMAT	The value of a label is multiplied by 100 and the character '%' is appended on the end of the label.
CURRENCEYBUSINESSFORMAT	A '$' character is appended to the front of the label and the values are abbreviated the same as the BUSINESSFORMAT
CURRENCEYFORMAT	A '$' character is appended to the front of the label. .

NumericAxisLabels constructor

There is only one main constructor for **NumericAxisLabels** objects.

```
public static newNumericAxisLabels(baseaxis: Axis): NumericAxisLabels
```

baseaxis	This is the axis the axis labels are for.

Other axis label properties: font, rotation, numeric format, axis labels direction and numeric precision are automatically set. These properties can be explicitly set if you need to override the automatically calculated values.

setAxisLabels method

```
public setAxisLabels(
        font: ChartFont,
        rotation: number,
        labdir: number,
        decimalpos: number,
        labelends: number,
        labcolor: number)
```

setAxisLabelsFormat method

```
public setAxisLabelsFormat(format: number)
```

font	The font object used to display the axis label text.
rotation	The rotation, in degrees, of label text in the normal viewing plane.
labdir	The justification of the axis label (AXIS_MIN or AXIS_MAX) with respect to the tick mark endpoint.
decimal	Sets the number of digits to the right of the decimal point for numeric axis labels.
labelends	Specifies whether there should be labels for the axis minimum (LABEL_MIN), maximum (LABEL_MAX) or tick mark starting point (LABEL_ORIGIN). The value of these constants can be OR'd together. The value of LABEL_MIN | LABEL_MAX | LABEL_ORIGIN is LABEL_ALL
labcolor	The color of the label text.
format	Sets the numeric format for the axis labels. Use one of the numeric format constants: DECIMALFORMAT, SCIENTIFICFORMAT, EXPONENTFORMAT, BUSINESSFORMAT, ENGINEERINGFORMAT, PERCENTFORMAT, CURRENCEYFORMAT, CURRENCYBUSINESSFORMAT.

Simple numeric axis labels example

[TypeScript]

```
let xMin: number = -5;
let xMax: number = 15;
let yMin: number = 0;
let yMax: number =  105;
let simpleScale: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinates(xMin, yMin, xMax, yMax);
let xAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(simpleScale,
QCChartTS.ChartConstants.X_AXIS);
let yAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(simpleScale,
QCChartTS.ChartConstants.Y_AXIS);
let xAxisLabels: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(xAxis);
let yAxisLabels: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(yAxis);
//  Add the x- and y-axes to the chartVu object
chartVu.addChartObject(xAxis);
chartVu.addChartObject(yAxis);
chartVu.addChartObject(xAxisLabels);
chartVu.addChartObject(yAxisLabels);
```

[JavaScript]

```
//  Define the coordinate system
let xMin = -5;
let xMax = 15;
let yMin = 0;
let yMax = 105;
let simpleScale = QCChartTS.CartesianCoordinates.newCartesianCoordinates(xMin, yMin, xMax, yMax);
let xAxis = QCChartTS.LinearAxis.newLinearAxis(simpleScale, QCChartTS.ChartConstants.X_AXIS);
let yAxis = QCChartTS.LinearAxis.newLinearAxis(simpleScale, QCChartTS.ChartConstants.Y_AXIS);
let xAxisLabels = QCChartTS.NumericAxisLabels.newNumericAxisLabels(xAxis);
let yAxisLabels = QCChartTS.NumericAxisLabels.newNumericAxisLabels(yAxis);
//  Add the x- and y-axes to the chartVu object
chartVu.addChartObject(xAxis);
chartVu.addChartObject(yAxis);
chartVu.addChartObject(xAxisLabels);
chartVu.addChartObject(yAxisLabels);
```

Should want to customize the axis you can add the following lines after the *xAxisLabels* object is created:

Custom numeric axis labels example

[TypeScript]

```
let labelfont: QCChartTS.ChartFont =
QCChartTS.ChartFont.newChartFont("Helvetica",QCChartTS.ChartFont.BOLD, 12);
let xAxisLabelsRotation: number = 0;
let xAxisLabelsDir: number = QCChartTS.ChartConstants.AXIS_MIN;
let xAxisLabelsDecimal: number = 1;
let xAxisLabelsEnds: number = QCChartTS.ChartConstants.LABEL_ALL;
let xAxisLabelsColor: number = QCChartTS.ChartColor.BLACK;
let xAxisNumericFormat: number = QCChartTS.ChartConstants.DECIMALFORMAT;
xAxisLabels.setAxisLabels(labelfont, xAxisLabelsRotation, xAxisLabelsDir, xAxisLabelsDecimal,
xAxisLabelsEnds, xAxisLabelsColor);
xAxisLabels.setAxisLabelsFormat(xAxisNumericFormat);
```

[JavaScript]

```
let labelfont = QCChartTS.ChartFont.newChartFont("Helvetica", QCChartTS.ChartFont.BOLD, 12);
let xAxisLabelsRotation = 0;
let xAxisLabelsDir = QCChartTS.ChartConstants.AXIS_MIN;
let xAxisLabelsDecimal = 1;
let xAxisLabelsEnds = QCChartTS.ChartConstants.LABEL_ALL;
let xAxisLabelsColor = QCChartTS.ChartColor.BLACK;
let xAxisNumericFormat = QCChartTS.ChartConstants.DECIMALFORMAT;
xAxisLabels.setAxisLabels(labelfont, xAxisLabelsRotation, xAxisLabelsDir, xAxisLabelsDecimal,
xAxisLabelsEnds, xAxisLabelsColor);
```

```
xAxisLabels.setAxisLabelsFormat(xAxisNumericFormat);
```

String Axis Labels

Class StringAxisLabels

```
GraphObj
        |
        +-- ChartText
                |
                +-- AxisLabels
                        |
                        +-- StringAxisLabels
```

Use the **StringAxisLabels** class to label major tick marks of a linear or logarithmic axis with arbitrary strings.

StringAxisLabels constructor

There is only one main constructor for **StringAxisLabels** objects.

```
public static newStringAxisLabels(baseaxis: Axis): StringAxisLabels
```

baseaxis	This is the axis the axis labels are for.

The axis strings, and other axis label properties: font, rotation, numeric format, axis labels direction and numeric precision are automatically set. These properties can be explicitly set if you need to override the automatically calculated values.

setAxisLabels method

```
public setAxisLabels(
        font: ChartFont,
        rotation: number,
        labdir: number,
        labelends: number,
        labcolor: number,
        tickstringls: string[],
        numtickstringls: number)
)
```

font	The font object used to display the axis label text.
rotation	The rotation, in degrees, of label text in the normal viewing plane.

labdir	The justification of the axis label (AXIS_MIN or AXIS_MAX) with respect to the tick mark endpoint.
decimal	Sets the number of digits to the right of the decimal point for numeric axis labels.
labelends	Specifies whether there should be labels for the axis minimum (LABEL_MIN), maximum (LABEL_MAX) or tick mark starting point (LABEL_ORIGIN). The value of these constants can be OR'd together. The value of LABEL_MIN | LABEL_MAX | LABEL_ORIGIN is LABEL_ALL
labcolor	The color of the label text.
tickstrings	*A*array of strings, one for each major tick mark that you want labeled.
numtickstrings	The number of strings in the tickstrings array.

If you want the first major tick mark, usually the intercept of the y-axis with the x-axis, to remain empty, just initialize the .first element of the tickstrings array to the empty string, “”, as in the example below.

Simple string axis labels example, extracted from the BarGraphs.BuildLandOfTheFry example program.

[TypeScript]

```
let NumberOfCountries: number = 13;
let NumberOfGroups: number = 2;
let CountryNames: string[] = [
    "", "China", "Japan", "Russia", "Italy", "Sweden", "Denmark",
    "Mexico", "Brazil", "France", "Australia", "Spain", "United States", "England"];
//  Positions bar
let x1: number[] = new Array(NumberOfCountries);
.
.
.

//  Create but do not draw the linear y-axis (OBJECT_ENABLE_NODRAW)
let yAxis1: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
//  Force major tick mark every unit, no minor tick marks,
//  to match tick mark string labels
yAxis1.AxisTickSpace = 1;
yAxis1.AxisMinorTicksPerMajor = 1;
chartVu.addChartObject(yAxis1);
//  Simple numeric x-axis labels
let xAxisLab1: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(xAxis1);
xAxisLab1.setColor(QCChartTS.ChartColor.BLACK);
chartVu.addChartObject(xAxisLab1);
//  Y-axis string labels
//  Each string will label a major tick mark
```

```
let yAxisLab1: QCChartTS.StringAxisLabels =
QCChartTS.StringAxisLabels.newStringAxisLabels(yAxis1);
yAxisLab1.setAxisLabels(theFont, 0, QCChartTS.ChartConstants.AXIS_MIN,
QCChartTS.ChartConstants.LABEL_ALL, QCChartTS.ChartColor.BLACK, CountryNames, 14);
yAxisLab1.setColor(QCChartTS.ChartColor.BLACK);
chartVu.addChartObject(yAxisLab1);
```

[JavaScript]

```
let NumberOfCountries = 13;
let NumberOfGroups = 2;
let CountryNames = [
"", "China", "Japan", "Russia", "Italy", "Sweden", "Denmark",
"Mexico", "Brazil", "France", "Australia", "Spain", "United States", "England"
];
//Positions bar
let x1 = new Array(NumberOfCountries);
.
.
.

//Create but do not draw the linear y-axis (OBJECT_ENABLE_NODRAW)
let yAxis1 = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.Y_AXIS);
//Force major tick mark every unit, no minor tick marks,
//to match tick mark string labels
yAxis1.AxisTickSpace = 1;
yAxis1.AxisMinorTicksPerMajor = 1;
chartVu.addChartObject(yAxis1);
//Simple numeric x-axis labels
let xAxisLab1 = QCChartTS.NumericAxisLabels.newNumericAxisLabels(xAxis1);
xAxisLab1.setColor(QCChartTS.ChartColor.BLACK);
chartVu.addChartObject(xAxisLab1);
//Y-axis string labels
//Each string will label a major tick mark
let yAxisLab1 = QCChartTS.StringAxisLabels.newStringAxisLabels(yAxis1);
yAxisLab1.setAxisLabels(theFont, 0, QCChartTS.ChartConstants.AXIS_MIN,
QCChartTS.ChartConstants.LABEL_ALL, QCChartTS.ChartColor.BLACK, CountryNames, 14);
yAxisLab1.setColor(QCChartTS.ChartColor.BLACK);
chartVu.addChartObject(yAxisLab1);
```

Time and Date Axis Labels

Class TimeAxisLabels

```
GraphObj
       |
       +-- ChartText
               |
               +-- AxisLabels
                       |
                       +-- TimeAxisLabels
```

The **TimeAxisLabels** class extends the **AxisLabels** class, adding extensive time and date formatting capability. Use it to label axes created using the **TimeAxis** class.

Label formats

A time axis label can take many forms. Variations on these forms include:

- Time formats (hh:mm:ss, hh:mm, mm:ss, etc.)
- Date formats (mm/dd/yy, dd/mm/yy, mm/yy, etc.)

There are more ways to format time and date information than numeric data. The **QCChart2D for JavaScript/TypeScript** software directly supports twelve time formats and eighteen date formats. It is also possible to create custom date/time formats. The software makes use of the **Date.toString** `method` to format times and dates. A table listing predefined date/time formats appears below.

Date/Time Format Constant	**Format String**	**Example String Result**
TIMEDATEFORMAT_MSDDD	"mm:ss.fff"	12.33.999
TIMEDATEFORMAT_MSDD	"mm:ss.ff"	12.33.99
TIMEDATEFORMAT_MSD	"mm:ss.f"	12.33.9
TIMEDATEFORMAT_MS	"m:ss"	12:33
TIMEDATEFORMAT_12HMSDD	"h:mm.ss.ff"	11:12:33.99
TIMEDATEFORMAT_12HMSD	"h:mm.ss.f"	11:12:33.9
TIMEDATEFORMAT_12HMS	"h:mm:ss"	11:12:33
TIMEDATEFORMAT_12HM	"h:mm"	11:12
TIMEDATEFORMAT_24HMDDD	"H:mm:ss.ff"	23:12:33.99
TIMEDATEFORMAT_24HMDD	"H:mm:ss.f"	23:12:33.9
TIMEDATEFORMAT_24HMS	"H:mm:ss"	23:12:33
TIMEDATEFORMAT_24HM	"H:mm"	23:12
TIMEDATEFORMAT_STANDARD	"MMMMM dd, yyyy"	December 7, 2000
TIMEDATEFORMAT_MDY	"M/dd/yy"	12/07/00
TIMEDATEFORMAT_DMY	"d/MM/yy"	7/12/00
TIMEDATEFORMAT_MY	"M/yy"	7/00
TIMEDATEFORMAT_Q	None	Q1
TIMEDATEFORMAT_MMMM	"MMMM"	January

TIMEDATEFORMAT_MMM	"MMM"	Jan
TIMEDATEFORMAT_M	"MMM"	J
TIMEDATEFORMAT_DDDD	"dddd"	Tuesday
TIMEDATEFORMAT_DDD	"ddd"	Tue
TIMEDATEFORMAT_D	"ddd"	T
TIMEDATEFORMAT_Y	"yy"	00
TIMEDATEFORMAT_MDY2000	"M/dd/yyyy"	12/07/2000
TIMEDATEFORMAT_DMY2000	"d/MM/yyyy"	7/12/2000
TIMEDATEFORMAT_MY2000	"M/yyyy"	7/2000
TIMEDATEFORMAT_Y2000	"yyyy"	2000
TIMEDATEFORMAT_MDY_HMS	"H:mm:ss\nM/dd/yy"	12:23:33 12/07/00
TIMEDATEFORMAT_DMY_HMS	"H:mm:ss\nd/M/yy"	23:12:33 12/07/00
TIMEDATEFORMAT_MDY_HM	"H:mm\nM/dd/yy"	12:23:33 12/07/00
TIMEDATEFORMAT_DMY_HM	"H:mm\nd/M/yy"	23:12:33 12/07/00

TimeAxis Labels constructor

There is only one main constructor for **TimeAxisLabels** objects.

```
public static newTimeAxisLabels(baseaxis: TimeAxis): TimeAxisLabels
```

baseaxis	This is the time axis the axis labels are for.

Other axis label properties: font, rotation, numeric format, axis labels direction and numeric precision are automatically set. These properties can be explicitly set if you need to override the automatically calculated values.

setAxisLabels method

```
 public setAxisLabels(
        font: ChartFont,
        rotation: number,
        labdir: number,
        decimalpos: number,
        timeformat: number,
        labelends: number,
        labcolor: number)
```

setAxisLabelsFormat method

```
public setAxisLabelsFormat(format: number)
```

font	The font object used to display the axis label text.
rotation	The rotation, in degrees, of label text in the normal viewing plane.
labdir	The justification of the axis label (AXIS_MIN or AXIS_MAX) with respect to the tick mark endpoint.
decimal	Sets the number of digits to the right of the decimal point for numeric axis labels.
labelends	Ignored for time axis labels
labcolor	The color of the label text.
format	Sets the numeric format for the axis labels. Use one of the time format constants: TIMEDATEFORMAT_MSDDD, TIMEDATEFORMAT_MSDD, TIMEDATEFORMAT_MSD, TIMEDATEFORMAT_MS, TIMEDATEFORMAT_12HMSDD, TIMEDATEFORMAT_12HMSD, TIMEDATEFORMAT_12HMS, TIMEDATEFORMAT_12HM, TIMEDATEFORMAT_24HMSDD, TIMEDATEFORMAT_24HMSD, TIMEDATEFORMAT_24HMS, TIMEDATEFORMAT_24HM, TIMEDATEFORMAT_STANDARD, TIMEDATEFORMAT_MDY,TIMEDATEFORMAT_DMY, TIMEDATEFORMAT_MY, TIMEDATEFORMAT_Q, TIMEDATEFORMAT_MMMM, TIMEDATEFORMAT_MMM, TIMEDATEFORMAT_M, TIMEDATEFORMAT_DDDD, TIMEDATEFORMAT_DDD, TIMEDATEFORMAT_D, TIMEDATEFORMAT_Y, TIMEDATEFORMAT_MDY2000,

TIMEDATEFORMAT_DMY2000, TIMEDATEFORMAT_MY2000, TIMEDATEFORMAT_Y2000,TIMEDATEFORMAT_MDY_HMS, TIMEDATEFORMAT_DMY_HMS, TIMEDATEFORMAT_MDY_HM, TIMEDATEFORMAT_DMY_HM.

Simple time axis labels 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);
let xAxis: QCChartTS.TimeAxis =  QCChartTS.TimeAxis.newTimeAxis(simpleTimeScale);
let yAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(simpleTimeScale,
QCChartTS.ChartConstants.Y_AXIS);
let xAxisLabels: QCChartTS.TimeAxisLabels = QCChartTS.TimeAxisLabels.newTimeAxisLabels(xAxis);
let yAxisLabels: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(yAxis);

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

[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);
let xAxis = QCChartTS.TimeAxis.newTimeAxis(simpleTimeScale);
let yAxis = QCChartTS.LinearAxis.newLinearAxis(simpleTimeScale, QCChartTS.ChartConstants.Y_AXIS);
let xAxisLabels = QCChartTS.TimeAxisLabels.newTimeAxisLabels(xAxis);
let yAxisLabels = QCChartTS.NumericAxisLabels.newNumericAxisLabels(yAxis);
//  Add the x- and y-axes to the chartVu object
chartVu.addChartObject(xAxis);
chartVu.addChartObject(yAxis);
chartVu.addChartObject(xAxisLabels);
chartVu.addChartObject(yAxisLabels);
//  Add the x- and y-axes to the chartVu object
chartVu.addChartObject(xAxis);
chartVu.addChartObject(yAxis);
chartVu.addChartObject(xAxisLabels);
chartVu.addChartObject(yAxisLabels);bels);
```

Custom time axis labels example

[TypeScript]

```
let xAxisLabels: QCChartTS.TimeAxisLabels = QCChartTS.TimeAxisLabels.newTimeAxisLabels(xAxis);

let labelfont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont("Arial",
QCChartTS.ChartFont.BOLD, 12);
let xAxisLabelsRotation: number = 0;
let xAxisLabelsDir: number = QCChartTS.ChartConstants.AXIS_MIN;
let xAxisLabelsEnds: number = QCChartTS.ChartConstants.LABEL_ALL;
let xAxisLabelsColor: number = QCChartTS.ChartColor.BLACK;
let xAxisNumericFormat: number = QCChartTS.ChartConstants.TIMEDATEFORMAT_MY;
xAxisLabels.setAxisLabels(labelfont, xAxisLabelsRotation, xAxisLabelsDir, xAxisLabelsEnds,
xAxisLabelsColor);
xAxisLabels.setAxisLabelsFormat(xAxisNumericFormat);
```

[JavaScript]

```
let xAxisLabels = QCChartTS.TimeAxisLabels.newTimeAxisLabels(xAxis);

let labelfont = QCChartTS.ChartFont.newChartFont("Arial", QCChartTS.ChartFont.BOLD, 12);
let xAxisLabelsRotation = 0;
let xAxisLabelsDir = QCChartTS.ChartConstants.AXIS_MIN;
let xAxisLabelsEnds = QCChartTS.ChartConstants.LABEL_ALL;
let xAxisLabelsColor = QCChartTS.ChartColor.BLACK;
let xAxisNumericFormat = QCChartTS.ChartConstants.TIMEDATEFORMAT_MY;
xAxisLabels.setAxisLabels(labelfont, xAxisLabelsRotation, xAxisLabelsDir, xAxisLabelsEnds,
xAxisLabelsColor);
xAxisLabels.setAxisLabelsFormat(xAxisNumericFormat);
```

Elapsed Time Axis Labels

Class ElapsedTimeAxisLabels

```
GraphObj
       |
       +-- ChartText
              |
              +-- AxisLabels
                     |
                     +-- ElapsedTimeAxisLabels
```

The **ElapsedTimeAxisLabels** class extends the **AxisLabels** class and provides for elapsed time labels. It adds extensive time formatting capability. Use it to label axes created using the **ElapsedTimeAxis** class.

Elapsed Time Label formats

A time axis label can take several forms. The TimeFormat property controls the elapsed time format.

TimeFormat Format Constant	Example String Result
TIMEDATEFORMAT_MS	12:33

TIMEDATEFORMAT_24HMS	23:12:33
TIMEDATEFORMAT_24HM	23:12

The formats can also have a decimal precision, appending a decimal point and the specified number of significant digits to the right of the time lable seconds, i.e. 12:33.7432. This is set using the **AxisLabelsDecimalPos** property. The decimal value is only displayed if it is non-zero, and the AxisLabelsDecimalPos value is != 0;

ElapsedTimeAxis Labels constructor

There is only one main constructor for **TimeAxisLabels** objects.

```
public static newElapsedTimeAxisLabels(baseaxis: Axis): ElapsedTimeAxisLabels
```

baseaxis	This is the elapsed time axis the axis labels are for.

Other axis label properties: font, rotation, time format, axis labels direction and numeric precision are automatically set. These properties can be explicitly set if you need to override the automatically calculated values.

setAxisLabels method

```
 public setAxisLabels(
        font: ChartFont,
        rotation: number,
        labdir: number,
        decimalpos: number,
        timeformat: number,
        labelends: number,
        labcolor: number)
```

setAxisLabelsFormat method

```
public setAxisLabelsFormat(format: number)
```

font	The font object used to display the axis label text.
rotation	The rotation, in degrees, of label text in the normal viewing plane.
labdir	The justification of the axis label (AXIS_MIN or AXIS_MAX) with respect to the tick mark endpoint.

decimal	Sets the number of digits to the right of the decimal point for elapsed time axis labels.
labelends	Ignored for time axis labels
labcolor	The color of the label text.
timeformat	Sets the numeric format for the axis labels. Use one of the time format constants: TIMEDATEFORMAT_MS, TIMEDATEFORMAT_24HMS, TIMEDATEFORMAT_24HM.

Simple ElapsedTimeAxisLabels example (extracted from the NewDemosRev2.BuildElapsedTime example program)

[TypeScript]

```
let numPoints: number = 100;
let x1: QCChartTS.ChartTimeSpan[] = new Array<QCChartTS.ChartTimeSpan>(numPoints);
let y1: number[] = new Array(numPoints);
let y2: number[] = new Array(numPoints);
let i: number;
for (i = 0; (i < numPoints); i++) {
.
.
.

}

theFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif", QCChartTS.ChartFont.BOLD,
12);
let Dataset1: QCChartTS.ElapsedTimeSimpleDataset =
QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDataset("First", x1, y1);
let Dataset2: QCChartTS.ElapsedTimeSimpleDataset =
QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDataset("Second", x1, y2);
let pTransform1: QCChartTS.ElapsedTimeCoordinates =
QCChartTS.ElapsedTimeCoordinates.newElapsedTimeCoordinatesXYScale(QCChartTS.ChartConstants.ELAPSE
DTIME_SCALE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.elapsedTimeAutoScaleDatasetRoundMode(Dataset2, QCChartTS.ChartConstants.X_AXIS,
QCChartTS.ChartConstants.AUTOAXES_FAR, QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setGraphBorderDiagonal(0.15, .15, .90, 0.7);
let chartbackground: QCChartTS.Background = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.GRAPH_BACKGROUND, QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(chartbackground);
let background: QCChartTS.Background = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.PLOT_BACKGROUND, QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(background);
let xAxis: QCChartTS.ElapsedTimeAxis =
QCChartTS.ElapsedTimeAxis.newElapsedTimeAxisType(pTransform1, QCChartTS.ChartConstants.X_AXIS);
chartVu.addChartObject(xAxis);
let yAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
chartVu.addChartObject(yAxis);
let xAxisLab: QCChartTS.ElapsedTimeAxisLabels =
QCChartTS.ElapsedTimeAxisLabels.newElapsedTimeAxisLabels(xAxis);
xAxisLab.setTextFont(theFont);
xAxisLab.AxisLabelsDayFormat = QCChartTS.ChartConstants.ELAPSEDTIMEFORMAT_NEXTTODAYSTRING;
```

```
chartVu.addChartObject(xAxisLab);
let yAxisLab: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(yAxis);
yAxisLab.setTextFont(theFont);
chartVu.addChartObject(yAxisLab);
```

[JavaScript]

```
let numPoints = 100;
let x1 = new Array(numPoints);
let y1 = new Array(numPoints);
let y2 = new Array(numPoints);
let i;
for (i = 0; (i < numPoints); i++) {
.
.
.
}
theFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif", QCChartTS.ChartFont.BOLD,
12);
let Dataset1 = QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDatasetXTimeSpan("First",
x1, y1);
let Dataset2 = QCChartTS.ElapsedTimeSimpleDataset.newElapsedTimeSimpleDatasetXTimeSpan("Second",
x1, y2);
let pTransform1 =
QCChartTS.ElapsedTimeCoordinates.newElapsedTimeCoordinatesXYScale(QCChartTS.ChartConstants.ELAPSE
DTIME_SCALE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.elapsedTimeAutoScaleDatasetRoundMode(Dataset2, QCChartTS.ChartConstants.X_AXIS,
QCChartTS.ChartConstants.AUTOAXES_FAR, QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setGraphBorderDiagonal(0.15, .15, .90, 0.7);
let chartbackground = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.GRAPH_BACKGROUND, QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(chartbackground);
let background = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.PLOT_BACKGROUND, QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(background);
let xAxis = QCChartTS.ElapsedTimeAxis.newElapsedTimeAxisType(pTransform1,
QCChartTS.ChartConstants.X_AXIS);
chartVu.addChartObject(xAxis);
let yAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.Y_AXIS);
chartVu.addChartObject(yAxis);
let xAxisLab = QCChartTS.ElapsedTimeAxisLabels.newElapsedTimeAxisLabels(xAxis);
xAxisLab.setTextFont(theFont);
xAxisLab.AxisLabelsDayFormat = QCChartTS.ChartConstants.ELAPSEDTIMEFORMAT_NEXTTODAYSTRING;
chartVu.addChartObject(xAxisLab);
let yAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(yAxis);
yAxisLab.setTextFont(theFont);
chartVu.addChartObject(yAxisLab);
```

Event Axis Labels

Class EventAxisLabels

```
GraphObj
   |
   +-- ChartText
          |
          +-- AxisLabels
                 |
                 +-- EventAxisLabels
```

The **EventAxisLabels** class extends the **AxisLabels** class, adding extensive time and date formatting capability. Use it to label axes created using the **EventAxis** class.

Label formats

A time axis label can take many forms. Variations on these forms include:

- Time formats (hh:mm:ss, hh:mm, mm:ss, etc.)
- Date formats (mm/dd/yy, dd/mm/yy, mm/yy, etc.)
- Time/Date formats (mmm/ddd/yyy hh:mm:ss)
- Elapsed time formats (hh:mm:ss, hh:mm, mm:ss, etc.)
- Numeric formats (1.234)

There are more ways to format time and date information than numeric data. The **QCChart2D for JavaScript/TypeScript** software directly supports twelve time formats and twenty-two date formats. It is also possible to create custom date/time formats. The software makes use of the **Date.toString** `method` to format times and dates. A table listing predefined date/time formats appears below.

Date/Time Format Constant	**Format String**	**Example String Result**
TIMEDATEFORMAT_MSDDD	"mm:ss.fff"	12.33.999
TIMEDATEFORMAT_MSDD	"mm:ss.ff"	12.33.99
TIMEDATEFORMAT_MSD	"mm:ss.f"	12.33.9
TIMEDATEFORMAT_MS	"m:ss"	12:33
TIMEDATEFORMAT_12HMSDD	"h:mm.ss.ff"	11:12:33.99
TIMEDATEFORMAT_12HMSD	"h:mm.ss.f"	11:12:33.9
TIMEDATEFORMAT_12HMS	"h:mm:ss"	11:12:33
TIMEDATEFORMAT_12HM	"h:mm"	11:12
TIMEDATEFORMAT_24HMDDD	"H:mm:ss.ff"	23:12:33.99
TIMEDATEFORMAT_24HMDD	"H:mm:ss.f"	23:12:33.9
TIMEDATEFORMAT_24HMS	"H:mm:ss"	23:12:33
TIMEDATEFORMAT_24HM	"H:mm"	23:12

TIMEDATEFORMAT_STANDARD	"MMMMM dd, yyyy"	December 7, 2000
TIMEDATEFORMAT_MDY	"M/dd/yy"	12/07/00
TIMEDATEFORMAT_DMY	"d/MM/yy"	7/12/00
TIMEDATEFORMAT_MY	"M/yy"	7/00
TIMEDATEFORMAT_Q	None	Q1
TIMEDATEFORMAT_MMMM	"MMMM"	January
TIMEDATEFORMAT_MMM	"MMM"	Jan
TIMEDATEFORMAT_M	"MMM"	J
TIMEDATEFORMAT_DDDD	"dddd"	Tuesday
TIMEDATEFORMAT_DDD	"ddd"	Tue
TIMEDATEFORMAT_D	"ddd"	T
TIMEDATEFORMAT_Y	"yy"	00
TIMEDATEFORMAT_MDY2000	"M/dd/yyyy"	12/07/2000
TIMEDATEFORMAT_DMY2000	"d/MM/yyyy"	7/12/2000
TIMEDATEFORMAT_MY2000	"M/yyyy"	7/2000
TIMEDATEFORMAT_Y2000	"yyyy"	2000
TIMEDATEFORMAT_MDY_HMS	"H:mm:ss\nM/dd/yy"	12:23:33 12/07/00
TIMEDATEFORMAT_DMY_HMS	"H:mm:ss\nd/M/yy"	23:12:33 12/07/00
TIMEDATEFORMAT_MDY_HM	"H:mm\nM/dd/yy"	12:23:33 12/07/00
TIMEDATEFORMAT_DMY_HM	"H:mm\nd/M/yy"	23:12:33 12/07/00

EventAxis Labels constructor

There is only one main constructor for Event**AxisLabels** objects.

```
public static newEventAxisLabels(baseaxis: Axis): EventAxisLabels;
```

baseaxis This is the event axis the axis labels are for.

setAxisLabelMode method

Set the label mode of the event tick mark label: EVENT_TIME_LABEL, EVENT_NUMERIC_LABEL, EVENT_TICK_STRING_LABEL.

```
 public setEventLabelMode(value: number)
```

value set the mode value: EVENT_TIME_LABEL, EVENT_NUMERIC_LABEL, EVENT_TICK_STRING_LABEL

Other axis label properties: font, rotation, numeric format, axis labels direction and numeric precision are automatically set. These properties can be explicitly set if you need to override the automatically calculated values.

setAxisLabels method

```
public setAxisLabels(
        font: ChartFont,
        rotation: number,
        labdir: number,
        decimalpos: number,
        timeformat: number,
        labelends: number,
        labcolor: number)
```

setAxisLabelsFormat method

```
public setAxisLabelsFormat(format: number)
```

font	The font object used to display the axis label text.
rotation	The rotation, in degrees, of label text in the normal viewing plane.
labdir	The justification of the axis label (AXIS_MIN or AXIS_MAX) with respect to the tick mark endpoint.
decimal	Sets the number of digits to the right of the decimal point for numeric axis labels.
labelends	Ignored for time axis labels
labcolor	The color of the label text.
format	Sets the format for the axis labels. Use one of the time format constants:

For EventLabelMode = EVENT_TIME_LABEL

TIMEDATEFORMAT_MSDDD, TIMEDATEFORMAT_MSDD,
TIMEDATEFORMAT_MSD, TIMEDATEFORMAT_MS,
TIMEDATEFORMAT_12HMSDD, TIMEDATEFORMAT_12HMSD,
TIMEDATEFORMAT_12HMS, TIMEDATEFORMAT_12HM,
TIMEDATEFORMAT_24HMSDD, TIMEDATEFORMAT_24HMSD,
TIMEDATEFORMAT_24HMS, TIMEDATEFORMAT_24HM,
TIMEDATEFORMAT_STANDARD,
TIMEDATEFORMAT_MDY,TIMEDATEFORMAT_DMY,
TIMEDATEFORMAT_MY, TIMEDATEFORMAT_Q,
TIMEDATEFORMAT_MMMM, TIMEDATEFORMAT_MMM,
TIMEDATEFORMAT_M, TIMEDATEFORMAT_DDDD,
TIMEDATEFORMAT_DDD, TIMEDATEFORMAT_D, TIMEDATEFORMAT_Y,
TIMEDATEFORMAT_MDY2000, TIMEDATEFORMAT_DMY2000,
TIMEDATEFORMAT_MY2000, TIMEDATEFORMAT_Y2000,
TIMEDATEFORMAT_MDY_HMS,
TIMEDATEFORMAT_DMY_HMS,TIMEDATEFORMAT_MDY_HM
TIMEDATEFORMAT_DMY_HMS .

For EventLabelMode = EVENT_ELAPSEDTIME_LABEL

TIMEDATEFORMAT_MS, TIMEDATEFORMAT_24HMS,
TIMEDATEFORMAT_24HM.

For EventLabelMode = EVENT_NUMERIC_LABEL

DECIMALFORMAT, SCIENTIFICFORMAT, EXPONENTFORMAT, BUSINESSFORMAT, ENGINEERINGFORMAT, PERCENTFORMAT, CURRENCEYFORMAT, CURRENCYBUSINESSFORMAT.

Simple time axis labels example

[TypeScript]

```
let theFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont("Arial",
QCChartTS.ChartFont.BOLD, 12);
let Dataset1: QCChartTS.EventSimpleDataset =
QCChartTS.EventSimpleDataset.newEventSimpleDataset("Actual Sales", chartevents);
let pTransform1: QCChartTS.EventCoordinates =
QCChartTS.EventCoordinates.newEventCoordinates(Dataset1);
pTransform1.setScaleStartY(0);
let xAxis: QCChartTS.EventAxis = QCChartTS.EventAxis.newEventAxisCoordsTickRuleType(pTransform1,
QCChartTS.ChartConstants.MAJOREVENT, QCChartTS.ChartConstants.X_AXIS);
xAxis.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(xAxis);
let yAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
yAxis.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(yAxis);
let xAxisLab: QCChartTS.EventAxisLabels = QCChartTS.EventAxisLabels.newEventAxisLabels(xAxis);
xAxisLab.setAxisLabelsFormat(QCChartTS.ChartConstants.TIMEDATEFORMAT_Y2000);
xAxisLab.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(xAxisLab);
```

[JavaScript]

```
let theFont = QCChartTS.ChartFont.newChartFont("Arial", QCChartTS.ChartFont.BOLD, 12);
let Dataset1 = QCChartTS.EventSimpleDataset.newEventSimpleDataset("Actual Sales", chartevents);
let pTransform1 = QCChartTS.EventCoordinates.newEventCoordinates(Dataset1);
pTransform1.setScaleStartY(0);
let xAxis = QCChartTS.EventAxis.newEventAxisCoordsTickRuleType(pTransform1,
QCChartTS.ChartConstants.MAJOREVENT, QCChartTS.ChartConstants.X_AXIS);
xAxis.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(xAxis);
let yAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.Y_AXIS);
yAxis.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(yAxis);
let xAxisLab = QCChartTS.EventAxisLabels.newEventAxisLabels(xAxis);
xAxisLab.setAxisLabelsFormat(QCChartTS.ChartConstants.TIMEDATEFORMAT_Y2000);
xAxisLab.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(xAxisLab);
```

Numeric axis labels example

[TypeScript]

```
let xAxisLab: QCChartTS.EventAxisLabels = QCChartTS.EventAxisLabels.newEventAxisLabels(xAxis);
xAxisLab.EventLabelMode = QCChartTS.ChartConstants.EVENT_NUMERIC_LABEL;
xAxisLab.AxisLabelsFormat = QCChartTS.ChartConstants.DECIMALFORMAT;
xAxisLab.setTextFont(theFont);
```

[JavaScript]

```
let xAxisLab = QCChartTS.EventAxisLabels.newEventAxisLabels(xAxis);
xAxisLab.EventLabelMode = QCChartTS.ChartConstants.EVENT_NUMERIC_LABEL;
xAxisLab.AxisLabelsFormat = QCChartTS.ChartConstants.DECIMALFORMAT;
xAxisLab.setTextFont(theFont);
```

Custom time axis labels example

[TypeScript]

```
let xAxisLab: QCChartTS.EventAxisLabels = QCChartTS.EventAxisLabels.newEventAxisLabels(xAxis);

let xAxisLabelsRotation: number = 0;
let xAxisLabelsDir: number = QCChartTS.ChartConstants.AXIS_MIN;
let xAxisLabelsEnds: number = QCChartTS.ChartConstants.LABEL_ALL;
let xAxisLabelsColor: number = QCChartTS.ChartColor.BLACK;
let xAxisTimeFormat: number = QCChartTS.ChartConstants.TIMEDATEFORMAT_MY;
xAxisLab.setAxisLabels(theFont, xAxisLabelsRotation, xAxisLabelsDir, 2, xAxisTimeFormat,
xAxisLabelsEnds, xAxisLabelsColor);
```

[JavaScript]

```
let xAxisLab = QCChartTS.EventAxisLabels.newEventAxisLabels(xAxis);

let xAxisLabelsRotation = 0;
let xAxisLabelsDir = QCChartTS.ChartConstants.AXIS_MIN;
let xAxisLabelsEnds = QCChartTS.ChartConstants.LABEL_ALL;
let xAxisLabelsColor = QCChartTS.ChartColor.BLACK;
let xAxisTimeFormat = QCChartTS.ChartConstants.TIMEDATEFORMAT_MY;
xAxisLab.setAxisLabels(theFont, xAxisLabelsRotation, xAxisLabelsDir, 2, xAxisTimeFormat,
xAxisLabelsEnds, xAxisLabelsColor);
```

Polar Axes Labels

Class PolarAxesLabels

```
GraphObj
   |
   +-- ChartText
          |
          +-- AxisLabels
                 |
+-- NumericAxisLabels
       |
       +-- PolarAxesLabels
```

The **PolarAxesLabels** class extends the **NumericAxisLabels** class and creates labels for objects of the **PolarAxes** class. The **PolarAxesLabels** class labels the two parts of the polar axes: the x- and y-axes pair defining the polar magnitude, and the polar angle circle, bounding the x- and y-axes. The class extends the **NumericAxisLabels** class and uses that class's methods and properties for managing the label properties.

The x- and y-axes have extents of +-R. The only labels needed for these axes are for the positive section of the x-axis. The easiest way to manage this is to create a local x-axis that extends from 0 to +R. This local axis is not drawn, but is used to create a **NumericAxisLabels** object for the class. This object draws the labels for the positive section of the x-axis.

PolarAxisLabels constructor

There is only one main constructor for **PolarAxesLabels** objects.

```
public static newPolarAxesLabels(baseaxis: PolarAxes): PolarAxesLabels
```

baseaxis	This is the axis the axis labels are for.

Other axis label properties: font, rotation, numeric format, axis labels direction and numeric precision are automatically set. These properties can be explicitly set if you need to override the automatically calculated values.

setAxisLabels method

```
public setAxisLabels(
        font: ChartFont,
        decimalpos: number,
        labcolor: number)
```

setAxisLabelsFormat method

```
public setAxisLabelsFormat(format: number)
```

font	The font object used to display the axis label text.
labcolor	The color of the label text.

format	Sets the numeric format for the axis labels. Use one of the numeric format constants: DECIMALFORMAT, SCIENTIFICFORMAT, EXPONENTFORMAT, BUSINESSFORMAT, ENGINEERINGFORMAT, PERCENTFORMAT, CURRENCEYFORMAT, CURRENCYBUSINESSFORMAT.

Polar axes labels example

[TypeScript]

```
let polarmagnitude: number = 5;
let polarscale: QCChartTS.PolarCoordinates =
QCChartTS.PolarCoordinates.newPolarCoordinates(polarmagnitude);
let polarAxes: QCChartTS.PolarAxes = QCChartTS.PolarAxes.newPolarAxes(polarscale);
let polarAxesLabels: QCChartTS.PolarAxesLabels =
QCChartTS.PolarAxesLabels.newPolarAxesLabels(polarAxes);
polarAxesLabels.setAxisLabelsFormat(QCChartTS.ChartConstants.DECIMALFORMAT);
polarAxesLabels.setAxisLabelsDecimalPos(2);

//  Add the polar axes to the chartVu object
chartVu.addChartObject(polarAxes);
chartVu.addChartObject(polarAxesLabels);
```

[JavaScript]

```
let polarmagnitude = 5;
let polarscale = QCChartTS.PolarCoordinates.newPolarCoordinates(polarmagnitude);
let polarAxes = QCChartTS.PolarAxes.newPolarAxes(polarscale);
let polarAxesLabels = QCChartTS.PolarAxesLabels.newPolarAxesLabels(polarAxes);
polarAxesLabels.setAxisLabelsFormat(QCChartTS.ChartConstants.DECIMALFORMAT);
polarAxesLabels.setAxisLabelsDecimalPos(2);
//  Add the polar axes to the chartVu object
chartVu.addChartObject(polarAxes);
chartVu.addChartObject(polarAxesLabels);
```

Antenna Axes Labels

Class AntennaAxesLabels

```
GraphObj
   |
   +-- ChartText
          |
          +-- AxisLabels
                 |
                 +-- NumericAxisLabels
                        |
```

+-- AntennaAxesLabels

The **AntennaAxesLabels** class extends the **NumericAxisLabels** class and creates labels for objects of the **AntennaAxes** class. The **AntennaAxesLabels** class labels the two parts of the antenna axes: the y-axis displaying the radius limits, and the angular circle bounding the y-axis. The class extends the **NumericAxisLabels** class and uses that class's methods and properties for managing the label properties.

AntennaAxisLabels constructor

There is only one main constructor for **AntennaAxesLabels** objects.

```
public static newAntennaAxesLabels(baseaxis: AntennaAxes): AntennaAxesLabels
```

baseaxis	This is the axis the axis labels are for.

Other axis label properties: font, rotation, numeric format, axis labels direction and numeric precision are automatically set. These properties can be explicitly set if you need to override the automatically calculated values.

setAxisLabels method

```
public setAxisLabels(
        font: ChartFont,
        decimalpos: number,
        labcolor: number)
```

setAxisLabelsFormat method

```
public setAxisLabelsFormat(format: number)
```

font	The font object used to display the axis label text.
labcolor	The color of the label text.
format	Sets the numeric format for the axis labels. Use one of the numeric format constants: DECIMALFORMAT, SCIENTIFICFORMAT, EXPONENTFORMAT, BUSINESSFORMAT, ENGINEERINGFORMAT, PERCENTFORMAT, CURRENCEYFORMAT, CURRENCYBUSINESSFORMAT.

Antenna axes labels example

[TypeScript]

```
let minvalue: number = -40;
let maxvalue: number = 20;
let antennascale: QCChartTS.AntennaCoordinates =
QCChartTS.AntennaCoordinates.newAntennaCoordinates(minvalue, maxvalue);
let antennaAxes: QCChartTS.AntennaAxes = QCChartTS.AntennaAxes.newAntennaAxes(antennascale);
chartVu.addChartObject(antennaAxes);
let antennaAxesLabels: QCChartTS.AntennaAxesLabels =
QCChartTS.AntennaAxesLabels.newAntennaAxesLabels(antennaAxes);
antennaAxesLabels.setAxisLabelsFormat(QCChartTS.ChartConstants.DECIMALFORMAT);
antennaAxesLabels.setAxisLabelsDecimalPos(2);
chartVu.addChartObject(antennaAxesLabels);
```

[JavaScript]

```
let minvalue = -40;
let maxvalue = 20;
let antennascale = QCChartTS.AntennaCoordinates.newAntennaCoordinates(minvalue, maxvalue);
let antennaAxes = QCChartTS.AntennaAxes.newAntennaAxes(antennascale);
chartVu.addChartObject(antennaAxes);
let antennaAxesLabels = QCChartTS.AntennaAxesLabels.newAntennaAxesLabels(antennaAxes);
antennaAxesLabels.setAxisLabelsFormat(QCChartTS.ChartConstants.DECIMALFORMAT);
antennaAxesLabels.setAxisLabelsDecimalPos(2);
chartVu.addChartObject(antennaAxesLabels);
```

9. Axis Grids

Grid

    PolarAxesGrid
    AntennaGrid

Axis grids are solid, dotted or dashed lines, aligned with the axis tick marks and which extend across the plot area of a graph. Axis grids are a separate class from the axis classes. An axis class, i.e. any class derived from **Axis**, can exist independent of a grid. Many graphs use axes that do not have grids. For example, the y-axis may have a grid, while the x-axis may not. The axis grid classes are not independent and they require valid references to both an x- and y-axis. The three axis grid classes are **Grid, PolarGrid** and **AntennaGrid.**

Linear, Logarithmic and Time Axis Grids

Class Grid

```
GraphObj
   |
   +-- Grid
```

The **Grid** class defines a grid for **LinearAxis**, **LogAxis**, and **TimeAxis** classes. A grid object needs a reference to both an x- and y-axis. The grid aligns with the tick marks of one axis, and extends across the plot area using the minimum and maximum values of the other axis. For example, an x-axis grid has lines aligned with the tick marks of the x-axis and extending from the minimum y-value to the maximum y-value of the y-axis, parallel to the y-axis.

Grid constructor

```
public static newGrid(xaxis: Axis, yaxis: Axis, gridaxistype: number, gridtype: number): Grid
```

xaxis	The x-axis associated with the grid.
yaxis	The y-axis associated with the grid.
gridaxistype	The grid is aligned with the tick marks of this axis. The grid is parallel to the other axis. Use one of the axis constants, X_AXIS or Y_AXIS.

gridtype	Specifies if the grid aligns with the major tick marks (GRID_MAJOR), the minor tick marks (GRID_MINOR) or the major and minor tick marks (GRID_ALL) of the reference axis.

Other grid properties are associated with the line properties used to draw the grid. The default values of the grid use a black dotted line of thickness 1.0. Change the default values using the **GraphObj** methods below.

setColor method

```
public setColor( rgbcolor: number);
```

setLineWidth method

```
public setLineWidth( linewidth: number);
```

setLineStyle method

```
Public setLineStyle(linestyle: number)
```

rgbcolor	Sets the primary line color for the chart object.
linewidth	Sets the line width, in device coordinates, for the chart object.
linestyle	Sets the line style for the chart object. Use one of the ChartConstants line style constants

Grid example

[TypeScript]

```
    // Define the coordinate system
let xMin: number = -5;
let xMax: number = 15;
let yMin: number = 0;
let yMax: number = 105;

let simpleScale: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinates(xMin, yMin, xMax, yMax);
let xAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(simpleScale,
QCChartTS.ChartConstants.X_AXIS);
let yAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(simpleScale,
QCChartTS.ChartConstants.Y_AXIS);
```

```
let gridX: QCChartTS.Grid = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.X_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
// Change default grid line properties
gridX.setLineWidth(2);
gridX.setLineStyle(QCChartTS.ChartConstants.LS_DOT_1_4);
gridX.setColor(QCChartTS.ChartColor.GRAY);
let gridY: QCChartTS.Grid = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.Y_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
// Change default grid line properties
gridY.setLineWidth(1);
gridY.setLineStyle(QCChartTS.ChartConstants.LS_SOLID);
gridY.setColor(QCChartTS.ChartColor.BLACK);

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

[JavaScript]

```
      // Define the coordinate system
let xMin = -5;
let xMax = 15;
let yMin = 0;
let yMax = 105;
let simpleScale = QCChartTS.CartesianCoordinates.newCartesianCoordinates(xMin, yMin, xMax, yMax);
let xAxis = QCChartTS.LinearAxis.newLinearAxis(simpleScale, QCChartTS.ChartConstants.X_AXIS);
let yAxis = QCChartTS.LinearAxis.newLinearAxis(simpleScale, QCChartTS.ChartConstants.Y_AXIS);
let gridX = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.X_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
// Change default grid line properties
gridX.setLineWidth(2);
gridX.setLineStyle(QCChartTS.ChartConstants.LS_DOT_1_4);
gridX.setColor(QCChartTS.ChartColor.GRAY);
let gridY = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.Y_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
// Change default grid line properties
gridY.setLineWidth(1);
gridY.setLineStyle(QCChartTS.ChartConstants.LS_SOLID);
gridY.setColor(QCChartTS.ChartColor.BLACK);
//  Add the x- and y-axes to the chartVu object
chartVu.addChartObject(xAxis);
chartVu.addChartObject(yAxis);
chartVu.addChartObject(gridX);
chartVu.addChartObject(gridY);
```

Polar Grids

Class PolarGrid

```
GraphObj
   |
   +-- Grid
          |
          +-- PolarGrid
```

The **PolarGrid** class defines a grid for polar axes. The polar grid consists of two parts: the magnitude grid and the polar angle grid. The magnitude grid consists of a group of concentric circles centered on the origin and aligned with the x- and y-axis tick marks. The polar angle grid consists of a group of radial lines, aligned with the angle tick marks and starting at the origin extending to the outer polar angle circle.

PolarGrid constructors

There are two **PolarGrid** constructors.

```
public static newPolarGrid(polaraxis: PolarAxes, gridtype: number): PolarGrid

public static newPolarGridAxes(polaraxis: PolarAxes,
        gridmagtype: number, gridangletype: number): PolarGrid
```

polaraxis	The polar axes associated with the grid.
gridtype	Specifies if the magnitude and angular grid aligns with the major tick marks (GRID_MAJOR), the minor tick marks (GRID_MINOR) or the major and minor tick marks (GRID_ALL) of the reference polar axis.
gridmagtype	Specifies if the magnitude grid aligns with the major tick marks (GRID_MAJOR), the minor tick marks (GRID_MINOR) or the major and minor tick marks (GRID_ALL) of the reference polar axis.
gridangletype	Specifies if the angular grid aligns with the major tick marks (GRID_MAJOR), the minor tick marks (GRID_MINOR) or the major and minor tick marks (GRID_ALL) of the reference polar axis.

The **setLineWidth**, **setLineStyle** and **setColor** methods are used to customize the drawing properties of the lines used to draw the axes lines and tick marks.

Polar grid example

[TypeScript]

```
let polarmagnitude: number = 5;
let polarscale: QCChartTS.PolarCoordinates =
QCChartTS.PolarCoordinates.newPolarCoordinates(polarmagnitude);
let polarAxes: QCChartTS.PolarAxes = QCChartTS.PolarAxes.newPolarAxes(polarscale);
let polarGrid: QCChartTS.PolarGrid =  QCChartTS.PolarGrid.newPolarGrid(polarAxes,
QCChartTS.ChartConstants.GRID_MAJOR);
```

```
//  Add the polar axes to the chartVu object
chartVu.addChartObject(polarAxes);
chartVu.addChartObject(polarGrid);
```

[JavaScript]

```
let polarmagnitude = 5;
let polarscale = QCChartTS.PolarCoordinates.newPolarCoordinates(polarmagnitude);
let polarAxes = QCChartTS.PolarAxes.newPolarAxes(polarscale);
let polarGrid = QCChartTS.PolarGrid.newPolarGrid(polarAxes, QCChartTS.ChartConstants.GRID_MAJOR);
//  Add the polar axes to the chartVu object
chartVu.addChartObject(polarAxes);
chartVu.addChartObject(polarGrid);
```

Antenna Grids

Class AntennaGrid

```
GraphObj
   |
   +-- Grid
          |
          +-- AntennaGrid
```

The **AntennaGrid** class defines a grid for antenna axes. The antenna grid consists of two parts: the circular grid and the radial grid. The circular grid consists of a group of concentric circles centered on the origin and aligned with the y-axis tick marks. The antenna radial grid consists of a group of radial lines, aligned with the angle tick marks and starting at the origin extending to the outer edge of the antenna coordinate system.

AntennaGrid constructors

There are two **AntennaGrid** constructors.

```
public static newAntennaGrid(antennaaxis: AntennaAxes, gridtype: number): AntennaGrid

public static newAntennaGridAngleType(antennaaxis: AntennaAxes,
        gridmagtype: number, gridangletype: number): AntennaGrid
```

baseaxis	The antenna axes associated with the grid.
gridtype	Specifies if the radial and angular grid aligns with the major tick marks (GRID_MAJOR), the minor tick marks (GRID_MINOR) or the major and minor tick marks (GRID_ALL) of the reference antenna axis.
gridmagtype	Specifies if the radial grid aligns with the major tick marks (GRID_MAJOR), the minor tick marks (GRID_MINOR) or the major and minor tick marks (GRID_ALL) of the reference antenna axis.
gridangletype	Specifies if the angular grid aligns with the major tick marks (GRID_MAJOR), the minor tick marks (GRID_MINOR) or the major and minor tick marks (GRID_ALL) of the reference antenna axis.

The **setLineWidth**, **setLineStyle** and **setColor** methods are used to customize the drawing properties of the lines used to draw the axes lines and tick marks.

Antenna grid example

[TypeScript]

```
let pAntennaAxis: QCChartTS.AntennaAxes = pAntennaTransform.getCompatibleAxes();
pAntennaAxis.LineColor = QCChartTS.ChartColor.BLACK;
chartVu.addChartObject(pAntennaAxis);
let pAntennaGrid: QCChartTS.AntennaGrid = QCChartTS.AntennaGrid.newAntennaGrid(pAntennaAxis,
QCChartTS.ChartConstants.GRID_ALL);
pAntennaGrid.ChartObjAttributes =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.LIGHTBLUE, 1,
QCChartTS.ChartConstants.LS_SOLID);
chartVu.addChartObject(pAntennaGrid);
```

[JavaScript]

```
let pAntennaAxis = pAntennaTransform.getCompatibleAxes();
pAntennaAxis.LineColor = QCChartTS.ChartColor.BLACK;
chartVu.addChartObject(pAntennaAxis);
let pAntennaGrid = QCChartTS.AntennaGrid.newAntennaGrid(pAntennaAxis,
QCChartTS.ChartConstants.GRID_ALL);
pAntennaGrid.ChartObjAttributes =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.LIGHTBLUE, 1,
QCChartTS.ChartConstants.LS_SOLID);
chartVu.addChartObject(pAntennaGrid);
```

10. Simple Plot Objects

SimplePlot
- **SimpleBarPlot**
- **SimpleLineMarkerPlot**
- **SimpleLinePlot**
- **SimpleScatterPlot**
- **SimpleVersaPlot**

The **SimplePlot** class is an abstract class representing plot types that use data organized as a simple array of xy points, where there is one y for every x. Simple plot types include: line plots, scatter plots, bar graphs, and line-marker plots. When used in the simplest mode, simple plot objects use a single **ChartAttribute** object to control the plot objects color, line, and gradient styles. In terms of memory usage, this is the most efficient method. If memory is not an issue, it is also possible to assign every line segment, bar and scatter plot symbol a unique **ChartAttribute** object. Used in this mode, a single line plot can have unlimited number of multi-colored line segments. Another option labels each data point with its numeric y-value.

Example program segments presented in this documentation are not complete programs and contain uninitialized and/or undefined objects and variables. Do not attempt to copy them into your own program. Refer to the referenced examples.

Simple Line Plots

Class SimpleLinePlot

```
GraphObj
   |
   +--ChartPlot
         |
         +--SimplePlot
               |
               +--SimpleLinePlot
```

The **SimpleLinePlot** class is a concrete implementation of the **SimplePlot** class and displays simple datasets in line plot format. Data points are connected using a straight line, or a step line.

SimpleLinePlot constructor

```
public static newSimpleLinePlot(transform: PhysicalCoordinates, dataset: SimpleDataset,
attrib: ChartAttribute): SimpleLinePlot
```

```
public static newSimpleLinePlotCoords(transform: PhysicalCoordinates): SimpleLinePlot

public static newSimpleLinePlotCoordsDatasetAttrib(transform: PhysicalCoordinates, dataset:
SimpleDataset,attrib: ChartAttribute): SimpleLinePlot

public static newSimpleLinePlotCoordsAttrib(transform: PhysicalCoordinates,
attrib: ChartAttribute): SimpleLinePlot
```

transform	The coordinate system for the new **SimpleLinePlot** object.
dataset	The line plot represents the values in this dataset.
attrib	Specifies the attributes (line color, thickness and style, fill color and fill mode) for the line plot.

A **ChartAttribute** object sets the objects global line color, line thickness, line style, fill color and fill mode. Change the **ChartAttribute** object using the objects **setChartObjAttributes** method. There is also a group of methods that set individual simple plot properties: **setColor**, **setLineWidth**, and **setLineStyle**. The line step style is using the **setStepMode** method.

Individual line segments in a simple line plot object can have unique properties. Use the objects **setSegmentAttributesMode** and **setSegmentAttributes** methods.

Simple line plot example (extracted from the example program SimpleLinePlots.BuildLineFill)

[TypeScript]

```
pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.autoScaleDatasetsRoundMode(DatasetArray, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setGraphBorderDiagonal(0.15, .1, .92, 0.75);
.
.
.
let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 3,
QCChartTS.ChartConstants.LS_SOLID);
Dataset1.sortByX(true);
thePlot1 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, Dataset1, attrib1);
thePlot1.setLineStyle(QCChartTS.ChartConstants.LS_DASH_DOT);
chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.autoScaleDatasetsRoundMode(DatasetArray, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setGraphBorderDiagonal(0.15, .1, .92, 0.75);
.
.
.
let attrib1 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 3,
QCChartTS.ChartConstants.LS_SOLID);
Dataset1.sortByX(true);
```

```
thePlot1 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, Dataset1, attrib1);
thePlot1.setLineStyle(QCChartTS.ChartConstants.LS_DASH_DOT);
chartVu.addChartObject(thePlot1);
```

Simple line plot example using segment colors (extracted from the example program SimpleLinePlots.BuildLineFill)

[TypeScript]

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

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

let nnumpnts: number = 32;
let currentdate: Date = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);
let pTransform1: QCChartTS.TimeCoordinates;
let thePlot1: QCChartTS.SimpleLinePlot;
let thePlot2: QCChartTS.SimpleLinePlot;
let thePlot3: QCChartTS.SimpleLinePlot;
let Dataset1: QCChartTS.TimeSimpleDataset =
QCChartTS.TimeSimpleDataset.newTimeSimpleDatasetN("Sales", 0);
let Dataset2: QCChartTS.TimeSimpleDataset =
QCChartTS.TimeSimpleDataset.newTimeSimpleDatasetN("Expenses", 0);
let yy1: number = 100;
let yy2: number = 30;
let i: number = 0;
Dataset1.addTimeDataPointDateX(currentdate, yy1);
Dataset2.addTimeDataPointDateX(currentdate, yy2);
QCChartTS.ChartCalendar.add(currentdate, QCChartTS.ChartConstants.MONTH, 3);
for (i = 1; (i < nnumpnts); i++) {
    yy1 = yy1 + (5 + i) * (0.75 - QCChartTS.ChartSupport.getRandomDouble());
    yy2 = yy2 + (15 + i) * (0.95 - QCChartTS.ChartSupport.getRandomDouble());
    Dataset1.addTimeDataPointDateX(currentdate, yy1);
    Dataset2.addTimeDataPointDateX(currentdate, yy2);
    QCChartTS.ChartCalendar.add(currentdate, QCChartTS.ChartConstants.MONTH, 3);
}

Dataset1.deleteDataPoint((nnumpnts / 2));
nnumpnts = Dataset2.deleteDataPoint((nnumpnts / 2));
let Dataset3: QCChartTS.TimeSimpleDataset = <QCChartTS.TimeSimpleDataset>(Dataset2.clone());
theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
Dataset3.combineDataset(Dataset1, QCChartTS.ChartConstants.COMBINE_DATASET_Y,
QCChartTS.ChartConstants.COMBINE_DATASET_SUBTRACT);
let DatasetArray: QCChartTS.TimeSimpleDataset[] = [Dataset1, Dataset2, Dataset3]

pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.autoScaleDatasetsRoundMode(DatasetArray, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setGraphBorderDiagonal(0.15, .1, .92, 0.75);
let background: QCChartTS.Background =
QCChartTS.Background.newBackgroundCoordsTypeColorGradient(pTransform1,
QCChartTS.ChartConstants.GRAPH_BACKGROUND,
    QCChartTS.ChartColor.fromRgb(100, 50, 255), QCChartTS.ChartColor.fromRgb(40, 25, 120),
QCChartTS.ChartConstants.Y_AXIS);
chartVu.addChartObject(background);
let plotbackground: QCChartTS.Background = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.PLOT_BACKGROUND, QCChartTS.ChartColor.BLACK);
chartVu.addChartObject(plotbackground);
let xAxis: QCChartTS.TimeAxis = QCChartTS.TimeAxis.newTimeAxis(pTransform1);
xAxis.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(xAxis);
let yAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxisCoordsType(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
yAxis.setColor(QCChartTS.ChartColor.WHITE);
```

```
chartVu.addChartObject(yAxis);
let yAxis2: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxisCoordsType(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
yAxis2.setAxisIntercept(xAxis.getAxisMax());
yAxis2.setAxisTickDir(QCChartTS.ChartConstants.AXIS_MAX);
yAxis2.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(yAxis2);
let xAxisLab: QCChartTS.TimeAxisLabels = QCChartTS.TimeAxisLabels.newTimeAxisLabels(xAxis);
xAxisLab.setAxisLabelsFormat(QCChartTS.ChartConstants.TIMEDATEFORMAT_Y2000);
xAxisLab.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(xAxisLab);
let yAxisLab: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(yAxis);
yAxisLab.setColor(QCChartTS.ChartColor.WHITE);
yAxisLab.setAxisLabelsFormat(QCChartTS.ChartConstants.CURRENCYFORMAT);
chartVu.addChartObject(yAxisLab);
let yaxistitle: QCChartTS.AxisTitle = QCChartTS.AxisTitle.newAxisTitle(yAxis, theFont, "Millions
$");
yaxistitle.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(yaxistitle);
let xgrid: QCChartTS.Grid = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.X_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
xgrid.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(xgrid);
let ygrid: QCChartTS.Grid = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.Y_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
ygrid.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(ygrid);
let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 3,
QCChartTS.ChartConstants.LS_SOLID);
Dataset1.sortByX(true);
thePlot1 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, Dataset1, attrib1);
thePlot1.setLineStyle(QCChartTS.ChartConstants.LS_DASH_DOT);
chartVu.addChartObject(thePlot1);
let attrib2: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.YELLOW, 3,
QCChartTS.ChartConstants.LS_SOLID);
Dataset2.sortByX(true);
thePlot2 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, Dataset2, attrib2);
chartVu.addChartObject(thePlot2);
let transparentRed: number = QCChartTS.ChartColor.fromArgb(200, 255, 0, 0);
let transparentGreen: number = QCChartTS.ChartColor.fromArgb(200, 0, 255, 0);
let lossAttrib: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(transparentRed, 1, QCChartTS.ChartConstants.LS_SOLID,
transparentRed);
let profitAttrib: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(transparentGreen, 1,
QCChartTS.ChartConstants.LS_SOLID, transparentGreen);
profitAttrib.setFillFlag(true);
lossAttrib.setFillFlag(true);
profitAttrib.setLineFlag(false);
lossAttrib.setLineFlag(false);
//  Must call the linePlot (or similar function) before setting segment
//  attributes so that it know size of segment buffer to allocate.
thePlot3 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, Dataset3, profitAttrib);
let yValues: number[] = Dataset3.getYData();
thePlot3.setSegmentAttributesMode(true);
for (i = 0; i < Dataset3.getNumberDatapoints(); i++) {
    if ((yValues[i] > 0)) {
  thePlot3.setSegmentAttributes(i, profitAttrib);
    }
    else {
  thePlot3.setSegmentAttributes(i, lossAttrib);
    }

}

chartVu.addChartObject(thePlot3);
```

[JavaScript]

```
let htmlcanvas = document.getElementById(canvasid);
let chartVu = QCChartTS.ChartView.newChartView(htmlcanvas);
if (!chartVu)
    return;
chartVu.setPreferredSize(800, 600);
let theFont;
let nnumpnts = 32;
let currentdate = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);
let pTransform1;
let thePlot1;
let thePlot2;
let thePlot3;
let Dataset1 = QCChartTS.TimeSimpleDataset.newTimeSimpleDatasetN("Sales", 0);
let Dataset2 = QCChartTS.TimeSimpleDataset.newTimeSimpleDatasetN("Expenses", 0);
let yy1 = 100;
let yy2 = 30;
let i = 0;
Dataset1.addTimeDataPointDateX(currentdate, yy1);
Dataset2.addTimeDataPointDateX(currentdate, yy2);
QCChartTS.ChartCalendar.add(currentdate, QCChartTS.ChartConstants.MONTH, 3);
for (i = 1; (i < nnumpnts); i++) {
    yy1 = yy1 + (5 + i) * (0.75 - QCChartTS.ChartSupport.getRandomDouble());
    yy2 = yy2 + (15 + i) * (0.95 - QCChartTS.ChartSupport.getRandomDouble());
    Dataset1.addTimeDataPointDateX(currentdate, yy1);
    Dataset2.addTimeDataPointDateX(currentdate, yy2);
    QCChartTS.ChartCalendar.add(currentdate, QCChartTS.ChartConstants.MONTH, 3);
}
Dataset1.deleteDataPoint((nnumpnts / 2));
nnumpnts = Dataset2.deleteDataPoint((nnumpnts / 2));
let Dataset3 = (Dataset2.clone());
theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
Dataset3.combineDataset(Dataset1, QCChartTS.ChartConstants.COMBINE_DATASET_Y,
QCChartTS.ChartConstants.COMBINE_DATASET_SUBTRACT);
let DatasetArray = [Dataset1, Dataset2, Dataset3];
pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.autoScaleDatasetsRoundMode(DatasetArray, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setGraphBorderDiagonal(0.15, .1, .92, 0.75);
let background = QCChartTS.Background.newBackgroundCoordsTypeColorGradient(pTransform1,
QCChartTS.ChartConstants.GRAPH_BACKGROUND, QCChartTS.ChartColor.fromRgb(100, 50, 255),
QCChartTS.ChartColor.fromRgb(40, 25, 120), QCChartTS.ChartConstants.Y_AXIS);
chartVu.addChartObject(background);
let plotbackground = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.PLOT_BACKGROUND, QCChartTS.ChartColor.BLACK);
chartVu.addChartObject(plotbackground);
let xAxis = QCChartTS.TimeAxis.newTimeAxis(pTransform1);
xAxis.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(xAxis);
let yAxis = QCChartTS.LinearAxis.newLinearAxisCoordsType(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
yAxis.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(yAxis);
let yAxis2 = QCChartTS.LinearAxis.newLinearAxisCoordsType(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
yAxis2.setAxisIntercept(xAxis.getAxisMax());
yAxis2.setAxisTickDir(QCChartTS.ChartConstants.AXIS_MAX);
yAxis2.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(yAxis2);
let xAxisLab = QCChartTS.TimeAxisLabels.newTimeAxisLabels(xAxis);
xAxisLab.setAxisLabelsFormat(QCChartTS.ChartConstants.TIMEDATEFORMAT_Y2000);
xAxisLab.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(xAxisLab);
let yAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(yAxis);
yAxisLab.setColor(QCChartTS.ChartColor.WHITE);
yAxisLab.setAxisLabelsFormat(QCChartTS.ChartConstants.CURRENCYFORMAT);
chartVu.addChartObject(yAxisLab);
let yaxistitle = QCChartTS.AxisTitle.newAxisTitle(yAxis, theFont, "Millions $");
yaxistitle.setColor(QCChartTS.ChartColor.WHITE);
```

```
chartVu.addChartObject(yaxistitle);
let xgrid = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.X_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
xgrid.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(xgrid);
let ygrid = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.Y_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
ygrid.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(ygrid);
let attrib1 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 3,
QCChartTS.ChartConstants.LS_SOLID);
Dataset1.sortByX(true);
thePlot1 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, Dataset1, attrib1);
thePlot1.setLineStyle(QCChartTS.ChartConstants.LS_DASH_DOT);
chartVu.addChartObject(thePlot1);
let attrib2 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.YELLOW, 3,
QCChartTS.ChartConstants.LS_SOLID);
Dataset2.sortByX(true);
thePlot2 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, Dataset2, attrib2);
chartVu.addChartObject(thePlot2);
let transparentRed = QCChartTS.ChartColor.fromArgb(200, 255, 0, 0);
let transparentGreen = QCChartTS.ChartColor.fromArgb(200, 0, 255, 0);
let lossAttrib = QCChartTS.ChartAttribute.newChartAttribute4(transparentRed, 1,
QCChartTS.ChartConstants.LS_SOLID, transparentRed);
let profitAttrib = QCChartTS.ChartAttribute.newChartAttribute4(transparentGreen, 1,
QCChartTS.ChartConstants.LS_SOLID, transparentGreen);
profitAttrib.setFillFlag(true);
lossAttrib.setFillFlag(true);
profitAttrib.setLineFlag(false);
lossAttrib.setLineFlag(false);
//  Must call the linePlot (or similar function) before setting segment
//  attributes so that it know size of segment buffer to allocate.
thePlot3 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, Dataset3, profitAttrib);
let yValues = Dataset3.getYData();
thePlot3.setSegmentAttributesMode(true);
for (i = 0; i < Dataset3.getNumberDatapoints(); i++) {
    if ((yValues[i] > 0)) {
thePlot3.setSegmentAttributes(i, profitAttrib);
    }
    else {
thePlot3.setSegmentAttributes(i, lossAttrib);
    }
}
chartVu.addChartObject(thePlot3);
```

Simple Bar Plots

Class SimpleBarPlot

```
GraphObj
   |
   +--ChartPlot
         |
         +--SimplePlot
               |
               +--SimpleBarPlot
```

The **SimpleBarPlot** class is a concrete implementation of the **SimplePlot** class and displays data in a bar format. Individual bars, the maximum value of which corresponds to the y-values of the dataset, display justified with respect to the x-values.

SimpleBarPlot constructor

```
public static newSimpleBarPlotCoords(transform: PhysicalCoordinates): SimpleBarPlot

public static newSimpleBarPlot(transform: PhysicalCoordinates,
        dataset: SimpleDataset, barwidth: number, barbase: number,
        attrib: ChartAttribute, barjust: number): SimpleBarPlot
```

transform	The coordinate system for the new **SimpleBarPlot** object.
dataset	The bar plot represents the values in this dataset.
barwidth	The width of the bars in physical coordinates.
barbase	The base value for bars in physical coordinates.
attrib	Specifies the attributes (line color and fill color) of the bars.
barjust	Specifies the justification with respect to the independent data value. Use one of the justification constants: JUSTIFY_MIN, JUSTIFY_CENTER, JUSTIFY_MAX.

A **ChartAttribute** object sets the objects global line color, line thickness, line style, fill color and fill mode. Change the **ChartAttribute** object using the objects **setChartObjAttributes** method. The simple bar plot **setColor** method can be used to change the bar color.

Individual bars in a simple bar plot object can have unique properties. Use the objects **setSegmentAttributesMode** and **setSegmentAttributes** methods.

Simple bar plot example (extracted from the example program Bargraphs.BuildSimpleBars)

[TypeScript]

```
let Dataset1: QCChartTS.TimeSimpleDataset =
QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Actual Sales", x1, y1);
let pTransform1: QCChartTS.TimeCoordinates = new QCChartTS.TimeCoordinates();
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setScaleStartY(0);
pTransform1.setTimeScaleStart(new Date(1997, QCChartTS.ChartConstants.JULY, 1));
pTransform1.setGraphBorderDiagonal(0.15, 0.15, 0.9, 0.8);
```

```
.
.
.
let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.GREEN, 0,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);

let thePlot1: QCChartTS.SimpleBarPlot = QCChartTS.SimpleBarPlot.newSimpleBarPlot(pTransform1,
Dataset1,
    QCChartTS.ChartCalendar.getCalendarWidthValue(QCChartTS.ChartConstants.MONTH, 8), 0, attrib1,
QCChartTS.ChartConstants.JUSTIFY_CENTER);

chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
var Dataset1 = QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Actual Sales", x1, y1);
var pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setScaleStartY(0);
pTransform1.setTimeScaleStart(new Date(1997, QCChartTS.ChartConstants.JULY, 1));
pTransform1.setGraphBorderDiagonal(0.15, 0.15, 0.9, 0.8);
.
.
.

var attrib1 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.GREEN, 0,
QCChartTS.ChartConstants.LS_DASH_4_2, QCChartTS.ChartColor.GREEN);

var thePlot1 = QCChartTS.SimpleBarPlot.newSimpleBarPlot(pTransform1, Dataset1,
QCChartTS.ChartCalendar.getCalendarWidthValue(QCChartTS.ChartConstants.MONTH, 8), 0, attrib1,
QCChartTS.ChartConstants.JUSTIFY_CENTER);

chartVu.addChartObject(thePlot1);
```

* Note how the **ChartCalendar.getCalendarWidthValue** method calculates the width of the bars as a function of time, in this case a width of 8 months.

Simple bar plot example that displays numeric data values (extracted from the example program Bargraphs.BuildSimpleBars)

[TypeScript]

```
.
.
.
let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.GREEN, 0,
QCChartTS.ChartConstants.LS_DASH_4_2, QCChartTS.ChartColor.GREEN);
let thePlot1: QCChartTS.SimpleBarPlot = QCChartTS.SimpleBarPlot.newSimpleBarPlot(pTransform1,
Dataset1,
QCChartTS.ChartCalendar.getCalendarWidthValue(QCChartTS.ChartConstants.MONTH, 8), 0, attrib1,
QCChartTS.ChartConstants.JUSTIFY_CENTER);
let bardatavalue: QCChartTS.NumericLabel = thePlot1.getPlotLabelTemplate();
bardatavalue.setTextFont(theFont);
bardatavalue.setNumericFormat(QCChartTS.ChartConstants.CURRENCYFORMAT);
bardatavalue.setDecimalPos(0);
bardatavalue.setColor(QCChartTS.ChartColor.WHITE);
thePlot1.setPlotLabelTemplate(bardatavalue);
thePlot1.setShowDatapointValue(true);
```

```
chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
.
.
.
var attrib1 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.GREEN, 0,
QCChartTS.ChartConstants.LS_DASH_4_2,
var thePlot1 = QCChartTS.SimpleBarPlot.newSimpleBarPlot(pTransform1, Dataset1,
QCChartTS.ChartCalendar.getCalendarWidthValue(QCChartTS.ChartConstants.MONTH, 8), 0, attrib1,
QCChartTS.ChartConstants.JUSTIFY_CENTER);
var bardatavalue = thePlot1.getPlotLabelTemplate();
bardatavalue.setTextFont(theFont);
bardatavalue.setNumericFormat(QCChartTS.ChartConstants.CURRENCYFORMAT);
bardatavalue.setDecimalPos(0);
bardatavalue.setColor(QCChartTS.ChartColor.WHITE);
thePlot1.setPlotLabelTemplate(bardatavalue);
thePlot1.setShowDatapointValue(true);
chartVu.addChartObject(thePlot1);
```

Simple Scatter Plots

Class SimpleScatterPlot

```
GraphObj
    |
    +--ChartPlot
           |
           +--SimplePlot
                  |
                  +--SimpleScatterPlot
```

The **SimpleScatterPlot** class is a concrete implementation of the **SimplePlot** class and displays simple datasets in scatter plot format where each data point is a symbol.

SimpleScatterPlot constructor

```
public static newSimpleScatterPlotCoords(transform: PhysicalCoordinates): SimpleScatterPlot

public static newSimpleScatterPlot(transform: PhysicalCoordinates,
        dataset: SimpleDataset, symtype: number,
        attrib: ChartAttribute): SimpleScatterPlot
```

transform	The coordinate system for the new **SimpleScatterPlot** object.
dataset	The scatter plot represents the values in this dataset.

symtype	The symbol used in the scatter plot. Use one of the scatter plot symbol constants: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE.
attrib	Specifies the attributes (size, line and fill color) for the scatter plot.

A **ChartAttribute** object sets the objects global outline and fill attributes. Change the simple plot objects **ChartAttribute** object using the objects **setChartObjAttributes** method. For a simple color change of the scatter plot symbol, use the scatter plot objects **setColor** method.

Should you need additional symbols, create your own. Any **GPath** object can be used as a symbol. The coordinates of the symbol should assume that 1.0 is the standard symbol size with a symbol center at the relative coordinates (0.5, 0.5). The example below demonstrates how to create a diamond symbol.

```
public getDiamondShape(): GPath {
  let result: GPath = new GPath();
  result.moveTo(0.5, 0.0);
  result.lineTo(0.0, 0.5);
  result.lineTo(0.5, 1.0);
  result.lineTo(1.0, 0.5);
  result.close();
  return result;
}
```

Set the custom symbol using **SimpleScatterPlot.setCustomScatterPlotSymbol** method after the **SimpleScatterPlot** object is created.

Individual scatter plot symbols in a scatter plot object can have unique properties. Use the objects **setSegmentAttributesMode** and **setSegmentAttributes** methods.

Simple scatter plot example (extracted from the example program ScatterPlots.BuildSimpleScatter)

[TypeScript]

```
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 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);
```

[JavaScript]

```
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 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);
```

Simple scatter plot example that uses setSegmentAttributesMode to change the size and color of individual scatter plot symbols in the plot (extracted from the example program ScatterPlots.BuildScatterPoints)

[TypeScript]

```
.
.
.
let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID);
attrib1.setFillColor(QCChartTS.ChartColor.BLUE);
attrib1.setLineFlag(true);
attrib1.setSymbolSize(10);
let thePlot1: QCChartTS.SimpleScatterPlot =
QCChartTS.SimpleScatterPlot.newSimpleScatterPlot(pTransform1, Dataset1,
QCChartTS.ChartConstants.SQUARE, attrib1);
// thePlot1.setCustomScatterPlotSymbol(QCChartTS.Rectangle2D.Double(0,0,1,1));
thePlot1.setSegmentAttributesMode(true);
let segmentAttrib: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.RED, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
segmentAttrib.setSymbolSize(20);
thePlot1.setSegmentAttributes(8, segmentAttrib);
thePlot1.setSegmentAttributes(9, segmentAttrib);
thePlot1.setSegmentAttributes(10, segmentAttrib);
thePlot1.setSegmentAttributes(11, segmentAttrib);
chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
.
.
.
let attrib1 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID);
attrib1.setFillColor(QCChartTS.ChartColor.BLUE);
attrib1.setLineFlag(true);
attrib1.setSymbolSize(10);
let thePlot1 = QCChartTS.SimpleScatterPlot.newSimpleScatterPlot(pTransform1, Dataset1,
QCChartTS.ChartConstants.SQUARE, attrib1);
// thePlot1.setCustomScatterPlotSymbol(QCChartTS.Rectangle2D.Double(0,0,1,1));
thePlot1.setSegmentAttributesMode(true);
let segmentAttrib = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.RED, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
segmentAttrib.setSymbolSize(20);
thePlot1.setSegmentAttributes(8, segmentAttrib);
thePlot1.setSegmentAttributes(9, segmentAttrib);
thePlot1.setSegmentAttributes(10, segmentAttrib);
thePlot1.setSegmentAttributes(11, segmentAttrib);
chartVu.addChartObject(thePlot1);
```

Simple Line Marker Plots

Class SimpleLineMarkerPlot

```
GraphObj
   |
   +--ChartPlot
         |
         +--SimplePlot
               |
               +-- SimpleLineMarkerPlot
```

The **SimpleLineMarkerPlot** class is a concrete implementation of the **SimplePlot** class and displays simple datasets in a line plot format where scatter plot symbols highlight individual data points.

SimpleLineMarkerPlot constructor

```
public static newSimpleLineMarkerPlotCoords(transform: PhysicalCoordinates): SimpleLineMarkerPlot

public static newSimpleLineMarkerPlot(transform: PhysicalCoordinates,
   dataset: SimpleDataset, symtype: number, lineattrib: ChartAttribute,
   symbolattrib: ChartAttribute, nsymbolstart: number, nsymbolskip: number): SimpleLineMarkerPlot
```

transform	The coordinate system for the new **SimpleLineMarkerPlot** object.
dataset	The line marker plot represents the values in this dataset.
symtype	The symbol used in the line marker plot. Use one of the scatter plot symbol constants: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE.
lineattrib	Specifies the attributes (line color and line style) for the line part of the line marker plot.
symbolattrib	Specifies the attributes (line and fill color) for the symbol part of the line marker plot.
nsymbolstart	Specifies the starting index for symbols in the line marker plot.
nsymbolskip	Specifies the skip factor for placing symbols in the line marker plot.

An **ChartAttribute** object sets the objects global line color, line width and line style attributes. Change the **ChartAttribute** object using the objects **setChartObjAttributes** method. Use the objects **setSymbolAttributes** to change the attributes of the marker symbol.

Should you need additional symbols, create your own. Any **GPath** object can be used as a symbol. The coordinates of the symbol should assume that 1.0 is the standard symbol size with a symbol center at the relative coordinates (0.5, 0.5). See the example in the discussion of the **SimpleScatterPlot** class.

Individual line segments in a line marker plot object can have unique properties. Use the objects **setSegmentAttributesMode** and **setSegmentAttributes** methods. If this option is used, the line and fill properties of the lines and the marker symbols will be the same.

Simple line marker plot example (extracted from the example program ScatterPlots.BuildLabeledDatapoints)

[TypeScript]

```
let Dataset1: QCChartTS.SimpleDataset = QCChartTS.SimpleDataset.newSimpleDataset("First", x1,
y1);
let pTransform1: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setGraphBorderDiagonal(0.15, .15, .90, 0.7);
.
.
.
let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID);
let attrib2: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.RED, 1,
QCChartTS.ChartConstants.LS_SOLID);
attrib2.setFillColor(QCChartTS.ChartColor.RED);
attrib2.setFillFlag(true);
attrib2.setSymbolSize(15);
let thePlot1: QCChartTS.SimpleLineMarkerPlot =
QCChartTS.SimpleLineMarkerPlot.newSimpleLineMarkerPlot(pTransform1, Dataset1,
QCChartTS.ChartConstants.SQUARE, attrib1, attrib2, 0, 1);

chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
let Dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("First", x1, y1);
let pTransform1 =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setGraphBorderDiagonal(0.15, .15, .90, 0.7);
.
.
.
let attrib1 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID);
let attrib2 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.RED, 1,
QCChartTS.ChartConstants.LS_SOLID);
attrib2.setFillColor(QCChartTS.ChartColor.RED);
attrib2.setFillFlag(true);
attrib2.setSymbolSize(15);
let thePlot1 = QCChartTS.SimpleLineMarkerPlot.newSimpleLineMarkerPlot(pTransform1, Dataset1,
QCChartTS.ChartConstants.SQUARE, attrib1, attrib2, 0, 1);

chartVu.addChartObject(thePlot1);
```

Add the following lines to the program segment above to add data point labeling to the line marker plot.

[TypeScript]

```
thePlot1.setShowDatapointValue(true);
let modellabel: QCChartTS.NumericLabel = new QCChartTS.NumericLabel();
modellabel.setXJust(QCChartTS.ChartConstants.JUSTIFY_CENTER);
modellabel.setYJust(QCChartTS.ChartConstants.JUSTIFY_MIN);
let modellabelfont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Arial",
QCChartTS.ChartFont.REGULAR, 12);
```

```
modellabel.setTextFont(modellabelfont);
modellabel.setTextNudge(0, -5);
thePlot1.setPlotLabelTemplate(modellabel);
```

[JavaScript]

```
thePlot1.setShowDatapointValue(true);
let modellabel = new QCChartTS.NumericLabel();
modellabel.setXJust(QCChartTS.ChartConstants.JUSTIFY_CENTER);
modellabel.setYJust(QCChartTS.ChartConstants.JUSTIFY_MIN);
let modellabelfont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.REGULAR, 12);
modellabel.setTextFont(modellabelfont);
modellabel.setTextNudge(0, -5);
thePlot1.setPlotLabelTemplate(modellabel);
```

Simple Versa Plots

Class SimpleVersaPlot

```
GraphObj
        |
        +--ChartPlot
                |
                +--SimplePlot
                        |
                        +-- SimpleLineMarkerPlot
                                |
                                +-- SimpleVersaPlot
```

The **SimpleVersaPlot** is a plottype that can be any of the four simple plot types: LINE_MARKER_PLOT, LINE_PLOT, BAR_PLOT, SCATTER_PLOT. It is used when you want to be able to change from one plot type to another, without deleting the instance of the old plot object and creating an instance of the new.

SimpleLineMarkerPlot constructor

```
public static newSimpleVersaPlotCoords(transform: PhysicalCoordinates): SimpleVersaPlot

public static newSimpleVersaPlotCoordsDatasetSymbolNumLineAttribSymbolAttribBarWidth(transform:
PhysicalCoordinates, dataset: SimpleDataset,symtype: number, lineattrib:
ChartAttribute,symbolattrib: ChartAttribute, barwidth: number): SimpleVersaPlot
```

```
public static newSimpleVersaPlotCoordsDatasetBarWidthBarBaseAttribBarJust(transform:
PhysicalCoordinates, dataset: SimpleDataset, barwidth: number, barbase: number, attrib:
ChartAttribute, barjust: number): SimpleVersaPlot

public static SimpleVersaPlotCoordinatesDatasetSymbolNumAttrib(transform: PhysicalCoordinates,
dataset: SimpleDataset, symtype: number, symbolattrib: ChartAttribute): SimpleVersaPlot

public static SimpleVersaPlotCoordinatesDatasetAttrib(transform: PhysicalCoordinates, dataset:
SimpleDataset, lineattrib: ChartAttribute): SimpleVersaPlot
```

transform	The coordinate system for the new **SimpleVersaPlot** object.
dataset	The versa plot represents the values in this dataset.
symtype	The symbol used in the line marker plot. Use one of the scatter plot symbol constants: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE.
lineattrib	Specifies the attributes (line color and line style) for the line part of the line marker plot.
symbolattrib	Specifies the attributes (line and fill color) for the symbol part of the line marker plot.
nsymbolstart	Specifies the starting index for symbols in the line marker plot.
nsymbolskip	Specifies the skip factor for placing symbols in the line marker plot.
barwidth	Specifies the width of the bar.

A **ChartAttribute** object sets the objects global line color, line width and line style attributes. Change the **ChartAttribute** object using the objects **setChartObjAttributes** method. Use the objects **setSymbolAttributes** to change the attributes of the marker symbol.

Change the plot type using the **PlotType** property. Use one of the plot type constants: LINE_MARKER_PLOT, LINE_PLOT, BAR_PLOT, SCATTER_PLOT.

```
thePlot1.PlotType = QCChartTS.ChartConstants.SCATTER_PLOT;
```

Simple versa-plot example (extracted from the example program NewDemosRev2.BuildSimpleVersaChart)

[TypeScript]

```
ChartAttribute attrib1 = new ChartAttribute(QCChartTS.ChartColor.GREEN, 0,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
Color[] barcolors = [ QCChartTS.ChartColor.RED, QCChartTS.ChartColor.ORANGE,
QCChartTS.ChartColor.YELLOW, QCChartTS.ChartColor.WHITE ];
```

```
double[] barbreakpoints = [ 0.0, 80, 160, 240 ];
int gradmode = ChartGradient.GRADIENT_MAPTO_PLOT_PHYSICAL_COORDINATES;
ChartGradient cg = new ChartGradient(pTransform1, gradmode,
    barcolors, barbreakpoints, -90);
attrib1.Gradient = cg;

attrib1.setFillFlag(true);
thePlot1 = new SimpleVersaPlot(pTransform1, Dataset1,
   Date.getCalendarWidthValue(ChartObj.MONTH, 8), 0.0,
   attrib1, ChartObj.JUSTIFY_CENTER);
NumericLabel bardatavalue = thePlot1.getPlotLabelTemplate();
bardatavalue.setTextFont(theFont);
bardatavalue.setNumericFormat(ChartObj.CURRENCYFORMAT);
bardatavalue.setDecimalPos(0);
bardatavalue.setColor(QCChartTS.ChartColor.WHITE);
thePlot1.setPlotLabelTemplate(bardatavalue);
thePlot1.setShowDatapointValue(true);
chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
var attrib1 As New ChartAttribute(QCChartTS.ChartColor.GREEN, 0,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN)
var barcolors As Color() = [QCChartTS.ChartColor.RED, QCChartTS.ChartColor.ORANGE,
QCChartTS.ChartColor.YELLOW, QCChartTS.ChartColor.WHITE}

var barbreakpoints() = [0.0R, 80, 160, 240}
var gradmode  = ChartGradient.GRADIENT_MAPTO_PLOT_PHYSICAL_COORDINATES
var cg As New ChartGradient(pTransform1, gradmode, barcolors, barbreakpoints, -90)
attrib1.Gradient = cg

attrib1.setFillFlag(True)
thePlot1 = New SimpleVersaPlot(pTransform1, Dataset1, _
    ChartCalendar.getCalendarWidthValue(ChartObj.MONTH, 8), 0.0R, attrib1, _
    ChartObj.JUSTIFY_CENTER)
 var bardatavalue As NumericLabel = thePlot1.getPlotLabelTemplate()
 bardatavalue.setTextFont(theFont)
 bardatavalue.setNumericFormat(ChartObj.CURRENCYFORMAT)
 bardatavalue.setDecimalPos(0)
 bardatavalue.setColor(QCChartTS.ChartColor.WHITE)
 thePlot1.setPlotLabelTemplate(bardatavalue)
 thePlot1.setShowDatapointValue(True)
 chartVu.addChartObject(thePlot1)
```

11. Group Plot Objects

GroupPlot
- **BubblePlot**
- **CandlestickPlot**
- **CellPlot**
- **ErrorBarPlot**
- **FloatingBarPlot**
- **GroupBarPlotChartPlot**
- **HistogramPlot**
- **LineGapPlot**
- **MultiLinePlot**
- **OHLCPlot**
- **StackedBarPlot**
- **StackedLinePlot**

The **GroupPlot** class is an abstract class representing plot types that use data organized as arrays of x- and y-values, where there is one or more y-value for each x-value. Group plot types include: multi-line plots, stacked line plots, stacked bar plots, group bar plots, error bar plots, floating bar plots, open-high-low-close plots, candlestick plots, arrow plots, histogram plots, cell plots and bubble plots.

The number of x-values in a group plot is referred to as the number of columns, or as **numberDatapoints** and the number of y-values *for each x-value* is referred to as the number of rows, or **numberGroups**. Think of spreadsheet that looks like:

x-values	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]
y-values group #0	y[0,0]	y[0,1]	y[0,2]	y[0,3]	y[0,4]	y[0,5]
y-values group #1	y[1,0]	y[1,1]	y[1,2]	y[1,3]	y[1,4]	y[1,5]
y-values group #2	y[2,0]	y[2,1]	y[2,2]	y[2,3]	y[2,4]	y[2,5]

number of x-values = **numberDatapoints** = numberColumns = 6

number of y-values for each x-value = **numberGroups** = numberRows = 3

This would be the ROW_MAJOR format if the data were stored in a CSV file.

Example program segments presented in this documentation are not complete programs and contain uninitialized and/or undefined objects and variables. Do not attempt to copy them into your own program. Refer to the referenced example program that the code is extracted from.

Box and Whisker Plots

Class BoxWhiskerPlot

```
GraphObj
   |
   +--ChartPlot
        |
        +--GroupPlot
             |
             +-- BoxWhiskerPlot
```

The **BoxWhiskerPlot** class extends the **GroupPlot** class and displays statistical data in a box and whisker format. The **BoxWhiskerPlot** class graphically represents groups of numerical data through their five-number summaries (the smallest observation, lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation). A **BoxWhiskerPlot** is unique among our chart types because the data is not represented by a 1D or 2D matrix. Instead, it consists of multiple populations, where each population can have a different number of data points. Each population is summarized by 5 statistics (the smallest observation, lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation), display graphically in a chart. Read the Wikipedia entry for more information concerning box and whisker plots: http://en.wikipedia.org/wiki/Box_plot

BoxWhiskerPlot constructor

```
public static newBoxWhiskerPlotCoords(transform: PhysicalCoordinates): BoxWhiskerPlot

public static newBoxWhiskerPlot(transform: PhysicalCoordinates, rwidth: number, attrib:
ChartAttribute): BoxWhiskerPlot
```

transform	The coordinate system for the new **BoxWhiskerPlot** object.
rwidth	The width of the candlestick box in physical coordinates.
attrib	Specifies the attributes (line color and fill color) of the candlestick lines when the close value is greater than the open value.

Once the initial **BoxWhiskerPlot** object is created, populations are added to it, one population at a time, using the a**ddPopulation** method. Each population can have a different number of data points. Once all of the population groups are added, the software will calculate the quartile data for each population in response to the a**utoBWChart** method call. Each population is summarized by a single box in the box and whisker plot.

```
public addPopulation(pop: number[], xvalue: number)

public addPopulationDoubleArray(poparray: DoubleArray, xvalue: number)
```

Parameters

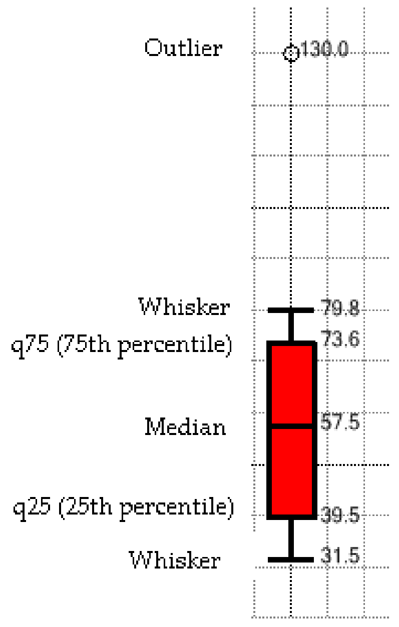
pop	The source population of y-values to add.
xvalue	The x-value of the of y-values population.

There are several variants of box and whisker plots. Select which one you want using the **BWFormat** property. Use one of the BW format constants: BW_MINMAX_WHISKER, BW_IRQ15_WHISKER_OUTLIERS, BW_IQR15_WHISKER_ALLPOINTS.

BW_MINMAX_WHISKER	Plot minimum, maximum values as whiskers, and the median, q25 and q75 values as the box.
BW_IQR15_WHISKER_OUTLIERS	Plot the minimum value within q25 - 1.5*IQR and maximum value within q75 + 1.5*IQR as whiskers, the median, q25 and q75 values as the box, and outliers as scatter plot symbols.
BW_IQR15_WHISKER_ALLPOINTS	Plot the minimum value within q25 - 1.5*IQR and maximum value within q75 + 1.5*IQR as the whiskers, the median, q25 and q75 values as the box, and plot all points as scatter plot symbols.

Where

q25	for a given population, q25 is the 25th percentile point in the population
q75	for a given population, q75 is the 75th percentile point in the population
IQR	for a given population, IQR is the value q75 - q25.



Turn on the numeric labeling of the Box and Whisker summary values (high whisker, q75, median, q25, low whisker) by setting the **BoxWhiskerPlot**. **showDatapointValue** property true. Turn on the numeric labeling of the scatter plot symbols used for outliers by setting the **BoxWhiskerPlot**. **ScatterPlot.showDatapointValue** property true. You should not combine numeric labeling with the W_IQR15_WHISKER_ALLPOINTS option, unless you are working with a very small number of data points; otherwise the labels will all overlap one another.

Box and whisker plot example (extracted from the example program NewDemosRev2.BuildBoxAndWhisker)

[TypeScript]

```
   // New York City
let NYCity: number[] = [
31.5,  33.6, 42.4, 52.5, 62.7, 71.6, 130, 79.8, 75.5, 68.2,  57.5, 47.6, 36.6];
// Houston
let Houston: number[] = [
50.4,  53.9, 60.6, 68.3, 74.5, 80.4, 5, 100, 82.6, 82.3, 78.2, 69.6, 61, 53.5];
// San Francisco
let SanFrancisco: number[] = [
48.7, 52.2, 53.3, 55.6,58.1,  76, 61.5, 62.7, 63.7, 64.5,  61, 54.8, 49.4];
// Boston
let Boston: number[] = [
32.4, 53.9, 44.6, 58.3, 64.5, 70.4, 73, 90, 72.6, 72.3, 68.2, 49.6, 41, 33.5];
// Pittsburgh
```

```
let Pittsburgh: number[] = [
41.4, 54, 24.6, 38.3, 44.5, 61.4, 63, 105, 72.6, 72.3, 68.2, 49.6, 41, 33.5];


let i: number;
let numpnts: number = NYCity.length + Houston.length + SanFrancisco.length + Boston.length +
Pittsburgh.length;
.
.
.

let defaultattrib: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
defaultattrib.setFillFlag(true);
let fillattrib: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
fillattrib.setFillFlag(true);
this.thePlot1 = QCChartTS.BoxWhiskerPlot.newBoxWhiskerPlot(pTransform1, 0.25, fillattrib);
this.thePlot1.addPopulation(NYCity, 1);
this.thePlot1.addPopulation(Houston, 2);
this.thePlot1.addPopulation(SanFrancisco, 3);
this.thePlot1.addPopulation(Boston, 4);
this.thePlot1.addPopulation(Pittsburgh, 5);
this.thePlot1.BWFormat = QCChartTS.ChartConstants.BW_IQR15_WHISKER_OUTLIERS;
this.thePlot1.BarDatapointLabelPosition = QCChartTS.ChartConstants.CENTERED_BAR;
this.thePlot1.PlotLabelTemplate.TextFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans
Serif", QCChartTS.ChartFont.REGULAR, 10);
this.thePlot1.PlotLabelTemplate.DecimalPos = 1;
this.thePlot1.ShowDatapointValue = true;
//  label outliers
this.thePlot1.ScatterPlot.ShowDatapointValue = true;
this.thePlot1.autoBWChart();
chartVu.addChartObject(this.thePlot1);
```

[JavaScript]

```
// New York City
let NYCity = [
31.5,  33.6, 42.4, 52.5, 62.7, 71.6, 130, 79.8, 75.5, 68.2,  57.5, 47.6, 36.6];
// Houston
let Houston = [
50.4,  53.9, 60.6, 68.3, 74.5, 80.4, 5, 100, 82.6, 82.3, 78.2, 69.6, 61, 53.5];
// San Francisco
let SanFrancisco = [
48.7, 52.2, 53.3, 55.6,58.1,  76, 61.5, 62.7, 63.7, 64.5,  61, 54.8, 49.4];
// Boston
let Boston = [
32.4, 53.9, 44.6, 58.3, 64.5, 70.4, 73, 90, 72.6, 72.3, 68.2, 49.6, 41, 33.5];
// Pittsburgh
let Pittsburgh = [
41.4, 54, 24.6, 38.3, 44.5, 61.4, 63, 105, 72.6, 72.3, 68.2, 49.6, 41, 33.5];
.
.
.

let defaultattrib = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
defaultattrib.setFillFlag(true);
let fillattrib = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
fillattrib.setFillFlag(true);
thePlot1 = QCChartTS.BoxWhiskerPlot.newBoxWhiskerPlot(pTransform1, 0.25, fillattrib);
thePlot1.addPopulation(NYCity, 1);
thePlot1.addPopulation(Houston, 2);
```

```
thePlot1.addPopulation(SanFrancisco, 3);
thePlot1.addPopulation(Boston, 4);
thePlot1.addPopulation(Pittsburgh, 5);
thePlot1.BWFormat = QCChartTS.ChartConstants.BW_IQR15_WHISKER_OUTLIERS;
thePlot1.BarDatapointLabelPosition = QCChartTS.ChartConstants.CENTERED_BAR;
thePlot1.PlotLabelTemplate.TextFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.REGULAR, 10);
thePlot1.PlotLabelTemplate.DecimalPos = 1;
thePlot1.ShowDatapointValue = true;
//  label outliers
thePlot1.ScatterPlot.ShowDatapointValue = true;
thePlot1.autoBWChart();
chartVu.addChartObject(thePlot1);
```

Bubble Plots

Class BubblePlot

GraphObj
```
   |
   +--ChartPlot
         |
         +--GroupPlot
               |
               +--BubblePlot
```

The **BubblePlot** class is a concrete implementation of the **GroupPlot** class. It displays bubble plots. A group dataset specifies the position and size of each bubble in a bubble plot. The numer of groups must be two.

BubblePlot constructor

```
public static newBubblePlotCoords(transform: PhysicalCoordinates): BubblePlot

public static newBubblePlot(transform: PhysicalCoordinates, dataset: GroupDataset,
        bubblesizetype: number, attrib: ChartAttribute): BubblePlot
```

transform	The coordinate system for the new bubble plot object.
dataset	A group dataset specifying the location and size of the bubbles in the bubble plot. The number of groups must be two. The dataset values for X and Y[0] set the position of the center of each bubble and the values for Y[1] set the size of each bubble, either the area (SIZE_BUBBLE_AREA) or the radius(SIZE_BUBBLE_RADIUS).
bubblesizetype	Sets whether the circle representing each bubble plot has a radius, or an area, proportional to the Y[1] data values in the group dataset. Set using one of the bubble plot type constants: SIZE_BUBBLE_RADIUS or SIZE_BUBBLE_AREA.

attrib	Specifies the attributes (line color and fill color) of the bubble plot circles.

An individual bubble in a bubble plot object can have unique attributes. Use the objects **setSegmentAttributesMode** and **setSegmentAttributes** methods in the manner described for **SimplePlot** objects..

Bubble plot example (extracted from the example program ScatterPlots.BuildRealGDPGrowth

[TypeScript]

```
let Dataset1: QCChartTS.GroupDataset = QCChartTS.GroupDataset.newGroupDatasetX2DY("First", x1,
y1);
Dataset1.setStackMode(QCChartTS.ChartConstants.AUTOAXES_UNSTACKED);
Dataset1.setAutoScaleNumberGroups(1);
let pTransform1: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
  //  Fudge axes to allow for radius of bubbles
pTransform1.ScaleStartX = -2;
pTransform1.ScaleStopY += 2;
pTransform1.setGraphBorderDiagonal(0.085, .11, .95, 0.8);
.
.
.

let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 0,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
let thePlot1: QCChartTS.BubblePlot = QCChartTS.BubblePlot.newBubblePlot(pTransform1, Dataset1,
QCChartTS.ChartConstants.SIZE_BUBBLE_AREA, attrib1);

chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
let Dataset1 = QCChartTS.GroupDataset.newGroupDatasetX2DY("First", x1, y1);
Dataset1.setStackMode(QCChartTS.ChartConstants.AUTOAXES_UNSTACKED);
Dataset1.setAutoScaleNumberGroups(1);
let pTransform1 =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
  //  Fudge axes to allow for radius of bubbles
pTransform1.ScaleStartX = -2;
pTransform1.ScaleStopY += 2;
pTransform1.setGraphBorderDiagonal(0.085, .11, .95, 0.8);
.
.
.

let attrib1 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 0,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
```

```
let thePlot1 = QCChartTS.BubblePlot.newBubblePlot(pTransform1, Dataset1,
QCChartTS.ChartConstants.SIZE_BUBBLE_AREA, attrib1);
thePlot1.setSegmentAttributesMode(true);
let segAttrib = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 0,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.BURLYWOOD);

chartVu.addChartObject(thePlot1);
```

*Note the use of the **GroupDataset** method **setStackMode**. This forces the auto-scale routine to look at the sum of y-values across groups, as is needed to auto-scale stacked plots. It is useful for bubble plots of type SIZE_BUBBLE_RADIUS because the y[0] value represents the y-position of the bubble, and the y[1] value the radius in physical coordinates. Adding the two for each bubble gives the maximum y-value for the scale needed to display the bubble. If SIZE_BUBBLE_AREA is used you may want to restrict the auto-scale routines to the just look at the bubble position using **setAutoScaleNumberGroups(1)**. You could then add in some fudge factor to make sure that the scale shows the entire bubble. The example under CellPlot demonstrates this.

Candlestick Plots

Class CandlestickPlot

```
GraphObj
    |
    +--ChartPlot
        |
        +--GroupPlot
            |
            +-- CandlestickPlot
```

The **CandlestickPlot** class is a concrete implementation of the **GroupPlot** class. It extends the **GroupPlot** class and displays stock market data in an open-high-low-close format common in financial technical analysis. Every item of the plot is a group of two horizontal lines representing High and Low values which are connected with a vertical line and a box representing the Open and Close values. If the Open value is greater than the Close value for a particular candlestick, the box is filled, otherwise it is unfilled. The number of groups must be four. The data in the dataset is organized in the following manner: The Y[0] values of the group dataset represent the values for Open, the Y[1] values for High, the Y[2] values for Low, and the Y[3] values for Close.

CandlestickPlot constructor

```
public static newCandlestickPlotCoords(transform: PhysicalCoordinates): CandlestickPlot

public static newCandlestickPlot(transform: PhysicalCoordinates,
        dataset: GroupDataset,
        rwidth: number,
        defaultattrib: ChartAttribute,
        fillattrib: ChartAttribute): CandlestickPlot
```

transform	The coordinate system for the new **CandlestickPlot** object.
dataset	The **CandlestickPlot** plot represents the group open-high-low-close values in this group dataset. The number of groups must be four. Orgainize the data in the following manner: The x-values of the group dataset set the x-positions of the candlestick objects. The Y[0] values of the group dataset represent the values for Open, the Y[1] values for High, the Y[2] values for Low, and the Y[3] values for Close.
rwidth	The width of the candlestick box in physical coordinates.
defaultattrib	Specifies the default attributes (line color and fill color) of the candlestick lines and box.
fillattrib	Specifies the attributes (line color and fill color) of the candlestick lines when the close value is greater than the open value.

An individual candlestick in a candlestick plot object can have unique attributes. Use the objects **setSegmentAttributesMode** and **setSegmentAttributes** methods in the manner described for **SimplePlot** objects..

Candlestick plot example (extracted from the example program FinancialExamples.BuildCandlestickChart)

[TypeScript]

```
let Dataset1: QCChartTS.TimeGroupDataset =
QCChartTS.TimeGroupDataset.newTimeGroupDatasetDateX2DY("Stock Data", xValues, stockPriceData);

let pTransform1: QCChartTS.TimeCoordinates = new QCChartTS.TimeCoordinates();
pTransform1.setWeekType(weekmode);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
pTransform1.setGraphBorderDiagonal(0.13, .15, .90, 0.8);

.
.
.
let defaultattrib: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.WHITE);
defaultattrib.setFillFlag(true);
let fillattrib: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
fillattrib.setFillFlag(true);
let thePlot1: QCChartTS.CandlestickPlot =
QCChartTS.CandlestickPlot.newCandlestickPlot(pTransform1, Dataset1,
QCChartTS.ChartCalendar.getCalendarWidthValue(QCChartTS.ChartConstants.DAY_OF_YEAR, 0.8),
defaultattrib, fillattrib);
chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
var Dataset1 = QCChartTS.TimeGroupDataset.newTimeGroupDatasetDateX2DY("Stock Data", xValues,
stockPriceData);

var pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.setWeekType(weekmode);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
pTransform1.setGraphBorderDiagonal(0.13, .15, .90, 0.8);
.
.
.
var defaultattrib = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.WHITE);
defaultattrib.setFillFlag(true);
var fillattrib = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
fillattrib.setFillFlag(true);
var thePlot1 = QCChartTS.CandlestickPlot.newCandlestickPlot(pTransform1, Dataset1,
QCChartTS.ChartCalendar.getCalendarWidthValue(QCChartTS.ChartConstants.DAY_OF_YEAR, 0.8),
defaultattrib, fillattrib);
chartVu.addChartObject(thePlot1);
```

* Note how the **ChartCalendar.getCalendarWidthValue** method calculates the width of the bars as a function of time, in this case a width of 0.8 months.

Cell Plots

Class CellPlot

```
GraphObj
   |
   +--ChartPlot
         |
         +--GroupPlot
               |
               +--CellPlot
```

The **CellPlot** class extends the **GroupPlot** class and displays cell plots. A cell plot is a collection of rectangular objects with independent positions, widths and heights, specified using the values of the associated group dataset. The number of groups must be three. The (X, Y[0]) values of the group dataset represent the xy position of the lower left corner of each cell, the Y[1] values set the width of the cell, and the Y[2] values set the height of the cell. Each cell can be filled using a color, or an image.

CellPlot constructor

```
public static newCellPlotCoords(transform: PhysicalCoordinates): CellPlot

public static newCellPlotCoordsDatasetAttrib(transform: PhysicalCoordinates,
        dataset: GroupDataset,
        attrib: ChartAttribute): CellPlot
```

transform	The coordinate system for the new **CellPlot** object.
dataset	The cell plot represents the values in this group dataset. The number of groups must be three. The (X, Y[0]) values of the group dataset represent the xy position of the lower left corner of each cell, the Y[1] values set the width of the cell, and the Y[2] values set the height of the cell.
attrib	Specifies the attributes (line color and line style) for the cell plot.

Cells can be filled with an image instead of a solid color. Use the **CellPlot.setPlotImage** method to place a HTMLImageElement object in the cells of a cell plot. One image applies to all of the cells in the cell plot.

An individual cell in the cell plot object can have unique attributes. Use the objects **setSegmentAttributesMode** and **setSegmentAttributes** methods in the manner described for **SimplePlot** objects.

Cell plot example (extracted from the example program ScatterPlots.BuildCellPlot)

[TypeScript]

```
let Dataset1: QCChartTS.GroupDataset = QCChartTS.GroupDataset.newGroupDatasetX2DY("First", x1,
y1);
Dataset1.setAutoScaleNumberGroups(1);
let pTransform1: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
let maxx: number = (pTransform1.getScaleStopX() + 20);
let maxy: number = (pTransform1.getScaleStopY() + 10);
pTransform1.setScaleStopX(maxx);
pTransform1.setScaleStopY(maxy);
//  Re-auto-scale to produce rounded axis values.
pTransform1.autoScaleRoundMode(QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
pTransform1.setGraphBorderDiagonal(0.15, .15, .90, 0.75);

.
.
.
```

```
let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID);
attrib1.setFillColor(QCChartTS.ChartColor.BLUE);
attrib1.setFillFlag(true);
let thePlot1: QCChartTS.CellPlot = QCChartTS.CellPlot.newCellPlotCoordsDatasetAttrib(pTransform1,
Dataset1, attrib1);
for (i = 0; i < numPoints; i++)
    thePlot1.setSegmentColor(i, QCChartTS.ChartColor.fromRgb((x1[i]),
(y1[0][i] * 2.0),
((y1[1][i] + y1[2][i]) * 7)));

chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
let Dataset1 = QCChartTS.GroupDataset.newGroupDatasetX2DY("First", x1, y1);
Dataset1.setAutoScaleNumberGroups(1);
let pTransform1 =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
let maxx = (pTransform1.getScaleStopX() + 20);
let maxy = (pTransform1.getScaleStopY() + 10);
pTransform1.setScaleStopX(maxx);
pTransform1.setScaleStopY(maxy);
//  Re-auto-scale to produce rounded axis values.
pTransform1.autoScaleRoundMode(QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
pTransform1.setGraphBorderDiagonal(0.15, .15, .90, 0.75);

.
.
.
let attrib1 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID);
attrib1.setFillColor(QCChartTS.ChartColor.BLUE);
attrib1.setFillFlag(true);
let thePlot1 = QCChartTS.CellPlot.newCellPlotCoordsDatasetAttrib(pTransform1, Dataset1, attrib1);
for (i = 0; i < numPoints; i++)
    thePlot1.setSegmentColor(i, QCChartTS.ChartColor.fromRgb((x1[i]),
(y1[0][i] * 2.0),
((y1[1][i] + y1[2][i]) * 7)));

chartVu.addChartObject(thePlot1);
```

*Note the use of the **GroupDataset** method **setAutoScaleNumberGroups**. This forces the auto-scale routine to look at just the first group of values, Y[0], because those are the only absolute position values. The maximum cell width and height are calculated and added to the initial scale. The auto-scale function is then rerun, producing a coordinate system that takes into account the widths and heights of the cells.

Error Bar Plots

Class ErrorBarPlot

GraphObj

|

```
+--ChartPlot
     |
     +--GroupPlot
          |
          +-- ErrorBarPlot
```

The **ErrorBarPlot** class extends the **GroupPlot** class and displays error bars. Error bars are two lines positioned around a data point to signify the statistical error associated with the data point. The number of groups must be two. The (X, Y[0]) values of the group dataset represent the xy position of the first error bar lines, the (X, Y[1]) values of the group dataset represent the xy position of the second error bar lines. The error bar lines center on the X. Connecting the error bar lines with a perpendicular line is an option.

ErrorBarPlot constructor

```
public static newErrorBarPlotCoords(transform: PhysicalCoordinates): ErrorBarPlot

public static newErrorBarPlot(transform: PhysicalCoordinates, dataset: GroupDataset,
rbarwidth: number, attrib: ChartAttribute): ErrorBarPlot
```

transform	The coordinate system for the new **ErrorBarPlot** object. The number of groups must be two.
dataset	The error bar plot represents the values in this group dataset. The number of groups must be two. The (X, Y[0]) values of the group dataset represent the xy position of the first error bar lines, the (X,Y[1]) values of the group dataset represent the xy position of the second error bar lines.
rbarwidth	The width of the error bars.
attrib	Specifies the attributes (line color and line style) for the error bars.

An individual set of error bars in an error bar plot object can have unique attributes. Use the objects **setSegmentAttributesMode** and **setSegmentAttributes** methods in the manner described for **SimplePlot** objects.

Error bar example (extracted from the example program MiscCharts.BuildErrorBarPlot)

[TypeScript]

```
let numGroups: number = 2;
let numPoints: number = 25;
let x1: number[] = new Array(numPoints);
let y1: number[] = new Array(numPoints);
let error: number[][] = QCChartTS.ChartSupport.newArray2D(numGroups, numPoints);
let i: number;
for (i = 0; i < numPoints; i++) {
x1[i] = 20 + i * 3;
y1[i] = 12 * 2 * (0.5 - QCChartTS.ChartSupport.getRandomDouble());
error[0][i] = y1[i] - 2 * QCChartTS.ChartSupport.getRandomDouble();
error[1][i] =  y1[i] + 2* 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.GroupDataset = QCChartTS.GroupDataset.newGroupDatasetX2DY("First", x1,
error);

let pTransform1: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);

pTransform1.setGraphBorderDiagonal(0.15, .15, .90, 0.85);
let background: QCChartTS.Background = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.PLOT_BACKGROUND, QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(background);
let xAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.X_AXIS);
chartVu.addChartObject(xAxis);
let yAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
chartVu.addChartObject(yAxis);
let xAxisLab: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(xAxis);
xAxisLab.setTextFont(theFont);
chartVu.addChartObject(xAxisLab);
let yAxisLab: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(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(yAxis, titleFont,
"Distance from Closest Walmart (Miles)");
chartVu.addChartObject(yaxistitle);
let xaxistitle: QCChartTS.AxisTitle = QCChartTS.AxisTitle.newAxisTitle(xAxis, titleFont, "Mean
Family Income ($1000)");
chartVu.addChartObject(xaxistitle);
let xgrid: QCChartTS.Grid = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.X_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
chartVu.addChartObject(xgrid);
let ygrid: QCChartTS.Grid = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.Y_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
chartVu.addChartObject(ygrid);


let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.RED, 1,
QCChartTS.ChartConstants.LS_SOLID);
attrib1.setFillColor(QCChartTS.ChartColor.RED);
attrib1.setFillFlag(true);
attrib1.setSymbolSize(14);
let thePlot1: QCChartTS.SimpleScatterPlot =
QCChartTS.SimpleScatterPlot.newSimpleScatterPlot(pTransform1, Dataset1,
QCChartTS.ChartConstants.STAR,attrib1);

chartVu.addChartObject(thePlot1);
```

```
let attrib2: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID);
attrib2.setFillColor(QCChartTS.ChartColor.BLUE);
attrib2.setFillFlag(true);
let width: number = 2;
let thePlot2: QCChartTS.ErrorBarPlot = QCChartTS.ErrorBarPlot.newErrorBarPlot(pTransform1,
Dataset2, width,attrib2);
chartVu.addChartObject(thePlot2);
```

[JavaScript]

```
let numGroups = 2;
let numPoints = 25;
let x1 = new Array(numPoints);
let y1 = new Array(numPoints);
let error = QCChartTS.ChartSupport.newArray2D(numGroups, numPoints);
let i;
for (i = 0; i < numPoints; i++) {
x1[i] = 20 + i * 3;
y1[i] = 12 * 2 * (0.5 - QCChartTS.ChartSupport.getRandomDouble());
error[0][i] = y1[i] - 2 * QCChartTS.ChartSupport.getRandomDouble();
error[1][i] = y1[i] + 2 * QCChartTS.ChartSupport.getRandomDouble();
}
theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
let Dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("First", x1, y1);
let Dataset2 = QCChartTS.GroupDataset.newGroupDatasetX2DY("First", x1, error);
let pTransform1 =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setGraphBorderDiagonal(0.15, .15, .90, 0.75);
let background = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.PLOT_BACKGROUND, QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(background);
let xAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.X_AXIS);
chartVu.addChartObject(xAxis);
let yAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.Y_AXIS);
chartVu.addChartObject(yAxis);
let xAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(xAxis);
xAxisLab.setTextFont(theFont);
chartVu.addChartObject(xAxisLab);
let yAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(yAxis);
yAxisLab.setTextFont(theFont);
chartVu.addChartObject(yAxisLab);
let titleFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
let yaxistitle = QCChartTS.AxisTitle.newAxisTitle(yAxis, titleFont, "Distance from Closest
Walmart (Miles)");
chartVu.addChartObject(yaxistitle);
let xaxistitle = QCChartTS.AxisTitle.newAxisTitle(xAxis, titleFont, "Mean Family Income
($1000)");
chartVu.addChartObject(xaxistitle);
let xgrid = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.X_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
chartVu.addChartObject(xgrid);
let ygrid = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.Y_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
chartVu.addChartObject(ygrid);
let attrib1 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.RED, 1,
QCChartTS.ChartConstants.LS_SOLID);
attrib1.setFillColor(QCChartTS.ChartColor.RED);
attrib1.setFillFlag(true);
attrib1.setSymbolSize(14);
```

```
let thePlot1 = QCChartTS.SimpleScatterPlot.newSimpleScatterPlot(pTransform1, Dataset1,
QCChartTS.ChartConstants.STAR, attrib1);
chartVu.addChartObject(thePlot1);
let attrib2 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID);
attrib2.setFillColor(QCChartTS.ChartColor.BLUE);
attrib2.setFillFlag(true);
let width = 2;
let thePlot2 = QCChartTS.ErrorBarPlot.newErrorBarPlot(pTransform1, Dataset2, width, attrib2);
chartVu.addChartObject(thePlot2);
```

Floating Bar Plots

Class FloatingBarPlot

```
GraphObj
   |
   +--ChartPlot
         |
         +--GroupPlot
               |
               +-- FloatingBarPlot
```

The **FloatingBarPlot** class extends the **GroupBarPlot** class and displays floating bar plots. The bars are free floating because each bar does not reference a fixed base value, as do the simple bar plots, stacked bar plots and group bar plots. The number of groups must be two. The (X, Y[0]) values of the group dataset represent the starting points of each bar, the (X, Y[1]) values of the group dataset represent the ending points of each bar. All bars in a given **FloatingBarPlot** object have the same width.

FloatingBarPlot constructor

```
public static newFloatingBarPlotCoords(transform: PhysicalCoordinates): FloatingBarPlot

public static newFloatingBarPlot(transform: PhysicalCoordinates,
        dataset: GroupDataset,
        rbarwidth: number,
        attrib: ChartAttribute,
        nbarjust: number): FloatingBarPlot
```

transform	The coordinate system for the new **FloatingBarPlot** object.
dataset	The floating bar plot represents the values in this group dataset. The number of groups must be two. The (X, Y[0]) values of the group dataset represent the starting points of each bar, the (X, Y[1]) values of the group dataset represent the ending points of each bar.
rbarwidth	The width of the floating bars in units of the independent axis.

attrib	Specifies the attributes (line and fill color) for the floating bars.
nbarjust	Specifies the justification with respect to the independent data value. Use one of the justification constants: JUSTIFY_MIN, JUSTIFY_CENTER, JUSTIFY_MAX.

An individual bar in a floating bar plot object can have unique attributes. Use the objects **setSegmentAttributesMode** and **setSegmentAttributes** methods in the manner described for **SimplePlot** objects..

Scheduling charts often use floating bars, where the starting and ending values of the bar represent the duration of some aspect of a project. The default use of the floating bar class assumes that the ends of the bars are floating point values, not date/time values. Yet the scheduling chart often uses date/time values to specify the bar ends. Since only the x-axis works with time values, the floating bars of a scheduling chart need to used in the horizontal orientation mode, set using the **FloatingBar.setBarOrient** method. Used in this mode, the x-values of the dataset position the bars with respect to the y-axis, and the y-values of the dataset position the ends of the bars with respect to the x-axis. In order to have a group dataset object that stores x-values as doubles and the y-group values as **Date** dates you must use a special **TimeGroupDataset** constructor:

```
public static newTimeGroupDatasetXDate2DY(sname: string, x: number[], y: Date[][]):
TimeGroupDataset
```

This constructor creates a new, group TimeGroupDataset object where the y-values are Date values and the x-values are floating point numbers.

sname	Specifies the name of the dataset.
x	An array that specifies the x-values of a group dataset.
y	An array that specifies the y-values of a group dataset.

If you manually scale the **TimeCoordinates** object, you can proceed as in all the other examples that use a date/time x-axis. If you need to use the auto-scaling capability, you will need to add a few additional steps.

Place the data in a **TimeGroupDataset** dataset and use it to auto-scale a **TimeCoordinates** scaling object. The only problem here is that since the y-values are the time values, the chart y-axis will end up as the time axis and the x-axis will end up the numeric axis. Call the **TimeCoordinates.swapScaleOrientation** in order to get it back to the orientation we want. This swaps the scale orientation so the x-axis is the time axis and the y-axis is the numeric axis. See the second of the two examples below for more details about how to use date/time values to specify the bar ends of a floating bar plot.

Floating bar plot example that uses numeric values as the floating bar endpoints (extracted from the example program Bargraphs.BuildFloatingBars)

[TypeScript]

```
let Dataset1: QCChartTS.GroupDataset = QCChartTS.GroupDataset.newGroupDatasetX2DY("Actual Sales",
x1, y1);
let pTransform1: QCChartTS.CartesianCoordinates = new QCChartTS.CartesianCoordinates();
pTransform1.setScaleStartX(0);
pTransform1.setScaleStartY(0);
pTransform1.setScaleStopX(12);
pTransform1.setScaleStopY(7);
pTransform1.setGraphBorderDiagonal(0.22, 0.15, 0.95, 0.7);
.
.
.
let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
attrib1.setFillFlag(true);
let thePlot1: QCChartTS.FloatingBarPlot =
QCChartTS.FloatingBarPlot.newFloatingBarPlot(pTransform1, Dataset1, 0.75, attrib1,
QCChartTS.ChartConstants.JUSTIFY_CENTER);
thePlot1.setBarOrient(QCChartTS.ChartConstants.HORIZ_DIR)

chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
var Dataset1 = QCChartTS.GroupDataset.newGroupDatasetX2DY("Actual Sales", x1, y1);
var pTransform1 = new QCChartTS.CartesianCoordinates();
pTransform1.setScaleStartX(0);
pTransform1.setScaleStartY(0);
pTransform1.setScaleStopX(12);
pTransform1.setScaleStopY(7);
pTransform1.setGraphBorderDiagonal(0.22, 0.15, 0.95, 0.7);
.
.
.
chartVu.addChartObject(xgrid);
var attrib1 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
attrib1.setFillFlag(true);
var thePlot1 = QCChartTS.FloatingBarPlot.newFloatingBarPlot(pTransform1, Dataset1, 0.75, attrib1,
QCChartTS.ChartConstants.JUSTIFY_CENTER);
thePlot1.setBarOrient(QCChartTS.ChartConstants.HORIZ_DIR);

chartVu.addChartObject(thePlot1);)
```

Floating bar plot example that uses date/time values as the floating bar endpoints (extracted from the example program CalendarData.InitializeNasaChart2.

[TypeScript]

```
let nnumpnts: number = 6;
let numgroups: number = 2;
let x1: number[] = new Array(nnumpnts);
//  x-values to hold position
//  Calendar data to hold start/stop of floating bars
let y1: Date[][] = QCChartTS.ChartSupport.newTimeArray2D(numgroups, nnumpnts);
//  Mercury
x1[0] = 6;
y1[0][0] = new Date(1960, QCChartTS.ChartConstants.JANUARY, 1);
y1[1][0] = new Date(1963, QCChartTS.ChartConstants.JANUARY, 1);
//  Gemini
x1[1] = 5;
y1[0][1] = new Date(1962, QCChartTS.ChartConstants.JANUARY, 1);
y1[1][1] = new Date(1967, QCChartTS.ChartConstants.JANUARY, 1);
//  Apollo
x1[2] = 4;
y1[0][2] = new Date(1963, QCChartTS.ChartConstants.JANUARY, 1);
y1[1][2] = new Date(1973, QCChartTS.ChartConstants.JANUARY, 1);
//  Skylab
x1[3] = 3;
y1[0][3] = new Date(1972, QCChartTS.ChartConstants.JANUARY, 1);
y1[1][3] = new Date(1974, QCChartTS.ChartConstants.JANUARY, 1);
//  Space Shuttle
x1[4] = 2;
y1[0][4] = new Date(1977, QCChartTS.ChartConstants.JANUARY, 1);
y1[1][4] = new Date(2005, QCChartTS.ChartConstants.DECEMBER, 31);
//  International Space Station
x1[5] = 1;
y1[0][5] = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);
y1[1][5] = new Date(2005, QCChartTS.ChartConstants.DECEMBER, 31);
theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.REGULAR, 10);
//  Time group dataset for floating bars
let Dataset1: QCChartTS.TimeGroupDataset =
QCChartTS.TimeGroupDataset.newTimeGroupDatasetXDate2DY("Space programs", x1, y1);
let pTransform1: QCChartTS.TimeCoordinates = new QCChartTS.TimeCoordinates();
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
//  Y values (Date values) of dataset will actually
//  be plotted against x-coordinates
//  so need to swap the chart coordinate system x and y scale
pTransform1.swapScaleOrientation();
pTransform1.ScaleStartY = 0;
pTransform1.setGraphBorderDiagonal(0.15, 0.6, 0.95, 0.9);

.
.
.

//  Create floating bar plot
let thePlot1: QCChartTS.FloatingBarPlot =
QCChartTS.FloatingBarPlot.newFloatingBarPlotCoords(pTransform1);
let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
attrib1.FillFlag = true;
thePlot1.initFloatingBarPlot(Dataset1, 0.75, attrib1, QCChartTS.ChartConstants.JUSTIFY_CENTER);
//  Use horizontal floating bars, this plots the y-values of group dataset
//  with respect to the x-scale.
thePlot1.setBarOrient(QCChartTS.ChartConstants.HORIZ_DIR);

chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
var nnumpnts = 6;
var numgroups = 2;
var x1 = new Array(nnumpnts);
//  x-values to hold position
//  Calendar data to hold start/stop of floating bars
var y1 = QCChartTS.ChartSupport.newTimeArray2D(numgroups, nnumpnts);
//  Mercury
x1[0] = 6;
y1[0][0] = new Date(1960, QCChartTS.ChartConstants.JANUARY, 1);
y1[1][0] = new Date(1963, QCChartTS.ChartConstants.JANUARY, 1);
```

```
//  Gemini
x1[1] = 5;
y1[0][1] = new Date(1962, QCChartTS.ChartConstants.JANUARY, 1);
y1[1][1] = new Date(1967, QCChartTS.ChartConstants.JANUARY, 1);
//  Apollo
x1[2] = 4;
y1[0][2] = new Date(1963, QCChartTS.ChartConstants.JANUARY, 1);
y1[1][2] = new Date(1973, QCChartTS.ChartConstants.JANUARY, 1);
//  Skylab
x1[3] = 3;
y1[0][3] = new Date(1972, QCChartTS.ChartConstants.JANUARY, 1);
y1[1][3] = new Date(1974, QCChartTS.ChartConstants.JANUARY, 1);
//  Space Shuttle
x1[4] = 2;
y1[0][4] = new Date(1977, QCChartTS.ChartConstants.JANUARY, 1);
y1[1][4] = new Date(2005, QCChartTS.ChartConstants.DECEMBER, 31);
//  International Space Station
x1[5] = 1;
y1[0][5] = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);
y1[1][5] = new Date(2005, QCChartTS.ChartConstants.DECEMBER, 31);
theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.REGULAR, 10);
//  Time group dataset for floating bars
var Dataset1 = QCChartTS.TimeGroupDataset.newTimeGroupDatasetXDate2DY("Space programs", x1, y1);
var pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
//  Y values (ChartCalendar values) of dataset will actually
//  be plotted against x-coordinates
//  so need to swap the chart coordinate system x and y scale
pTransform1.swapScaleOrientation();
pTransform1.ScaleStartY = 0;
pTransform1.setGraphBorderDiagonal(0.15, 0.6, 0.95, 0.9);

.
.
.

//  Create floating bar plot
var thePlot1 = QCChartTS.FloatingBarPlot.newFloatingBarPlotCoords(pTransform1);
var attrib1 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
attrib1.FillFlag = true;
thePlot1.initFloatingBarPlot(Dataset1, 0.75, attrib1, QCChartTS.ChartConstants.JUSTIFY_CENTER);
//  Use horizontal floating bars, this plots the y-values of group dataset
//  with respect to the x-scale.
thePlot1.setBarOrient(QCChartTS.ChartConstants.HORIZ_DIR);

chartVu.addChartObject(thePlot1);ORIZ_DIR)
chartVu.addChartObject(thePlot1)
```

Floating Stacked Bar Plots

Class FloatingStackedBarPlot

GraphObj

```
   |
   +--ChartPlot
         |
         +--GroupPlot
               |
               +-- FloatingStackedBarPlot
```

The **FloatingStackedBarPlot** class extends the **GroupPlot** class and displays floating stacked bar plots. The bars are free floating because each bar does not reference a fixed base value, as do the simple bar plots, stacked bar plots and group bar plots. The starting value for each stacked bar is Y[0]. Each bar after that is defined by the succeeding value in the group value array (Y[1], Y[2]..). Unlike the **StackedBarPlot** plot, the displayed values are not a cumulative sum of the group values. All bars in a given **FloatingStackedBarPlot** object have the same width.

FloatingStackedBarPlot constructor

```
public static newFloatingStackedBarPlotCoords(transform: PhysicalCoordinates):
FloatingStackedBarPlot

public static newFloatingStackedBarPlot(transform: PhysicalCoordinates, dataset: GroupDataset,
        rbarwidth: number, attribs: ChartAttribute[], nbarjust: number): FloatingStackedBarPlot
```

transform	The coordinate system for the new **FloatingStackedBarPlot** object.
dataset	The starting value for each stacked bar is Y[0]. Each bar after that is defined by the succeeding value in the group value array (Y[1], Y[2]..).
rbarwidth	The width of the floating bars in units of the independent axis.
attrib	An array of **ChartAttribute** specifying the color for each bar. The number of colors, and the length of the array should be one less than the number of groups in the source dataset..
nbarjust	Specifies the justification with respect to the independent data value. Use one of the justification constants: JUSTIFY_MIN, JUSTIFY_CENTER, JUSTIFY_MAX.

FloatingStackedBarPlot plots can be used in place of **OHLCPlots**, or **CandlestickPlots** for the display of financial information. See the example below.

Floating stacked bar plot example for displaying stock data. Extracted from the NewDemosRev2. BuildFloatingStackedBars example.

[TypeScript]

```
let xValues: Date[] = new Array<Date>();
let stockPriceData: number[][];
let NASDAQData: number[];
let currentdate: Date;
let datastartdate: Date;
let thePlot1: QCChartTS.FloatingStackedBarPlot;
let thePlot2: QCChartTS.SimpleLinePlot;
let pTransform1: QCChartTS.TimeCoordinates;
let pTransform2: QCChartTS.TimeCoordinates;
let xAxis1: QCChartTS.TimeAxis;
let yAxis1: QCChartTS.LinearAxis;
let xAxisLab1: QCChartTS.TimeAxisLabels;
let yAxisLab1: QCChartTS.NumericAxisLabels;
let yAxis2: QCChartTS.LinearAxis;
let yAxisLab2: QCChartTS.NumericAxisLabels;

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

let chartVu: QCChartTS.ChartView = QCChartTS.ChartView.newChartView(htmlcanvas);
if (!chartVu) return;
chartVu.setPreferredSize(800, 600);

let nNumPnts: number = 80;
//  One hundred days
let nNumGroups: number = 4;
let weekmode: number = QCChartTS.ChartConstants.WEEK_5D;
let i: number;
currentdate = new Date();
datastartdate = new Date();
stockPriceData = QCChartTS.ChartSupport.newArray2D(nNumGroups, nNumPnts);
xValues = new Array(nNumPnts);
NASDAQData = new Array(nNumPnts);

let theFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.BOLD, 12);
currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
//  No weekend
datastartdate = new Date(currentdate);
xValues[0] = new Date(currentdate);
currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
//  values should be in ascending order
stockPriceData[0][0] = 22;
stockPriceData[1][0] = 24;
stockPriceData[2][0] = 26;
stockPriceData[3][0] = 28;
NASDAQData[0] = 1800;
for (i = 1; (i < nNumPnts); i++) {
    xValues[i] = new Date(currentdate);
    //  Unlike the OHLC chart, the values within a group need to be in ascending order
    stockPriceData[0][i] = stockPriceData[0][i - 1] + (3 * (0.52 -
QCChartTS.ChartSupport.getRandomDouble()));
    stockPriceData[1][i] = stockPriceData[0][i] + (2 * QCChartTS.ChartSupport.getRandomDouble());
    stockPriceData[2][i] = stockPriceData[1][i] + (1.5 *
QCChartTS.ChartSupport.getRandomDouble());
    stockPriceData[3][i] = stockPriceData[2][i] + (1.5 *
QCChartTS.ChartSupport.getRandomDouble());
    NASDAQData[i] = NASDAQData[i - 1] + (20 * (0.52 - QCChartTS.ChartSupport.getRandomDouble()));
    //  open
    currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
}

let Dataset1: QCChartTS.TimeGroupDataset =
QCChartTS.TimeGroupDataset.newTimeGroupDatasetDateX2DY("Stock Data", xValues, stockPriceData);
pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.setWeekType(weekmode);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
pTransform1.setGraphBorderDiagonal(0.1, .15, .90, 0.75);

.
.
.

let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.RED, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
```

```
let attrib2: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.YELLOW, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.YELLOW);
let attrib3: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.BLUE);
let attrib4: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.GREEN, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
let attribArray: QCChartTS.ChartAttribute[] = [
    attrib1,
    attrib2,
    attrib3,
    attrib4];
attrib1.setFillFlag(true);
thePlot1 = QCChartTS.FloatingStackedBarPlot.newFloatingStackedBarPlot(pTransform1, Dataset1,
QCChartTS.ChartCalendar.getCalendarWidthValue(QCChartTS.ChartConstants.DAY_OF_YEAR, 0.75),
attribArray, QCChartTS.ChartConstants.JUSTIFY_CENTER);
thePlot1.setFastClipMode(QCChartTS.C
```

[JavaScript]

```
let xValues = new Array();
let stockPriceData;
let NASDAQData;
let currentdate;
let datastartdate;
let thePlot1;
let thePlot2;
let pTransform1;
let pTransform2;
let xAxis1;
let yAxis1;
let xAxisLab1;
let yAxisLab1;
let yAxis2;
let yAxisLab2;

let htmlcanvas = document.getElementById(canvasid);

let chartVu = QCChartTS.ChartView.newChartView(htmlcanvas);
if (!chartVu) return;
chartVu.setPreferredSize(800, 600);

let nNumPnts = 80;
//  One hundred days
let nNumGroups = 4;
let weekmode = QCChartTS.ChartConstants.WEEK_5D;
let i;
currentdate = new Date();
datastartdate = new Date();
stockPriceData = QCChartTS.ChartSupport.newArray2D(nNumGroups, nNumPnts);
xValues = new Array(nNumPnts);
NASDAQData = new Array(nNumPnts);

let theFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif", QCChartTS.ChartFont.BOLD,
12);
currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
//  No weekend
datastartdate = new Date(currentdate);
xValues[0] = new Date(currentdate);
currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
//  values should be in ascending order
stockPriceData[0][0] = 22;
stockPriceData[1][0] = 24;
stockPriceData[2][0] = 26;
stockPriceData[3][0] = 28;
NASDAQData[0] = 1800;
for (i = 1; (i < nNumPnts); i++) {
    xValues[i] = new Date(currentdate);
    //  Unlike the OHLC chart, the values within a group need to be in ascending order
    stockPriceData[0][i] = stockPriceData[0][i - 1] + (3 * (0.52 -
QCChartTS.ChartSupport.getRandomDouble()));
    stockPriceData[1][i] = stockPriceData[0][i] + (2 * QCChartTS.ChartSupport.getRandomDouble());
    stockPriceData[2][i] = stockPriceData[1][i] + (1.5 *
QCChartTS.ChartSupport.getRandomDouble());
    stockPriceData[3][i] = stockPriceData[2][i] + (1.5 *
QCChartTS.ChartSupport.getRandomDouble());
    NASDAQData[i] = NASDAQData[i - 1] + (20 * (0.52 - QCChartTS.ChartSupport.getRandomDouble()));
    //  open
    currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
}

let Dataset1 = QCChartTS.TimeGroupDataset.newTimeGroupDatasetDateX2DY("Stock Data", xValues,
stockPriceData);
pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.setWeekType(weekmode);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
pTransform1.setGraphBorderDiagonal(0.1, .15, .90, 0.75);

.
.
.
let attrib1 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.RED, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
let attrib2 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.YELLOW, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.YELLOW);
let attrib3 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.BLUE);
let attrib4 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.GREEN, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
let attribArray = [
    attrib1,
    attrib2,
    attrib3,
    attrib4];
attrib1.setFillFlag(true);
thePlot1 = QCChartTS.FloatingStackedBarPlot.newFloatingStackedBarPlot(pTransform1, Dataset1,
QCChartTS.ChartCalendar.getCalendarWidthValue(QCChartTS.ChartConstants.DAY_OF_YEAR, 0.75),
attribArray, QCChartTS.ChartConstants.JUSTIFY_CENTER);
thePlot1.setFastClipMode(QCChartTS.ChartConstants.FASTCLIP_X);
chartVu.addChartObject(thePlot1);hartConstants.FASTCLIP_X);
chartVu.addChartObject(thePlot1);
```

Group Bar Plots

Class GroupBarPlot

```
GraphObj
   |
   +--ChartPlot
         |
         +--GroupPlot
               |
               +--GroupBarPlot
```

The **GroupBarPlot** class extends the **GroupPlot** class and displays data in a group bar format. Individual bars, the height of which corresponds to the group values (Y[0], Y[1], Y[2], ...) of the dataset, are displayed side by side, as a group, justified with respect to the X-position value for each group.

GroupBarPlot constructor

```
public static newGroupBarPlotCoords(transform: PhysicalCoordinates): GroupBarPlot

public static newGroupBarPlot(transform: PhysicalCoordinates,
  dataset: GroupDataset,rbarwidth: number, rbarbase: number, attribs: ChartAttribute[],
  nbarjust: number): GroupBarPlot
```

transform	The coordinate system for the new **GroupPlot** object.
dataset	The group bar graph represents the values in this group dataset. Individual bars, the height of which corresponds to the group values (Y[0], Y[1], Y[2], ...) of the dataset.
rbarwidth	The width of the group bars in units of the independent axis. All bars within a group are squeezed into the width defined by *rbarwidth*. Each individual bar within the group has a width of *rbarwidth*/*dataset*.getNumberGroups().
rbarbase	The group bars start at the value *rbarbase*, and extend to the group bar values represented by the dataset.
attribs	An array of **ChartAttribute** objects, sized the same as the number of groups in the dataset specify the attributes (outline color and fill color) for each group of a group bar graph.
nbarjust	The group bars are justified with respect to the x-values in the dataset using the *rbarjust* justification value (JUSTIFY_MIN, JUSTIFY_CENTER, or JUSTIFY_MAX).

The attributes for each group can set or modified using the setSegment… methods, where the segment number parameter cooresponds to the group number. These methods include setSegmentAttributes, setSegmentFillColor, setSegmentLineColor, and setSegmentColor.

Group bar plot example (extracted from the example program Bargraphs.BuildGroupBars)

[TypeScript]

```
let nNumPnts: number = 5, nNumGroups = 4;
let xValues: Date[] = new Array<Date>(nNumPnts);
let groupBarData: number[][] = QCChartTS.ChartSupport.newArray2D(nNumGroups, nNumPnts);

let theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 10);
xValues[0] = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);
```

```
groupBarData[0][0] = 6.3;
groupBarData[1][0] = 3.1;
groupBarData[2][0] = 2.2;
groupBarData[3][0] = 1.8;
xValues[1] = new Date(1999, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][1] = 5.8;
groupBarData[1][1] = 4.3;
groupBarData[2][1] = 2.8;
groupBarData[3][1] = 1.5;
xValues[2] = new Date(2000, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][2] = 5.5;
groupBarData[1][2] = 4.5;
groupBarData[2][2] = 2.5;
groupBarData[3][2] = 2.1;
xValues[3] = new Date(2001, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][3] = 4.1;
groupBarData[1][3] = 5.4;
groupBarData[2][3] = 4.1;
groupBarData[3][3] = 3.2;
xValues[4] = new Date(2002, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][4] = 3.8;
groupBarData[1][4] = 5.6;
groupBarData[2][4] = 4.3;
groupBarData[3][4] = 3.3;
let Dataset1: QCChartTS.TimeGroupDataset =
QCChartTS.TimeGroupDataset.newTimeGroupDatasetDateX2DY("GroupTimeData", xValues, groupBarData);
let pTransform1: QCChartTS.TimeCoordinates = new QCChartTS.TimeCoordinates();
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
pTransform1.setTimeScaleStart(new Date(1997, QCChartTS.ChartConstants.JANUARY, 1));
pTransform1.setTimeScaleStop(new Date(2003, QCChartTS.ChartConstants.JANUARY, 1));
pTransform1.setGraphBorderDiagonal(0.1, 0.1, 0.45, 0.75);

.
.
.
let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.RED, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
let attrib2: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.YELLOW, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.YELLOW);
let attrib3: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.BLUE);
let attrib4: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.GREEN, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
let attribArray: QCChartTS.ChartAttribute[] = [attrib1, attrib2, attrib3, attrib4];
let thePlot1: QCChartTS.GroupBarPlot = QCChartTS.GroupBarPlot.newGroupBarPlot(pTransform1,
Dataset1, QCChartTS.ChartCalendar.getCalendarWidthValue(QCChartTS.ChartConstants.YEAR, 0.75), 0,
attribArray, QCChartTS.ChartConstants.JUSTIFY_CENTER);
thePlot1.setBarOverlap(0);
chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
var nNumPnts = 5, nNumGroups = 4;
var xValues = new Array(nNumPnts);
var groupBarData = QCChartTS.ChartSupport.newArray2D(nNumGroups, nNumPnts);

var theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 10);
xValues[0] = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][0] = 6.3;
groupBarData[1][0] = 3.1;
groupBarData[2][0] = 2.2;
groupBarData[3][0] = 1.8;
xValues[1] = new Date(1999, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][1] = 5.8;
groupBarData[1][1] = 4.3;
```

```
groupBarData[2][1] = 2.8;
groupBarData[3][1] = 1.5;
xValues[2] = new Date(2000, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][2] = 5.5;
groupBarData[1][2] = 4.5;
groupBarData[2][2] = 2.5;
groupBarData[3][2] = 2.1;
xValues[3] = new Date(2001, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][3] = 4.1;
groupBarData[1][3] = 5.4;
groupBarData[2][3] = 4.1;
groupBarData[3][3] = 3.2;
xValues[4] = new Date(2002, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][4] = 3.8;
groupBarData[1][4] = 5.6;
groupBarData[2][4] = 4.3;
groupBarData[3][4] = 3.3;
var Dataset1 = QCChartTS.TimeGroupDataset.newTimeGroupDatasetDateX2DY("GroupTimeData", xValues,
groupBarData);
var pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
pTransform1.setTimeScaleStart(new Date(1997, QCChartTS.ChartConstants.JANUARY, 1));
pTransform1.setTimeScaleStop(new Date(2003, QCChartTS.ChartConstants.JANUARY, 1));
pTransform1.setGraphBorderDiagonal(0.1, 0.1, 0.45, 0.75);
.
.
.

var attrib1 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.RED, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
var attrib2 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.YELLOW, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.YELLOW);
var attrib3 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.BLUE);
var attrib4 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.GREEN, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
var attribArray = [attrib1, attrib2, attrib3, attrib4];
var thePlot1 = QCChartTS.GroupBarPlot.newGroupBarPlot(pTransform1, Dataset1,
QCChartTS.ChartCalendar.getCalendarWidthValue(QCChartTS.ChartConstants.YEAR, 0.75), 0,
attribArray, QCChartTS.ChartConstants.JUSTIFY_CENTER);
thePlot1.setBarOverlap(0);
chartVu.addChartObject(thePlot1);
```

Histogram Plots

Class HistogramPlot

```
GraphObj
   |
   +--ChartPlot
         |
         +--GroupPlot
               |
               +--HistogramPlot
```

The **HistogramPlot** class extends the **GroupPlot** class and displays histogram plots. A histogram plot is a collection of rectangular objects with independent positions, widths and heights, specified using the values of the associated group dataset. The number of groups must be two. The X-values of the group dataset represent the x-position of the lower left corner of each histogram bar, the Y[0] values set the height of each histogram bar, and the Y[1] values set the width of each histogram bar. The histogram bars share a common base value.

Histogram constructor

```
public static newHistogramPlotCoords(transform: PhysicalCoordinates): HistogramPlot

public static newHistogramPlot(transform: PhysicalCoordinates,
  dataset: GroupDataset, rbarbase: number, attrib: ChartAttribute): HistogramPlot
```

transform	The coordinate system for the new **HistogramPlot** object.
dataset	The histogram plot represents the values in this group dataset. The number of groups must be two. The X-values of the group dataset represent the x-position of the lower left corner of each histogram bar, the Y[0] values set the height of each histogram bar, and the Y[1] values set the width of each histogram bar.
rbarbase	The histogram bars start at the value *rbarbase*, and extend to the histogram bar values represented by the dataset.
attrib	Specifies the attributes (line color and line style) for the histogram bars.

An individual histogram bar in the histogram plot object can have unique attributes. Use the objects **setSegmentAttributesMode** and **setSegmentAttributes** methods in the manner described for **SimplePlot** objects. Each histogram bar can be labeled with the Y[0] group value bar (bar height) using the bar data point methods, see the example below.

Histogram plot example (extracted from the example program Bargraphs.BuildHistogram)

[TypeScript]

```
let nnumpnts: number = 6;
let numgroups: number = 2;
let x1: number[] = new Array(nnumpnts);
let y1: number[][] = QCChartTS.ChartSupport.newArray2D(numgroups, nnumpnts);
//    height   width
x1[0] = 0; y1[0][0] = .12; y1[1][0] = 13;
x1[1] = 13; y1[0][1] = .97; y1[1][1] = 7;
x1[2] = 20; y1[0][2] = .80; y1[1][2] = 10;
x1[3] = 30; y1[0][3] = .44; y1[1][3] = 10;
x1[4] = 40; y1[0][4] = .28; y1[1][4] = 20;
x1[5] = 60; y1[0][5] = .4; y1[1][5] = 20;

theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 10);
```

```
let Dataset1: QCChartTS.GroupDataset = QCChartTS.GroupDataset.newGroupDatasetX2DY("Actual Sales",
x1, y1);
let pTransform1: QCChartTS.CartesianCoordinates = new QCChartTS.CartesianCoordinates();
pTransform1.setScaleStartY(0);
pTransform1.setScaleStartX(0);
pTransform1.setScaleStopX(80);
pTransform1.setScaleStopY(1);
pTransform1.setGraphBorderDiagonal(0.15, 0.15, 0.9, 0.75);

.
.
.
chartVu.addChartObject(ygrid);
let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 0,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
attrib1.setFillFlag(true);
let thePlot1: QCChartTS.HistogramPlot = QCChartTS.HistogramPlot.newHistogramPlot(pTransform1,
Dataset1, 0, attrib1);
let bardatavalue: QCChartTS.NumericLabel = thePlot1.getPlotLabelTemplate();
bardatavalue.setTextFont(theFont);
bardatavalue.setNumericFormat(QCChartTS.ChartConstants.PERCENTFORMAT);
bardatavalue.setColor(QCChartTS.ChartColor.BLACK);
thePlot1.setBarDatapointLabelPosition(QCChartTS.ChartConstants.INSIDE_BAR);
thePlot1.setPlotLabelTemplate(bardatavalue);
thePlot1.setShowDatapointValue(true);
thePlot1.setSegmentAttributesMode(true);
thePlot1.setSegmentFillColor(0, QCChartTS.ChartColor.RED);
thePlot1.setSegmentFillColor(1, QCChartTS.ChartColor.MAGENTA);
thePlot1.setSegmentFillColor(2, QCChartTS.ChartColor.BLUE);
thePlot1.setSegmentFillColor(3, QCChartTS.ChartColor.GREEN);
thePlot1.setSegmentFillColor(4, QCChartTS.ChartColor.YELLOW);
thePlot1.setSegmentFillColor(5, QCChartTS.ChartColor.PINK);
chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
var nnumpnts = 6;
var numgroups = 2;
var x1 = new Array(nnumpnts);
var y1 = QCChartTS.ChartSupport.newArray2D(numgroups, nnumpnts);
//height   width
x1[0] = 0; y1[0][0] = .12; y1[1][0] = 13;
x1[1] = 13; y1[0][1] = .97; y1[1][1] = 7;
x1[2] = 20; y1[0][2] = .80; y1[1][2] = 10;
x1[3] = 30; y1[0][3] = .44; y1[1][3] = 10;
x1[4] = 40; y1[0][4] = .28; y1[1][4] = 20;
x1[5] = 60; y1[0][5] = .4; y1[1][5] = 20;

theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 10);
var Dataset1 = QCChartTS.GroupDataset.newGroupDatasetX2DY("Actual Sales", x1, y1);
var pTransform1 = new QCChartTS.CartesianCoordinates();
pTransform1.setScaleStartY(0);
pTransform1.setScaleStartX(0);
pTransform1.setScaleStopX(80);
pTransform1.setScaleStopY(1);
pTransform1.setGraphBorderDiagonal(0.15, 0.15, 0.9, 0.75);

.
.
.
var attrib1 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 0,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
attrib1.setFillFlag(true);
var thePlot1 = QCChartTS.HistogramPlot.newHistogramPlot(pTransform1, Dataset1, 0, attrib1);
var bardatavalue = thePlot1.getPlotLabelTemplate();
bardatavalue.setTextFont(theFont);
bardatavalue.setNumericFormat(QCChartTS.ChartConstants.PERCENTFORMAT);
bardatavalue.setColor(QCChartTS.ChartColor.BLACK);
thePlot1.setBarDatapointLabelPosition(QCChartTS.ChartConstants.INSIDE_BAR);
thePlot1.setPlotLabelTemplate(bardatavalue);
thePlot1.setShowDatapointValue(true);
thePlot1.setSegmentAttributesMode(true);
thePlot1.setSegmentFillColor(0, QCChartTS.ChartColor.RED);
thePlot1.setSegmentFillColor(1, QCChartTS.ChartColor.MAGENTA);
thePlot1.setSegmentFillColor(2, QCChartTS.ChartColor.BLUE);
thePlot1.setSegmentFillColor(3, QCChartTS.ChartColor.GREEN);
thePlot1.setSegmentFillColor(4, QCChartTS.ChartColor.YELLOW);
thePlot1.setSegmentFillColor(5, QCChartTS.ChartColor.PINK);
chartVu.addChartObject(thePlot1);
```

Line Gap Plots

Class LineGapPlot

```
GraphObj
   |
   +--ChartPlot
         |
         +--GroupPlot
               |
               +-- LineGapPlot
```

The **LineGapPlot** class extends the **GroupPlot** class and displays a line gap chart. The number of groups must be two. A line gap chart consists of two line plots where a contrasting color fills and highlights the area between the two lines. The (X, Y[0]) values of the group dataset represent the first of the bounding lines, and the (X,Y[1]) values of the group dataset represent the second of the bounding lines.

LineGapPlot constructor

```
public static newLineGapPlotCoords(transform: PhysicalCoordinates): LineGapPlot

public static newLineGapPlotCoordsDatasetAttrib(transform: PhysicalCoordinates,
  dataset: GroupDataset, attrib: ChartAttribute): LineGapPlot
```

transform	The coordinate system for the new **LineGapPlot** object.
dataset	The line gap plot represents the values in this group dataset. The number of groups in this group dataset must be two.
attrib	Specifies the attributes (line and fill color) for the fill area.

A segment between adjacent x-values in the line gap plot object can have unique attributes. Use the objects **setSegmentAttributesMode** and **setSegmentAttributes** methods in the manner described for **SimplePlot** objects..

Line gap bar plot example (extracted from the example program MiscCharts.BuildLineGapPlot).

[TypeScript]

```
let nNumPnts: number = 5;
let nNumGroups: number = 2;
let xValues: Date[]= new Array<Date>(nNumPnts);
let groupBarData: number[][] =QCChartTS.ChartSupport.newArray2D(nNumGroups, nNumPnts);
let theFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.BOLD, 12);
xValues[0] = new Date(1998,QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][ 0] = 43;
groupBarData[1][ 0] = 71;
xValues[1] = new Date(1999,QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][ 1] = 40;
groupBarData[1][ 1] = 81;
xValues[2] = new Date(2000,QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][ 2] = 54;
groupBarData[1][ 2] = 71;
xValues[3] = new Date(2001,QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][ 3] = 56;
groupBarData[1][ 3] = 65;
xValues[4] = new Date(2002,QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][ 4] = 58;
groupBarData[1][ 4] = 63;
let Dataset1: QCChartTS.TimeGroupDataset =
QCChartTS.TimeGroupDataset.newTimeGroupDatasetDateX2DY("GroupTimeData", xValues, groupBarData);
let pTransform1: QCChartTS.TimeCoordinates = new QCChartTS.TimeCoordinates();
pTransform1.autoScaleDatasetRoundMode(Dataset1,QCChartTS.ChartConstants.AUTOAXES_NEAR,QCChartTS.C
hartConstants.AUTOAXES_FAR);
pTransform1.setGraphBorderDiagonal(0.15, .1, .95, 0.8);

.
.
.
let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
attrib1.setFillFlag(true);
attrib1.setLineFlag(false);
let thePlot1: QCChartTS.LineGapPlot
=QCChartTS.LineGapPlot.newLineGapPlotCoordsDatasetAttrib(pTransform1, Dataset1, attrib1);
thePlot1.StepMode =QCChartTS.ChartConstants.STEP_END;
chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
let nNumPnts = 5;
let nNumGroups = 2;
let xValues = new Array(nNumPnts);
let groupBarData = QCChartTS.ChartSupport.newArray2D(nNumGroups, nNumPnts);
let theFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif", QCChartTS.ChartFont.BOLD,
12);
xValues[0] = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][0] = 43;
```

```
groupBarData[1][0] = 71;
xValues[1] = new Date(1999, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][1] = 40;
groupBarData[1][1] = 81;
xValues[2] = new Date(2000, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][2] = 54;
groupBarData[1][2] = 71;
xValues[3] = new Date(2001, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][3] = 56;
groupBarData[1][3] = 65;
xValues[4] = new Date(2002, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][4] = 58;
groupBarData[1][4] = 63;
let Dataset1 = QCChartTS.TimeGroupDataset.newTimeGroupDatasetDateX2DY("GroupTimeData", xValues,
groupBarData);
let pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setGraphBorderDiagonal(0.15, .1, .95, 0.8);
.
.
.

let attrib1 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
attrib1.setFillFlag(true);
attrib1.setLineFlag(false);
let thePlot1 = QCChartTS.LineGapPlot.newLineGapPlotCoordsDatasetAttrib(pTransform1, Dataset1,
attrib1);
thePlot1.StepMode = QCChartTS.ChartConstants.STEP_END;
chartVu.addChartObject(thePlot1);
```

Multi-Line Plots

Class MultiLinePlot

GraphObj
```
        |
        +--ChartPlot
                |
                +--GroupPlot
                        |
                        +-- MultiLinePlot
```

The **MultiLinePlot** class extends the **GroupPlot** class and displays group data in multi-line format. A group dataset with eight groups will display eight separate line plots. The y-values for each group of the dataset are the y-values for each line in the plot. Each line plot share the same x-values of the group dataset.

MultiLinePlot constructor

```
public static newMultiLinePlotCoords(transform: PhysicalCoordinates): MultiLinePlot

public static newMultiLinePlot(transform: PhysicalCoordinates, dataset: GroupDataset,
        attribs: ChartAttribute[]): MultiLinePlot
```

transform	The coordinate system for the new **MultiLinePlot** object.
dataset	The multi-line plot represents the values in this group dataset.
attribs	An array of **ChartAttribute** objects, sized the same as the number of groups in the dataset, specify the attributes (line color and line style) for each group of the multi-line plot.

The attributes for each group can set or modified using the setSegment… methods, where the segment number parameter cooresponds to the group number. These methods include setSegmentAttributes, setSegmentFillColor, setSegmentLineColor, and setSegmentColor.

Multi-line plot example (extracted from the example program MultiLinePlots.BuildMultiLine)

[TypeScript]

```
let numPoints: number = 100;
let numGroups: number = 7;
let x1: number[] = new Array(numPoints);
let y1: number[][] = QCChartTS.ChartSupport.newArray2D(numGroups, numPoints);
let i: number;
let j: number;
for (i = 0; i < numPoints; i++) {
    x1[i] = i * 0.2;
    for (j = 0; j < numGroups; j++)
y1[j][i] = j * (i * 0.01) + (j + 1) * 5.0 * (1.0 - Math.exp(-x1[i] / 0.7));
}


theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
let Dataset1: QCChartTS.GroupDataset = QCChartTS.GroupDataset.newGroupDatasetX2DY("First", x1,
y1);
let pTransform1: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setScaleStartX(0);
pTransform1.setScaleStartY(0);
let background: QCChartTS.Background = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.PLOT_BACKGROUND, QCChartTS.ChartColor.fromRgb(255, 255, 255));
chartVu.addChartObject(background);
//  should establish border before background
pTransform1.setGraphBorderDiagonal(0.15, .15, .90, 0.7);

.
.
.
let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 3,
QCChartTS.ChartConstants.LS_SOLID);
let attribArray: QCChartTS.ChartAttribute[] = new Array(numGroups);
```

```
for (i = 0; (i < numGroups); i++) {
    attribArray[i] = (<QCChartTS.ChartAttribute>(attrib1.clone()));
}

let thePlot1: QCChartTS.MultiLinePlot = QCChartTS.MultiLinePlot.newMultiLinePlot(pTransform1,
Dataset1, attribArray);
chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
let numPoints = 100;
let numGroups = 7;
let x1 = new Array(numPoints);
let y1 = QCChartTS.ChartSupport.newArray2D(numGroups, numPoints);
let i;
let j;
for (i = 0; i < numPoints; i++) {
x1[i] = i * 0.2;
for (j = 0; j < numGroups; j++)
y1[j][i] = j * (i * 0.01) + (j + 1) * 5.0 * (1.0 - Math.exp(-x1[i] / 0.7));
}
y1[0][5] = QCChartTS.ChartConstants.rBadDataValue;
y1[3][15] = QCChartTS.ChartConstants.rBadDataValue;

theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
let Dataset1 = QCChartTS.GroupDataset.newGroupDatasetX2DY("First", x1, y1);
let pTransform1 =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setScaleStartX(0);
pTransform1.setScaleStartY(0);
let background = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.PLOT_BACKGROUND, QCChartTS.ChartColor.fromRgb(255, 255, 255));
chartVu.addChartObject(background);
//  should establish border before background
pTransform1.setGraphBorderDiagonal(0.15, .15, .90, 0.7);

.
.
.

let attrib1 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 3,
QCChartTS.ChartConstants.LS_SOLID);
let attribArray = new Array(numGroups);
for (i = 0; (i < numGroups); i++) {
attribArray[i] = ((attrib1.clone()));
}

let thePlot1 = QCChartTS.MultiLinePlot.newMultiLinePlot(pTransform1, Dataset1, attribArray);
chartVu.addChartObject(thePlot1);
```

Open-High-Low-Close Plots

Class OHLCPlot

```
GraphObj
   |
   +--ChartPlot
         |
+--GroupPlot
       |
       +-- OHLCPlot
```

The **OHLCPlot** class extends the **GroupPlot** class and displays stock market data in an open-high-low-close format common in financial technical analysis. Every item of the plot is a vertical line, representing High and Low values, with two small horizontal "flags", one left and one right extending from the vertical High-Low line and representing the Open and Close values. The number of groups must be four. The Y[0] values of the group dataset represent the values for Open, the Y[1] values for High, the Y[2] values for Low, and the Y[3] values for Close.

OHLCPlot constructor

```
public static newOHLCPlotCoords(transform: PhysicalCoordinates): OHLCPlot

public static newOHLCPlot(transform: PhysicalCoordinates, dataset: GroupDataset,
        rflagwidth: number, attrib: ChartAttribute): OHLCPlot
```

transform	The coordinate system for the new **OHLCPlot** object.
dataset	The **OHLCPlot** plot will represent the group open-high-low-close values in this group dataset. The number of groups must be four. The Y[0] values of the group dataset represent the values for Open, the Y[1] values for High, the Y[2] values for Low, and the Y[3] values for Close.
rflagwidth	The width of the open and close markers in units of the independent axis.
attrib	Specifies the attributes (line color and line style) for the open-high-low-close plot.

An individual OHLC element in an OHLC plot object can have unique attributes. Use the objects **setSegmentAttributesMode** and **setSegmentAttributes** methods in the manner described for **SimplePlot** objects.

OHLC plot example (extracted from the example program FinancialExamples.BuildOHLCChart)

[TypeScript]

```
let xValues: Date[];
let stockPriceData: number[][];
let stockVolumeData: number[];
let NASDAQData: number[];

let currentdate: Date;
let datastartdate: Date;
let thePlot1: QCChartTS.OHLCPlot;
let thePlot2: QCChartTS.SimpleBarPlot;
let thePlot3: QCChartTS.SimpleLinePlot;
let pTransform1: QCChartTS.TimeCoordinates;
let pTransform2: QCChartTS.TimeCoordinates;
let pTransform3: QCChartTS.TimeCoordinates;
let xAxis1: QCChartTS.TimeAxis;
let yAxis1: QCChartTS.LinearAxis;
let xAxisLab1: QCChartTS.TimeAxisLabels;
let yAxisLab1: QCChartTS.NumericAxisLabels;
let xAxis2: QCChartTS.TimeAxis;
let yAxis2: QCChartTS.LinearAxis;
let yAxisLab2: QCChartTS.NumericAxisLabels;
let yAxis3: QCChartTS.LinearAxis;
let yAxisLab3: QCChartTS.NumericAxisLabels;

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

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

let nNumPnts: number = 90;
let nNumGroups: number = 4;
let weekmode: number = QCChartTS.ChartConstants.WEEK_5D;
let maxval: number = 0;
let minval: number = 0;
let i: number;


currentdate = new Date();
datastartdate = new Date();
stockPriceData = QCChartTS.ChartSupport.newArray2D(nNumGroups, nNumPnts);

xValues = new Array(nNumPnts);
stockVolumeData = new Array(nNumPnts);
NASDAQData = new Array(nNumPnts);
theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
//  No weekend
datastartdate = new Date(currentdate);
xValues[0] = new Date(currentdate);
currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);


stockPriceData[3][0] = 25;
//  close
stockPriceData[0][0] = 25;
//  open
stockPriceData[1][0] = 26;
//  high
stockPriceData[2][0] = 24;
//  low
stockVolumeData[0] = 1400000;
NASDAQData[0] = 1800;
for (i = 1; (i < nNumPnts); i++) {
    xValues[i] = new Date(currentdate);
    stockPriceData[3][i] = stockPriceData[3][i - 1] + (3 * (0.52 -
QCChartTS.ChartSupport.getRandomDouble()));
    //  close
    stockPriceData[0][i] = stockPriceData[0][i] + (stockPriceData[3][i] + (2 * (0.5 -
QCChartTS.ChartSupport.getRandomDouble())));
    //  open
    minval = Math.min(stockPriceData[3][i], stockPriceData[0][i]);
    maxval = Math.max(stockPriceData[3][i], stockPriceData[0][i]);
    stockPriceData[1][i] = (maxval + (1.5 * QCChartTS.ChartSupport.getRandomDouble()));
    //  high
    stockPriceData[2][i] = (minval - (1.5 * QCChartTS.ChartSupport.getRandomDouble()));
    //  low
    stockVolumeData[i] = (3000000 * QCChartTS.ChartSupport.getRandomDouble());
    NASDAQData[i] = NASDAQData[i - 1] + (20 * (0.52 - QCChartTS.ChartSupport.getRandomDouble()));
    //  open
    currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);

}

let Dataset1: QCChartTS.TimeGroupDataset =
QCChartTS.TimeGroupDataset.newTimeGroupDatasetDateX2DY("Stock Data", xValues, stockPriceData);
pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.setWeekType(weekmode);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
pTransform1.setGraphBorderDiagonal(0.1, .15, .90, 0.55);

.
.
.

let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.RED, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
attrib1.setFillFlag(true);
thePlot1 = QCChartTS.OHLCPlot.newOHLCPlot(pTransform1, Dataset1,
QCChartTS.ChartCalendar.getCalendarWidthValue(QCChartTS.ChartConstants.DAY_OF_YEAR, 0.75),
attrib1);
thePlot1.setFastClipMode(QCChartTS.ChartConstants.FASTCLIP_X);
chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
var xValues;
var stockPriceData;
var stockVolumeData;
var NASDAQData;

var currentdate;
var datastartdate;
var thePlot1;
var thePlot2;
var thePlot3;
var pTransform1;
var pTransform2;
var pTransform3;
var xAxis1;
var yAxis1;
var xAxisLab1;
var yAxisLab1;
var xAxis2;
var yAxis2;
var yAxisLab2;
var yAxis3;
var yAxisLab3;

var htmlcanvas = document.getElementById(canvasid);

var chartVu = QCChartTS.ChartView.newChartView(htmlcanvas);
if (!chartVu) return;
chartVu.setPreferredSize(800, 600);
var theFont;

var nNumPnts = 90;
var nNumGroups = 4;
var weekmode = QCChartTS.ChartConstants.WEEK_5D;
var maxval = 0;
var minval = 0;
var i;


currentdate = new Date();
```

```
datastartdate = new Date();
stockPriceData = QCChartTS.ChartSupport.newArray2D(nNumGroups, nNumPnts);

xValues = new Array(nNumPnts);
stockVolumeData = new Array(nNumPnts);
NASDAQData = new Array(nNumPnts);
theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);
//  No weekend
datastartdate = new Date(currentdate);
xValues[0] = new Date(currentdate);
currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);


stockPriceData[3][0] = 25;
//  close
stockPriceData[0][0] = 25;
//  open
stockPriceData[1][0] = 26;
//  high
stockPriceData[2][0] = 24;
//  low
stockVolumeData[0] = 1400000;
NASDAQData[0] = 1800;
for (i = 1; (i < nNumPnts); i++) {
xValues[i] = new Date(currentdate);
stockPriceData[3][i] = stockPriceData[3][i - 1] + (3 * (0.52 -
QCChartTS.ChartSupport.getRandomDouble()));
//  close
stockPriceData[0][i] = stockPriceData[0][i] + (stockPriceData[3][i] + (2 * (0.5 -
QCChartTS.ChartSupport.getRandomDouble())));
//  open
minval = Math.min(stockPriceData[3][i], stockPriceData[0][i]);
maxval = Math.max(stockPriceData[3][i], stockPriceData[0][i]);
stockPriceData[1][i] = (maxval + (1.5 * QCChartTS.ChartSupport.getRandomDouble()));
//  high
stockPriceData[2][i] = (minval - (1.5 * QCChartTS.ChartSupport.getRandomDouble()));
//  low
stockVolumeData[i] = (3000000 * QCChartTS.ChartSupport.getRandomDouble());
NASDAQData[i] = NASDAQData[i - 1] + (20 * (0.52 - QCChartTS.ChartSupport.getRandomDouble()));
//  open
currentdate = QCChartTS.ChartCalendar.calendarDaysAdd(currentdate, 1, weekmode);

}

var Dataset1 = QCChartTS.TimeGroupDataset.newTimeGroupDatasetDateX2DY("Stock Data", xValues,
stockPriceData);
pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.setWeekType(weekmode);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
pTransform1.setGraphBorderDiagonal(0.1, .15, .90, 0.55);
.
.
.

var attrib1 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.RED, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
attrib1.setFillFlag(true);
thePlot1 = QCChartTS.OHLCPlot.newOHLCPlot(pTransform1, Dataset1,
QCChartTS.ChartCalendar.getCalendarWidthValue(QCChartTS.ChartConstants.DAY_OF_YEAR, 0.75),
attrib1);
thePlot1.setFastClipMode(QCChartTS.ChartConstants.FASTCLIP_X);
chartVu.addChartObject(thePlot1);
```

* Note how the **ChartCalendar.getCalendarWidthValue** method calculates the width of the bars as a function of time, in this case a width of 0.75 days.

Stacked Bar Plots

Class StackedBarPlot

```
GraphObj
   |
   +--ChartPlot
         |
         +--GroupPlot
               |
               +--StackedBarPlot
```

The **StackedBarPlot** class extends the **GroupPlot** class and displays data in stacked bar format. In a stacked bar plot each group is stacked on top of one another, each group bar a cumulative sum of the related group items before it.

StackedBarPlot constructor

```
public static newStackedBarPlotCoords(transform: PhysicalCoordinates): StackedBarPlot

public static newStackedBarPlot(transform: PhysicalCoordinates,
        dataset: GroupDataset, rbarwidth: number, rbarbase: number,
        attribs: ChartAttribute[], nbarjust: number): StackedBarPlot
```

transform	The coordinate system for the new **StackedBarPlot** object.
dataset	The stacked bar graph represents the values in this group dataset.
rbarwidth	The width of the stacked bars in units of the independent axis.
rbarbase	The stacked bars start at the value *rbarbase*, and extend to the group bar values represented by the dataset.
attribs	An array of **ChartAttribute** objects, sized the same as the number of groups in the dataset, that specify the attributes (outline color and fill color) for each group of a stacked bar graph.
nbarjust	The stacked bars are justified with respect to the x-values in the dataset using the *rbarjust* justification value (JUSTIFY_MIN, JUSTIFY_CENTER, or JUSTIFY_MAX).

Each stacked bar can be labeled with the group value bar using the bar data point methods, see the example below.

The attributes for each group can set or modified using the setSegment… methods, where the segment number parameter cooresponds to the group number. These methods include setSegmentAttributes, setSegmentFillColor, setSegmentLineColor, and setSegmentColor.

Stacked bar plot example (extracted from the example program Bargraphs,.BuildGroupBars)

[TypeScript]

```
let nNumPnts: number = 5, nNumGroups = 4;
let xValues: Date[] = new Array<Date>(nNumPnts);
let groupBarData: number[][] = QCChartTS.ChartSupport.newArray2D(nNumGroups, nNumPnts);
      //  let groupBarData: number[][] =[[6, 3, 2, 1, 1], [6, 4, 3, 1, 2], [7, 4, 5, 6, 3],
[2,3,4,5, 6]];

let theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 10);
xValues[0] = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][0] = 6.3;
groupBarData[1][0] = 3.1;
groupBarData[2][0] = 2.2;
groupBarData[3][0] = 1.8;
xValues[1] = new Date(1999, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][1] = 5.8;
groupBarData[1][1] = 4.3;
groupBarData[2][1] = 2.8;
groupBarData[3][1] = 1.5;
xValues[2] = new Date(2000, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][2] = 5.5;
groupBarData[1][2] = 4.5;
groupBarData[2][2] = 2.5;
groupBarData[3][2] = 2.1;
xValues[3] = new Date(2001, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][3] = 4.1;
groupBarData[1][3] = 5.4;
groupBarData[2][3] = 4.1;
groupBarData[3][3] = 3.2;
xValues[4] = new Date(2002, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][4] = 3.8;
groupBarData[1][4] = 5.6;
groupBarData[2][4] = 4.3;
groupBarData[3][4] = 3.3;

let Dataset1: QCChartTS.TimeGroupDataset =
QCChartTS.TimeGroupDataset.newTimeGroupDatasetDateX2DY("GroupTimeData", xValues, groupBarData);

let pTransform2: QCChartTS.TimeCoordinates = new QCChartTS.TimeCoordinates();
//  User same dataset as Group bar plot, set stacked mode flag
Dataset1.setStackMode(QCChartTS.ChartConstants.AUTOAXES_STACKED);
pTransform2.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
pTransform2.setTimeScaleStart(new Date(1997, QCChartTS.ChartConstants.JANUARY, 1));
pTransform2.setTimeScaleStop(new Date(2003, QCChartTS.ChartConstants.JANUARY, 1));
pTransform2.setGraphBorderDiagonal(0.55, 0.1, 0.95, 0.75);
pTransform2.setScaleStartY(0);

.
.
.
```

```
let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.RED, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
let attrib2: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.YELLOW, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.YELLOW);
let attrib3: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.BLUE);
let attrib4: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.GREEN, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
let attribArray: QCChartTS.ChartAttribute[] = [attrib1, attrib2, attrib3, attrib4];

let thePlot2: QCChartTS.StackedBarPlot = QCChartTS.StackedBarPlot.newStackedBarPlot(pTransform2,
Dataset1, QCChartTS.ChartCalendar.getCalendarWidthValue(QCChartTS.ChartConstants.YEAR, 0.75), 0,
attribArray, QCChartTS.ChartConstants.JUSTIFY_CENTER);
let bardatavalue: QCChartTS.NumericLabel = thePlot2.getPlotLabelTemplate();
bardatavalue.setTextFont(theFont);
bardatavalue.setNumericFormat(QCChartTS.ChartConstants.CURRENCYFORMAT);
bardatavalue.setDecimalPos(1);
bardatavalue.setColor(QCChartTS.ChartColor.BLACK);
thePlot2.setPlotLabelTemplate(bardatavalue);
thePlot2.setBarDatapointLabelPosition(QCChartTS.ChartConstants.CENTERED_BAR);
thePlot2.setShowDatapointValue(true);
chartVu.addChartObject(thePlot2);
```

[JavaScript]

```
var nNumPnts = 5, nNumGroups = 4;
var xValues = new Array(nNumPnts);
var groupBarData = QCChartTS.ChartSupport.newArray2D(nNumGroups, nNumPnts);

var theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 10);
xValues[0] = new Date(1998, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][0] = 6.3;
groupBarData[1][0] = 3.1;
groupBarData[2][0] = 2.2;
groupBarData[3][0] = 1.8;
xValues[1] = new Date(1999, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][1] = 5.8;
groupBarData[1][1] = 4.3;
groupBarData[2][1] = 2.8;
groupBarData[3][1] = 1.5;
xValues[2] = new Date(2000, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][2] = 5.5;
groupBarData[1][2] = 4.5;
groupBarData[2][2] = 2.5;
groupBarData[3][2] = 2.1;
xValues[3] = new Date(2001, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][3] = 4.1;
groupBarData[1][3] = 5.4;
groupBarData[2][3] = 4.1;
groupBarData[3][3] = 3.2;
xValues[4] = new Date(2002, QCChartTS.ChartConstants.JANUARY, 1);
groupBarData[0][4] = 3.8;
groupBarData[1][4] = 5.6;
groupBarData[2][4] = 4.3;
groupBarData[3][4] = 3.3;
var Dataset1 = QCChartTS.TimeGroupDataset.newTimeGroupDatasetDateX2DY("GroupTimeData", xValues,
groupBarData);

.
.
.

var attrib1 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.RED, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.RED);
```

```
var attrib2 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.YELLOW, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.YELLOW);
var attrib3 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.BLUE);
var attrib4 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.GREEN, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.GREEN);
var attribArray = [attrib1, attrib2, attrib3, attrib4];


var pTransform2 = new QCChartTS.TimeCoordinates();
//  User same dataset as Group bar plot, set stacked mode flag
Dataset1.setStackMode(QCChartTS.ChartConstants.AUTOAXES_STACKED);
pTransform2.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_NEAR);
pTransform2.setTimeScaleStart(new Date(1997, QCChartTS.ChartConstants.JANUARY, 1));
pTransform2.setTimeScaleStop(new Date(2003, QCChartTS.ChartConstants.JANUARY, 1));
pTransform2.setGraphBorderDiagonal(0.55, 0.1, 0.95, 0.75);
pTransform2.setScaleStartY(0);

.
.
.
var thePlot2 = QCChartTS.StackedBarPlot.newStackedBarPlot(pTransform2, Dataset1,
QCChartTS.ChartCalendar.getCalendarWidthValue(QCChartTS.ChartConstants.YEAR, 0.75), 0,
attribArray, QCChartTS.ChartConstants.JUSTIFY_CENTER);
var bardatavalue = thePlot2.getPlotLabelTemplate();
bardatavalue.setTextFont(theFont);
bardatavalue.setNumericFormat(QCChartTS.ChartConstants.CURRENCYFORMAT);
bardatavalue.setDecimalPos(1);
bardatavalue.setColor(QCChartTS.ChartColor.BLACK);
thePlot2.setPlotLabelTemplate(bardatavalue);
thePlot2.setBarDatapointLabelPosition(QCChartTS.ChartConstants.CENTERED_BAR);
thePlot2.setShowDatapointValue(true);
chartVu.addChartObject(thePlot2);
```

Stacked Line Plots

Class StackedLinePlot

```
GraphObj
   |
   +--ChartPlot
         |
         +--GroupPlot
               |
               +--StackedLinePlot
```

The **StackedLinePlot** class extends the **GroupPlot** class and displays data in stacked line format. In a stacked line plot each group is stacked on top of one another, each group line a cumulative sum of the groups before it.

StackedLinePlot constructor

```
public static newStackedLinePlotCoords(transform: PhysicalCoordinates): StackedLinePlot

public static newStackedLinePlot(transform: PhysicalCoordinates,
        dataset: GroupDataset, attribs: ChartAttribute[]): StackedLinePlot
```

transform	The coordinate system for the new **StackedLinePlot** object.
dataset	The stacked line plot represents the values in this group dataset.
attribs	An array of **ChartAttribute** objects, sized the same as the number of groups in the dataset specify the attributes (line color and line style) for each group of the stacked line graph.

The attributes for each group can set or modified using the setSegment… methods, where the segment number parameter cooresponds to the group number. These methods include setSegmentAttributes, setSegmentFillColor, setSegmentLineColor, and setSegmentColor.

Stacked line plot example (extracted from the example program MultiLinePlots.BuildStackedLines)

[TypeScript]

```
let numPoints: number = 100;
let numGroups: number = 7;
let x1: number[] = new Array(numPoints);
let y1: number[][] = QCChartTS.ChartSupport.newArray2D(numGroups, numPoints);
let k: number;
let i: number;
let j: number;
for (i = 0; i < numPoints; i++) {
x1[i] = i * 0.2;

   for (i = 0; i < numPoints; i++) {
            x1[i] = i * 0.2;

            for (j = 0; j < numGroups; j++) {
                switch (j) {
                    case 0: y1[j][i] = 5 * QCChartTS.ChartSupport.getRandomDouble() + 25.0 * (1.0
- Math.exp(-x1[i] / 60.0)); break;
                    case 1: y1[j][i] = 8 * QCChartTS.ChartSupport.getRandomDouble() + 25.0 * (1.0
- Math.exp(-x1[i] / 50.0)); break;
                    case 2: y1[j][i] = 13 * QCChartTS.ChartSupport.getRandomDouble() + 35.0 *
(1.0 - Math.exp(-x1[i] / 40.0)); break;
                    case 3: y1[j][i] = 4 * QCChartTS.ChartSupport.getRandomDouble() + 15.0 * (1.0
- Math.exp(-x1[i] / 30.0)); break;
                    case 4: y1[j][i] = 9 * QCChartTS.ChartSupport.getRandomDouble() + 35.0 * (1.0
- Math.exp(-x1[i] / 20.0)); break;
                    case 5: y1[j][i] = 3 * QCChartTS.ChartSupport.getRandomDouble() + 15.0 * (1.0
- Math.exp(-x1[i] / 30.0)); break;
                    case 6: y1[j][i] = 7 * QCChartTS.ChartSupport.getRandomDouble() + 25.0 * (1.0
- Math.exp(-x1[i] / 50.0)); break;
                }
                if ((i >= numPoints / 2 - 7) && (i <= numPoints / 2 + 7)) {
                    for (k = 0; k < numGroups; k++)
                        y1[k][i] += (123 - 10 * Math.abs((numPoints / 2) - i)) *
QCChartTS.ChartSupport.getRandomDouble();
```

```
                }
            }
        }


theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
let Dataset1: QCChartTS.GroupDataset = QCChartTS.GroupDataset.newGroupDatasetX2DY("First", x1,
y1);
Dataset1.setStackMode(QCChartTS.ChartConstants.AUTOAXES_STACKED);
let pTransform1: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setScaleStartX(0);
pTransform1.setScaleStartY(0);

.
.
.
let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID);
attrib1.setFillFlag(true);
attrib1.setLineFlag(false);
let attribArray: QCChartTS.ChartAttribute[] = new Array(numGroups);
for (i = 0; (i < numGroups); i++) {
  attribArray[i] = (<QCChartTS.ChartAttribute>(attrib1.clone()));
}

attribArray[0].setFillColor(QCChartTS.ChartColor.BLUE);
attribArray[1].setFillColor(QCChartTS.ChartColor.YELLOW);
attribArray[2].setFillColor(QCChartTS.ChartColor.MAGENTA);
attribArray[3].setFillColor(QCChartTS.ChartColor.ORANGE);
attribArray[4].setFillColor(QCChartTS.ChartColor.GRAY);
attribArray[5].setFillColor(QCChartTS.ChartColor.RED);
attribArray[6].setFillColor(QCChartTS.ChartColor.GREEN);
let thePlot1: QCChartTS.StackedLinePlot =
QCChartTS.StackedLinePlot.newStackedLinePlot(pTransform1, Dataset1, attribArray);
.
.
.
```

[JavaScript]

```
let numPoints = 100;
let numGroups = 7;
let x1 = new Array(numPoints);
let y1 = QCChartTS.ChartSupport.newArray2D(numGroups, numPoints);
let k;
let i;
let j;
for (i = 0; i < numPoints; i++) {
x1[i] = i * 0.2;

    for (i = 0; i < numPoints; i++) {
        x1[i] = i * 0.2;

        for (j = 0; j < numGroups; j++) {
            switch (j) {
                case 0: y1[j][i] = 5 * QCChartTS.ChartSupport.getRandomDouble() + 25.0 * (1.0 -
Math.exp(-x1[i] / 60.0)); break;
                case 1: y1[j][i] = 8 * QCChartTS.ChartSupport.getRandomDouble() + 25.0 * (1.0 -
Math.exp(-x1[i] / 50.0)); break;
```

```
                case 2: y1[j][i] = 13 * QCChartTS.ChartSupport.getRandomDouble() + 35.0 * (1.0 -
Math.exp(-x1[i] / 40.0)); break;
                case 3: y1[j][i] = 4 * QCChartTS.ChartSupport.getRandomDouble() + 15.0 * (1.0 -
Math.exp(-x1[i] / 30.0)); break;
                case 4: y1[j][i] = 9 * QCChartTS.ChartSupport.getRandomDouble() + 35.0 * (1.0 -
Math.exp(-x1[i] / 20.0)); break;
                case 5: y1[j][i] = 3 * QCChartTS.ChartSupport.getRandomDouble() + 15.0 * (1.0 -
Math.exp(-x1[i] / 30.0)); break;
                case 6: y1[j][i] = 7 * QCChartTS.ChartSupport.getRandomDouble() + 25.0 * (1.0 -
Math.exp(-x1[i] / 50.0)); break;
            }
            if ((i >= numPoints / 2 - 7) && (i <= numPoints / 2 + 7)) {
                for (k = 0; k < numGroups; k++)
                    y1[k][i] += (123 - 10 * Math.abs((numPoints / 2) - i)) *
QCChartTS.ChartSupport.getRandomDouble();
            }
        }
    }
theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
let Dataset1 = QCChartTS.GroupDataset.newGroupDatasetX2DY("First", x1, y1);
Dataset1.setStackMode(QCChartTS.ChartConstants.AUTOAXES_STACKED);
let pTransform1 =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setScaleStartX(0);
pTransform1.setScaleStartY(0);
let background = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.PLOT_BACKGROUND, QCChartTS.ChartColor.fromRgb(255, 255, 255));
chartVu.addChartObject(background);
pTransform1.setGraphBorderDiagonal(0.15, .15, .90, 0.75);

.
.
.
let attrib1 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID);
attrib1.setFillFlag(true);
attrib1.setLineFlag(false);
let attribArray = new Array(numGroups);
for (i = 0; (i < numGroups); i++) {
attribArray[i] = ((attrib1.clone()));
}

attribArray[0].setFillColor(QCChartTS.ChartColor.BLUE);
attribArray[1].setFillColor(QCChartTS.ChartColor.YELLOW);
attribArray[2].setFillColor(QCChartTS.ChartColor.MAGENTA);
attribArray[3].setFillColor(QCChartTS.ChartColor.ORANGE);
attribArray[4].setFillColor(QCChartTS.ChartColor.GRAY);
attribArray[5].setFillColor(QCChartTS.ChartColor.RED);
attribArray[6].setFillColor(QCChartTS.ChartColor.GREEN);
let thePlot1 = QCChartTS.StackedLinePlot.newStackedLinePlot(pTransform1, Dataset1, attribArray);
chartVu.addChartObject(thePlot1);
```

12. Buttons and Scroll bars

The default HTML5 Canvas supported by modern browsers does not include any support for integrated scroll bars and buttons. While you can add scroll bars and buttons as HTML elements to the parent HTML page, you cannot add them directly to a canvas. But, the interaction with a chart is enhanced if the scroll bar and button elements are right next to the chart element. To that end we created some simple button and scroll bar control which you can add to a canvas.

Buttons

Class ButtonArea

```
GraphObj
    |
    +--MouseListener
        |
        +--ButtonArea
```

The **ButtonArea** class is subclassed from the MouseListener class. It is drawn in the parent Canvas using standard Canvas drawing functions. It performs the actions of a toggle button, a momentary closure button, and when associated with other buttons, a radio button.

ButtonArea constructor

```
public static newButtonAreaType(component: ChartView, buttontype: number): ButtonArea

public static newButtonAreaPosTypeString(component: ChartView, pos: ChartRectangle2D, buttontype:
number, text: string): ButtonArea

public static newButtonAreaPosTypeStringAttrib(component: ChartView, pos: ChartRectangle2D,
buttontype: number, text: string, buttonattrib: ChartAttribute): ButtonArea
```

Parameters

component	A reference to the ChartView object that the button is placed in.
buttontype	The button type. Use one of the button type constants: MOMENTARYBUTTON, TOGGLEBUTTON, RADIOBUTTON
pos	A reference to the positioning rectangle, specifying the button position and size using chart normalized coordinates.

buttontype	The button type. Use one of the button type constants: MOMENTARYBUTTON, TOGGLEBUTTON, RADIOBUTTON
text	The text string used as the button label.
buttonattrib	The button line and fill style attributes

Here are some properties you can use to set/get the button colors, text, and state.

Selected Public Instance Properties

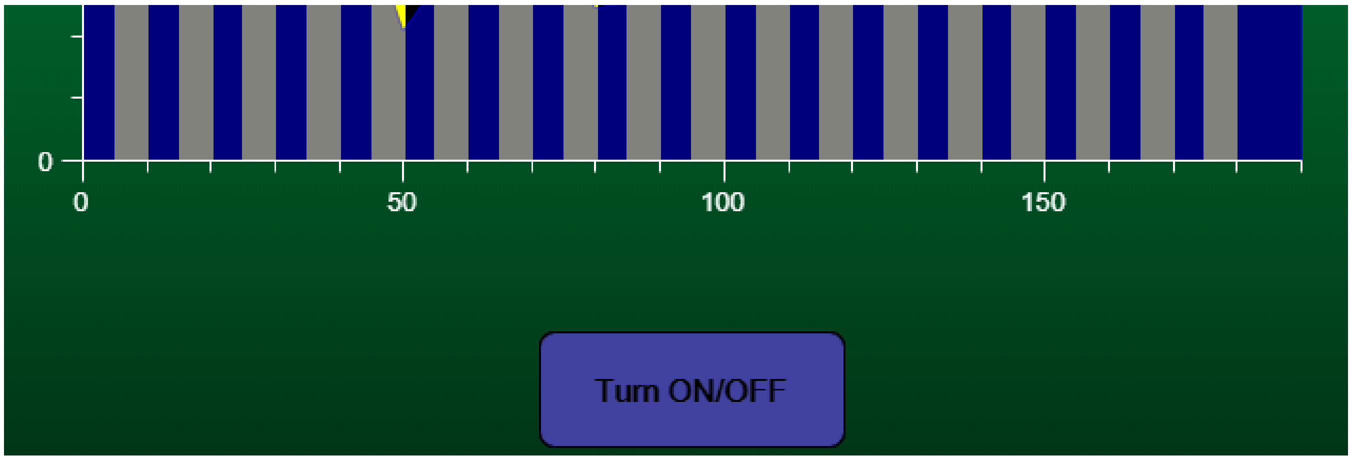
ButtonChecked	Get/Set the button check state.
ButtonCheckedColor	Get/Set the color of the button when the button is checked.
ButtonCheckedText	Get/Set the button text when the button is checked.
ButtonCheckedTextColor	Get/Set the color of the button text when the button is checked.
ButtonSubtype	Get/Set the button subtype. RADIOBUTTON, MOMENTARYBUTTON,TOGGLEBUTTON
ButtonUncheckedColor	Get/Set the color of the button when the button is unchecked.
ButtonUncheckedText	Get/Set the button text when the button is unchecked.
ButtonUncheckedTextColor	Get/Set the color of the button text when the button is unchecked.
RoundedButonCornerRadius	Get/Set the a radius to use on the corners of the button.
ButtonFont	Get/Set the font used for the button text
Position	Specify the position and size of the button using a Chart2DRectangle

When using the RADIOBUTTON button type, you need to associate the buttons in the same Radio button group. This is done using the **setButtonGroupAssociates** function of the first button in the group, passing in an array of all of the button in the group. For example:

```
let groupButton1Array: QCChartTS.ButtonArea[] = [button1, button2, button3];
button1.setButtonGroupAssociates(groupButton1Array); // only need to call on one button in the
group
```

Example for momentary and toggle buttons

The example below, extracted from the DynamicCharts.BuildDynMixedPlot example uses a simple toggle button to turn on/off the chart update.



[TypeScript]

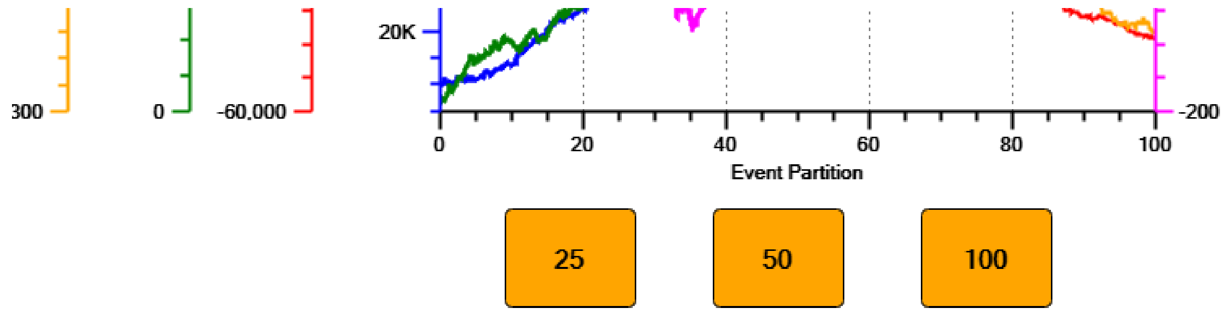
```
let buttonAttrib: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1, 0,
QCChartTS.ChartColor.BLUE);
let buttonFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.BOLD, 16);

let buttonpos: QCChartTS.ChartRectangle2D = QCChartTS.ChartRectangle2D.newChartRectangle2D(0.4,
0.85, 0.2, 0.1);
let button1: QCChartTS.ButtonArea =
QCChartTS.ButtonArea.newButtonAreaPosTypeStringAttrib(chartVu, buttonpos,
QCChartTS.ChartConstants.TOGGLEBUTTON, "Turn ON/OFF", buttonAttrib);
button1.setButtonFont(buttonFont);
button1.RoundedButonCornerRadius = 0.05;
button1.setButtonAreaEventListener(this);
chartVu.addChartObject(button1);
}
```

[JavaScript]

```
let buttonAttrib = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1, 0,
QCChartTS.ChartColor.BLUE);
let buttonFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.BOLD, 16);
// The following button events emulate radio buttons
let buttonpos = QCChartTS.ChartRectangle2D.newChartRectangle2D(0.4, 0.85, 0.2, 0.1);
let button1 = QCChartTS.ButtonArea.newButtonAreaPosTypeStringAttrib(chartVu, buttonpos,
QCChartTS.ChartConstants.TOGGLEBUTTON, "Turn ON/OFF", buttonAttrib);
button1.setButtonFont(buttonFont);
button1.RoundedButonCornerRadius = 0.05;
button1.setButtonAreaEventListener(this);
chartVu.addChartObject(button1);
chartVu.addMouseListener(button1);
```

Example for radio buttons

The example below, extracted from the MultipleAxes.BuildMultiAxes example, uses three radio buttons to select the x-axis scale range.



[TypeScript]

```
let buttonFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.BOLD, 16);

let buttongroup1attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.ORANGE);

let buttonpos: QCChartTS.ChartRectangle2D = QCChartTS.ChartRectangle2D.newChartRectangle2D(0.4,
0.8, 0.1, 0.1);
let button1: QCChartTS.ButtonArea =
QCChartTS.ButtonArea.newButtonAreaPosTypeStringAttrib(this.chartVu, buttonpos,
QCChartTS.ChartConstants.RT_CONTROL_RADIOBUTTON_SUBTYPE, "25", buttongroup1attrib1);
button1.setButtonFont(buttonFont);
button1.RoundedButonCornerRadius = 0.05;

button1.buttonTouchAreaEventListeners = (e: QCChartTS.ButtonAreaEventArgs) => {
this.RescaleAxes(25.0);
} ;// 3 second

this.chartVu.addChartObject(button1);
this.chartVu.addMouseListener(button1);


buttonpos.setFrame(0.56, 0.8, 0.1, 0.1);
let button2: QCChartTS.ButtonArea =
QCChartTS.ButtonArea.newButtonAreaPosTypeStringAttrib(this.chartVu, buttonpos,
QCChartTS.ChartConstants.RT_CONTROL_RADIOBUTTON_SUBTYPE, "50", buttongroup1attrib1);
button2.setButtonFont(buttonFont);
button2.RoundedButonCornerRadius = 0.05;
button2.buttonTouchAreaEventListeners = (e: QCChartTS.ButtonAreaEventArgs) => {
    this.RescaleAxes(50.0);
} ;// 3 second


this.chartVu.addChartObject(button2);
this.chartVu.addMouseListener(button2);

buttonpos.setFrame(0.72, 0.8, 0.1, 0.1);
let button3: QCChartTS.ButtonArea =
QCChartTS.ButtonArea.newButtonAreaPosTypeStringAttrib(this.chartVu, buttonpos,
QCChartTS.ChartConstants.RT_CONTROL_RADIOBUTTON_SUBTYPE, "100", buttongroup1attrib1);
button3.setButtonFont(buttonFont);
button3.RoundedButonCornerRadius = 0.05;
button3.buttonTouchAreaEventListeners = (e: QCChartTS.ButtonAreaEventArgs) => {
    this.RescaleAxes(100.0);
} ;// 3 second

this.chartVu.addChartObject(button3);
this.chartVu.addMouseListener(button3);
```

```
let groupButton1Array: QCChartTS.ButtonArea[] = [button1, button2, button3];
button1.setButtonGroupAssociates(groupButton1Array); // only need to call on one button in the
group

.
.
.
}
```

[JavaScript]

```
let buttonFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.BOLD, 16);
let buttongroup1attrib1 = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK,
1, QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.ORANGE);
let buttonpos = QCChartTS.ChartRectangle2D.newChartRectangle2D(0.4, 0.8, 0.1, 0.1);
let button1 = QCChartTS.ButtonArea.newButtonAreaPosTypeStringAttrib(this.chartVu, buttonpos,
QCChartTS.ChartConstants.RT_CONTROL_RADIOBUTTON_SUBTYPE, "25", buttongroup1attrib1);
button1.setButtonFont(buttonFont);
button1.RoundedButonCornerRadius = 0.05;
button1.buttonTouchAreaEventListeners = (e) => {
    this.RescaleAxes(25.0);
}; // 3 second
this.chartVu.addChartObject(button1);
this.chartVu.addMouseListener(button1);
buttonpos.setFrame(0.56, 0.8, 0.1, 0.1);
let button2 = QCChartTS.ButtonArea.newButtonAreaPosTypeStringAttrib(this.chartVu, buttonpos,
QCChartTS.ChartConstants.RT_CONTROL_RADIOBUTTON_SUBTYPE, "50", buttongroup1attrib1);
button2.setButtonFont(buttonFont);
button2.RoundedButonCornerRadius = 0.05;
button2.buttonTouchAreaEventListeners = (e) => {
    this.RescaleAxes(50.0);
}; // 3 second
this.chartVu.addChartObject(button2);
this.chartVu.addMouseListener(button2);
buttonpos.setFrame(0.72, 0.8, 0.1, 0.1);
let button3 = QCChartTS.ButtonArea.newButtonAreaPosTypeStringAttrib(this.chartVu, buttonpos,
QCChartTS.ChartConstants.RT_CONTROL_RADIOBUTTON_SUBTYPE, "100", buttongroup1attrib1);
button3.setButtonFont(buttonFont);
button3.RoundedButonCornerRadius = 0.05;
button3.buttonTouchAreaEventListeners = (e) => {
    this.RescaleAxes(100.0);
}; // 3 second
this.chartVu.addChartObject(button3);
this.chartVu.addMouseListener(button3);
let groupButton1Array = [button1, button2, button3];
button1.setButtonGroupAssociates(groupButton1Array); // only need to call on one button in the
group
```

Scroll bar

Class ScrollbarArea

```
GraphObj
   |
   +--MouseListener
         |
         +--ScrollbarArea
```

The **ScrollbarArea** class is subclassed from the MouseListener class. It is drawn in the parent Canvas using standard Canvas drawing function. It performs the actions of a simple horizontal or vertical scroll bar.

ScrollbarArea constructor

```
public static newScrollbarArea(component: ChartView, orientation: number): ScrollbarArea

public static newScrollbarAreaOrientMinMax(component: ChartView, orientation: number, scrollmin: 
number, scrollmax: number): ScrollbarArea

public static newScrollbarAreaOrientMinMaxValueAttrib(component: ChartView, orientation: number, 
scrollmin: number, scrollmax: number, initialvalue: number, scrollareaattrib: ChartAttribute): 
ScrollbarArea

public static newScrollbarAreaOrientMinMaxValue(component: ChartView, orientation: number, 
scrollmin: number, scrollmax: number, initialvalue: number): ScrollbarArea

public static newScrollbarAreaOrientMinMaxValuePos(component: ChartView, orientation: number, 
scrollmin: number, scrollmax: number, initialvalue: number, pos: ChartRectangle2D): ScrollbarArea
```

Parameters

component	A reference to the ChartView object that the scroll is placed in.
orientation	Specifies the horizontal or vertical orientation of the scroll bar. Use the chart constant HORIZ_DIR or VERT_DIR.
scrollmin	Specifies the minimum value for the scroll bar.
scrollmax	Specifies the maximum value for the scroll bar.
initialvalue	Specifies the initial position of the scroll bar.
scrollareaattrib	Specifies the color attributes of the scroll bar.
pos	A reference to the positioning rectangle, specifying the button position and size using chart normalized coordinates.

Selected Public Instance Properties

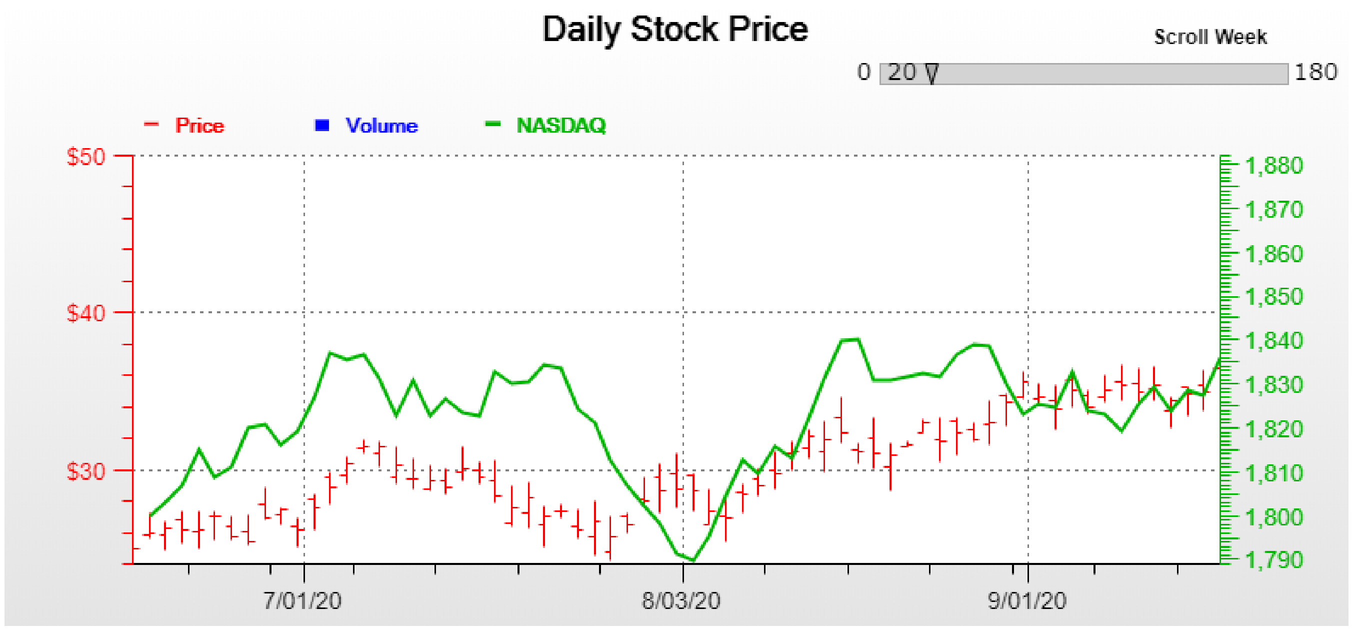
Orientation	Gets or sets a value indicating the horizontal or vertical orientation (**ChartConstants.HORIZ_DIR** or **ChartConstants.VERT_DIR**) of the scroll bar.
Maximum	Specifies the floating point maximum value for the scroll bar.
Minimum	Specifies the floating point minimum value for

	the scroll bar.
Value	Specifies the numeric value of the scroll bar.
Position	Specify the position and size of the scroll bar using a Chart2DRectangle

Example for independently positioned horizontal scroll bar

The example below, extracted from the OHLCChart.BuildOHLC example uses a simple scroll bar to pan the chart data back and forth.

By default, if the scroll bar is specified as horizontal, it is placed at the bottom of the Canvas. If the scroll bar is specified as vertical, it is placed on the right edge of the canvas. You can change the default position using the Position property, as seen below.



[TypeScript]

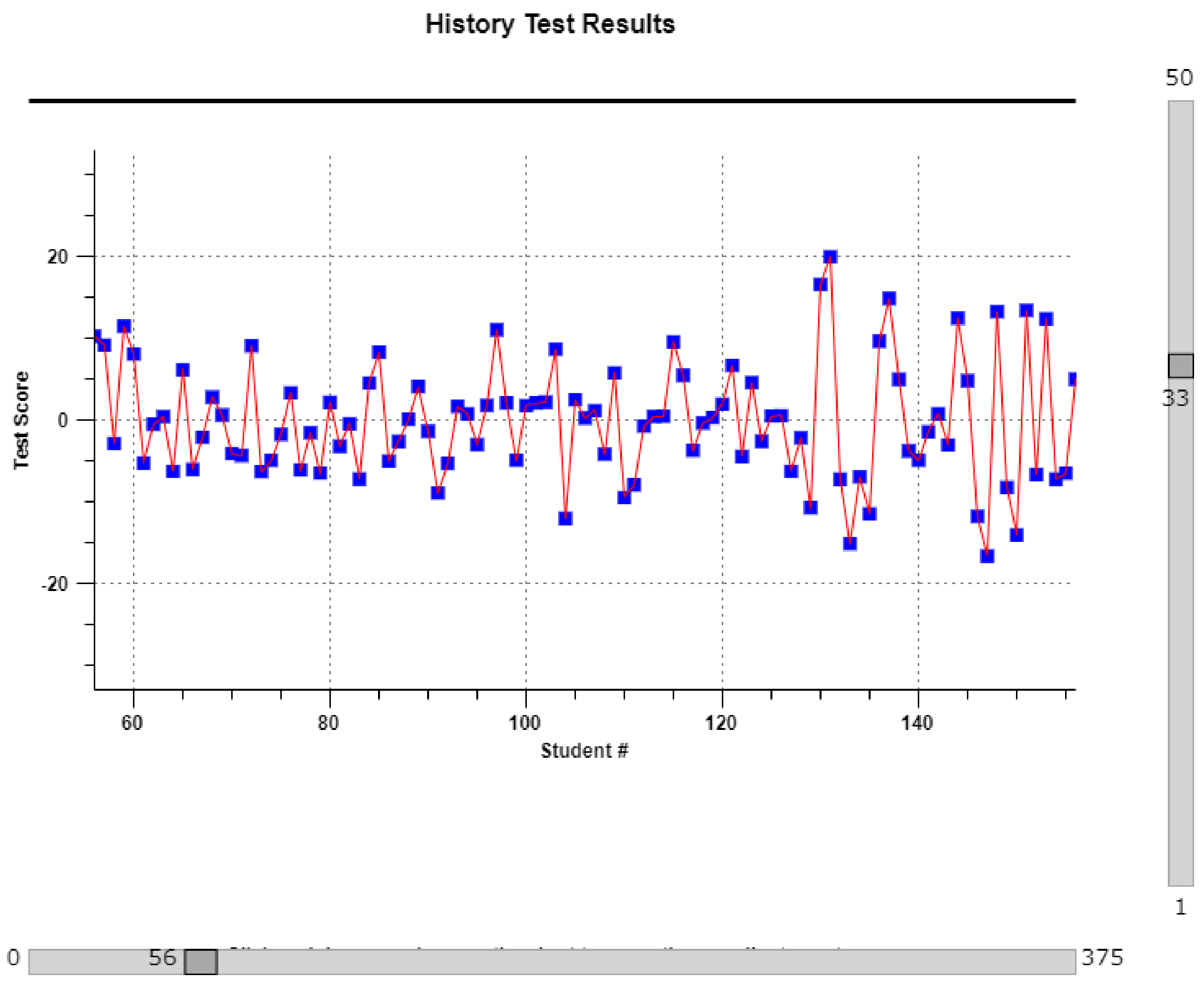
```
let scrollbarAttributes: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.GRAY, 1,
QCChartTS.ChartConstants.LS_SOLID);

let scrollbar: QCChartTS.ScrollbarArea =
QCChartTS.ScrollbarArea.newScrollbarAreaOrientMinMaxValueAttrib(this.chartVu,
QCChartTS.ChartConstants.HORIZ_DIR, 0, 180, 20, scrollbarAttributes);
scrollbar.setPosition(QCChartTS.ChartRectangle2D.newChartRectangle2D(0.65, 0.06, 0.3, 0.02));
this.chartVu.addChartObject(scrollbar);
this.chartVu.setCurrentMouseListener(scrollbar);

// You assign a scrollbar event listener this way, using the anonymouse fat arrow syntax
scrollbar.ScrollbarAreaEventListener = (args: QCChartTS.ScrollbarAreaEventArgs) => {
    let index: number = Math.round(args.getScrollValue());
    this.UpdateScaleAndAxes(index);
}
```

[JavaScript]

```
let scrollbarAttributes = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.GRAY,
1, QCChartTS.ChartConstants.LS_SOLID);
let scrollbar = QCChartTS.ScrollbarArea.newScrollbarAreaOrientMinMaxValueAttrib(this.chartVu,
QCChartTS.ChartConstants.HORIZ_DIR, 0, 180, 20, scrollbarAttributes);
scrollbar.setPosition(QCChartTS.ChartRectangle2D.newChartRectangle2D(0.65, 0.06, 0.3, 0.02));
this.chartVu.addChartObject(scrollbar);
this.chartVu.setCurrentMouseListener(scrollbar);
// You assign a scrollbar event listener this way, using the anonymouse fat arrow syntax
scrollbar.ScrollbarAreaEventListener = (args) => {
    let index = Math.round(args.getScrollValue());
    this.UpdateScaleAndAxes(index);
};
```

Example for horizontal and vertical scroll bars in default positions

The example below, extracted from the MouseListeners.BuildLinePlotScrollBar uses the default horizontal and vertical scroll bar to control the chart. In this case, it doesn't need to explicitly create the scroll bar objects, as a default horizontal and vertical scroll bar are already part of the ChartView. They only need to be turned on and properties set.



[TypeScript]

```
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);

}
}
```

[JavaScript]

```
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);
}
```

13. Data Markers and Data Cursors

Marker

Data markers are symbols and lines that can be "dropped" on to the data presented in a graph, much like a bookmark in a word processing document. Place the markers in a chart under program control or in response to a mouse event in the graph window.

Data cursors are temporary lines or symbols that are used to help position the mouse cursor over the desired section of a graph. Standard data cursors include cross hairs, a box, and horizontal and/or vertical lines.

Note: We use the term mouse throughout the manual when it would be more accurate to use mouse/touch. In order to simplify programming for mouse applications (desktop) and touch applications (phone and tablet) our **MouseListener** class processes both mouse and touch events in one core set of event handlers (onMouseDown, onMouseMove, onMouseUp, and onClick events). That way you/we don't have to constantly write two sets of code, one for mouse driven applications and one for touch driven applications. While the software has some button processing built-in for left, right and middle buttons, in order to maintain compatibility with touch devices, should refrain for using anything other than the left button. The right button is hijacked by the browser for context-sensitive menu options and we found no straightforward means of disabling it which works across all browser.

Data Markers

Class Marker

GraphObj

```
   |
   +--Marker
```

Create data markers using the **Marker** class. The constructor below creates a new **Marker** object using the specified coordinate system, marker type, marker position and marker size.

Marker constructor

```
public static newMarker(transform: PhysicalCoordinates, nmarkertype: number, x: number, y:
number, rsize: number, npostype: number): Marker

public static newMarkerCoords(transform: PhysicalCoordinates): Marker
```

transform	Places the marker in the coordinate system defined by transform.
nmarkertype	Specifies the shape of the current chart marker. Use one of the chart marker constants: MARKER_NULL, MARKER_VLINE,

	MARKER_HLINE, MARKER_CROSS, MARKER_BOX or MARKER_HVLINE.
x	Specifies the x-value of the marker position
y	Specifies the y-value of the marker position
rsize	Specifies the size of the cross hair marker (MARKER_CROSS) and the box marker (MARKER_BOX) in Canvas device coordinates.
npostype	Specifies the if the position of the marker is specified in physical coordinates, normalized coordinates or Canvas device coordinates. Use one of the position constants: DEV_POS, PHYS_POS, NORM_GRAPH_POS, NORM_PLOT_POS.

The marker constants signify:

MARKER_NULL	An invisible marker
MARKER_VLINE	The marker is a vertical line extending from the top of the plot area to the bottom, passing through the x-value of the marker position.
MARKER_HLINE	The marker is a horizontal line extending from the left of the plot area to the right, passing through the y-value of the marker position.
MARKER_HVLINE	The marker combines both MARKER_VLINE and MARKER_HLINE, marking the data point with horizontal and vertical lines.
MARKER_CROSS	The marker is a cross hair centered on the marker position. Set the size of the cross hair using Canvas device coordinates, in the object constructor, or later using the **setMarkerSize** method.
MARKER_BOX	The marker is a box centered on the marker position. Set the size of the box using Canvas device coordinates, in the object constructor, or later using the **setMarkerSize** method.

Drop a marker anywhere on a plot by specifying the coordinates. The example below places a 10 pixel wide marker in the center of the plot area.

Simple marker example

[TypeScript]

```
let pTransform1:  QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 10, 20);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
let xpos: number = 5;
let ypos: number = 10;
let amarker:  QCChartTS.Marker = QCChartTS.Marker.newMarker(pTransform1,
QCChartTS.ChartConstants.MARKER_BOX, xpos, ypos, 10, QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(amarker);
```

[JavaScript]

```
let pTransform1 = QCChartTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 10, 20);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
let xpos = 5;
let ypos = 10;
let amarker = QCChartTS.Marker.newMarker(pTransform1, QCChartTS.ChartConstants.MARKER_BOX, xpos,
ypos, 10, QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(amarker);
```

Data Cursors

Class DataCursor

MouseListener

```
    |
    + -- DataCursor
```

Data cursors are an extension of the MouseListener class. Data cursors combine the mouse event delegates with the **Marker** class, creating a marker that tracks the mouse and updates dynamically. This constructor creates a new **DataCursor** object using the specified coordinate system, marker type and marker size.

DataCursor constructor

```
public static newDataCursor(component: ChartView, transform: PhysicalCoordinates, nmarkertype:
number, rsize: number): DataCursor
```

component	A reference to the **ChartView** object that the chart is placed in.
transform	The **PhysicalCoordinates** object associated with the data cursor.
nmarkertype	The marker type. Use one of the **Marker** marker type constants: MARKER_VLINE .. MARKER_ BOX.

rsize	The size in Canvas device coordinates of the MARKER_BOX and MARKER_CROSS style cursors.

See the **Marker** constructor description for more information about the **Marker** type constants.

Create the **DataCursor** object and then install it using the **ChartView.setCurrentMouseListener** method. This adds the **DataCursor** object as a **MouseListener** to the **ChartView** object. Enable/Disable the function using **DataCursor.setEnable** method. Call the **ChartView removeMouseListener(mouselistener)** to remove the object as a mouse listener for the chart view.

Since the **DataCursor** class implements mouse event delegates, it has methods implementing the mouse/touch events onMouseDown, onDoubleClick, onClick, onMouseDown, and onMouseUp. The default usage of the **DataCursor** class creates the data marker when the mouse is pressed. As long as the mouse is pressed, the data cursor tracks the mouse position. Release the mouse button and the marker disappears. When using data markers it is often desirable to do additional things during the mouse/touch events. In this case, you derive a new class from **DataCursor**, override the mouse/touch events that you want to intercept, and add your own code to these events. Make sure you call the parents version of the same mouse event function so that the data cursor continues to track the mouse.

Simple data cursor example (Adapted from the DataCursorsAndMarkers example)

[TypeScript]

```
export class CustomChartDataCursor extends QCChartTS.DataCursor {


    public constructor() {
        super();
    }

    public static newCustomChartDataCursor(achartview: QCChartTS.ChartView, thetransform:
QCChartTS.CartesianCoordinates,
        nmarkertype: number, rsize: number): CustomChartDataCursor {
        let result: CustomChartDataCursor = new CustomChartDataCursor();
        result.initChartDataCursor(achartview, thetransform, nmarkertype, rsize);

        return result;

    }

    public onMouseUp(mouseevent: MouseEvent) {


    }
}
.
```

```
.
.

let dataCursorObj: CustomChartDataCursor =
CustomChartDataCursor.newCustomChartDataCursor(this.chartVu, pTransform1,
QCChartTS.ChartConstants.MARKER_HVLINE, 8);
dataCursorObj.setEnable(true);
this.chartVu.setCurrentMouseListener(dataCursorObj);
```

[JavaScript]

```
export class CustomChartDataCursor extends QCChartTS.DataCursor {
    constructor() {
        super();

    }

    static newCustomChartDataCursor(achartview, thetransform, nmarkertype, rsize) {
        let result = new CustomChartDataCursor();
        result.initChartDataCursor(achartview, thetransform, nmarkertype, rsize);

        return result;
    }
    onMouseUp(mouseevent) {

    }
}
.
.
.
let dataCursorObj = CustomChartDataCursor.newCustomChartDataCursor(this.chartVu, pTransform1,
CChartTS.ChartConstants.MARKER_HVLINE, 8);
dataCursorObj.setEnable(true);
this.chartVu.setCurrentMouseListener(dataCursorObj);
```

A marker can be placed at any xy coordinate location in a graph. It is often desirable to place a marker at the exact location of a data point in one of the datasets plotted in the graph. Many applications require the user to click on the approximate location of a point, and then the software must find the data point nearest that click and mark it. The **DataCursor** and **Marker** classes, in combination with the plot objects calcNearestPoint methods, accomplish this. The **DataCursor** class positions the mouse cursor and retrieves the initial xy coordinates. The calcNearestPoint method for each plot object (**SimpleLinePlot**, **ScatterPlot**, etc.) in the graph determines the nearest data point to the mouse cursor for that object. Once all the plot objects are checked the data point nearest the mouse cursor position is marked by placing a **Marker** object at that exact xy location.

The example below extends the previous **Marker** and **DataCursor** examples. In this example, the onMouseUp event of the subclassed **DataCursor** object processes the plot objects looking for the nearest point, and then places a **Marker** object and a numeric label at that point.

Marking a data point (Adapted from the DataCursorView and CustomChartDataCursor classes)

[TypeScript]

```
export class CustomChartDataCursor extends QCChartTS.DataCursor {

    pointLabel: QCChartTS.NumericLabel = new QCChartTS.NumericLabel();
    textCoordsFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont("Microsoft Sans
Serif", QCChartTS.ChartFont.REGULAR, 10);
    rNumericLabelCntr: number = 0;
    thePlot1: QCChartTS.SimpleLinePlot = new QCChartTS.SimpleLinePlot();
    thePlot2: QCChartTS.SimpleLinePlot = new QCChartTS.SimpleLinePlot();;

    public constructor() {
        super();
    }

    public static newCustomChartDataCursor(achartview: QCChartTS.ChartView, thetransform:
QCChartTS.CartesianCoordinates, plot1: QCChartTS.SimpleLinePlot, plot2: QCChartTS.SimpleLinePlot,
        nmarkertype: number, rsize: number): CustomChartDataCursor {
        let result: CustomChartDataCursor = new CustomChartDataCursor();
        result.initChartDataCursor(achartview, thetransform, nmarkertype, rsize);
        result.thePlot1 = plot1;
        result.thePlot2 = plot2;
        return result;

    }

    public onMouseUp(mouseevent: MouseEvent) {
        let nearestPointObj1: QCChartTS.NearestPointData = new QCChartTS.NearestPointData();
        let nearestPointObj2: QCChartTS.NearestPointData = new QCChartTS.NearestPointData();
        let nearestPoint: QCChartTS.ChartPoint2D = QCChartTS.ChartPoint2D.newChartPoint2D(0, 0);
        let chartview: QCChartTS.ChartView | null = this.getChartObjComponent();
        if (!chartview) return;
        let bfound1: boolean = false;
        let bfound2: boolean = false;
        super.onMouseUp(mouseevent);
        if (this.getEnable()) {
            //  Find nearest point for each line plot object
            let location: QCChartTS.ChartPoint2D = this.getLocation();
            bfound1 = this.thePlot1.calcNearestPoint(location,
QCChartTS.ChartConstants.FNP_NORMDIST, nearestPointObj1);
            bfound2 = this.thePlot2.calcNearestPoint(location,
QCChartTS.ChartConstants.FNP_NORMDIST, nearestPointObj2);
            if ((bfound1 && bfound2)) {
                //  choose the nearest point
                if ((nearestPointObj1.getNearestPointMinDistance() <
nearestPointObj2.getNearestPointMinDistance())) {
                    nearestPoint = nearestPointObj1.getNearestPoint();
                }
                else {
                    nearestPoint = nearestPointObj2.getNearestPoint();
                }

                //  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();
            }

        }

    }
}
.
.
.

let dataCursorObj: CustomChartDataCursor =
CustomChartDataCursor.newCustomChartDataCursor(this.chartVu, pTransform1, this.thePlot1,
this.thePlot2, QCChartTS.ChartConstants.MARKER_HVLINE, 8);
dataCursorObj.setEnable(true);
this.chartVu.setCurrentMouseListener(dataCursorObj);
```

[JavaScript]

```
export class CustomChartDataCursor extends QCChartTS.DataCursor {
    constructor() {
        super();
        this.pointLabel = new QCChartTS.NumericLabel();
        this.textCoordsFont = QCChartTS.ChartFont.newChartFont("Microsoft Sans Serif",
QCChartTS.ChartFont.REGULAR, 10);
        this.rNumericLabelCntr = 0;
        this.thePlot1 = new QCChartTS.SimpleLinePlot();
        this.thePlot2 = new QCChartTS.SimpleLinePlot();
    }
    ;
    static newCustomChartDataCursor(achartview, thetransform, plot1, plot2, nmarkertype, rsize) {
        let result = new CustomChartDataCursor();
        result.initChartDataCursor(achartview, thetransform, nmarkertype, rsize);
        result.thePlot1 = plot1;
        result.thePlot2 = plot2;
        return result;
    }
    onMouseUp(mouseevent) {
        let nearestPointObj1 = new QCChartTS.NearestPointData();
        let nearestPointObj2 = new QCChartTS.NearestPointData();
        let nearestPoint = QCChartTS.ChartPoint2D.newChartPoint2D(0, 0);
        let chartview = this.getChartObjComponent();
        if (!chartview)
            return;
        let bfound1 = false;
        let bfound2 = false;
        super.onMouseUp(mouseevent);
        if (this.getEnable()) {
            //  Find nearest point for each line plot object
            let location = this.getLocation();
            bfound1 = this.thePlot1.calcNearestPoint(location,
QCChartTS.ChartConstants.FNP_NORMDIST, nearestPointObj1);
            bfound2 = this.thePlot2.calcNearestPoint(location,
QCChartTS.ChartConstants.FNP_NORMDIST, nearestPointObj2);
            if ((bfound1 && bfound2)) {
                //  choose the nearest point
                if ((nearestPointObj1.getNearestPointMinDistance() <
nearestPointObj2.getNearestPointMinDistance())) {
                    nearestPoint = nearestPointObj1.getNearestPoint();
                }
                else {
                    nearestPoint = nearestPointObj2.getNearestPoint();
                }
                //  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();
            }
        }
    }
}
.
.
.
let dataCursorObj = CustomChartDataCursor.newCustomChartDataCursor(this.chartVu, pTransform1,
this.thePlot1, this.thePlot2, QCChartTS.ChartConstants.MARKER_HVLINE, 8);
dataCursorObj.setEnable(true);
this.chartVu.setCurrentMouseListener(dataCursorObj);
```

Another common reason for locating a data point is to display information associated with that data point. A good example is stock market data. A typical stock market display is a one-month chart of daily closing values for one or more stocks. You want to be able to click on a point in the chart and have the open, high, low and closing value for that day displayed in a pop-up box. The example program OHLCChart demonstrates how to use a **ChartText** object as a popup box to display this type of data.

14. Moving Chart Objects, Data Points and Coordinate Systems

MoveObj
MoveData
MoveCoordinates

Many of the subclasses of **GraphObj** are moveable using the mouse. This includes the axis, legend, text, image, shape classes. If you add the necessary support to your program, you can click and drag the object around in the chart. This may or not be desirable, since a user can ruin a carefully constructed chart by dragging objects around. It is just an option though, that you can add to the program.

It is also possible to select a single data point in a simple plot object (**SimpleLinePlot**, **SimpleBarPlot**, **SimpleLineMarkerPlot** and **SimpleScatterPlot**) and move it with a click and drag operation of the mouse. Again, it is an option that you can add to the program if you want.

You can also move the coordinates system of a graph, using the **MoveCoordinates** class. The move operation is analogous to the way you can change the latitude and longitude of an internet map by clicking and dragging it.

Note: We use the term mouse throughout the manual when it would be more accurate to use mouse/touch. In order to simplify programming for mouse applications (desktop) and touch applications (phone and tablet) our **MouseListener** class processes both mouse and touch events in one core set of event handlers (onMouseDown, onMouseMove, onMouseUp, and onClick events). That way you/we don’t have to constantly write two sets of code, one for mouse driven applications and one for touch driven applications. While the software has some button processing built-in for left, right and middle buttons, in order to maintain compatibility with touch devices, should refrain for using anything other than the left button. **The right button is hijacked by the browser for context-sensitive menu options and we found no straightforward means of disabling it which works across all browser.**

Moving Chart Objects

Class MoveObj

```
MouseListener
        |
        +--MoveObj
```

The **MoveObj** mouse listener traps a mouse pressed event and then searches through all of the **GraphObj** derived objects in the view. A rectangle highlights the first object that meets the filter criteria and intersects the mouse cursor. Hold the mouse button down and the rectangle tracks the mouse. Release the mouse button and the position of the graph object updates to reflect the new physical coordinates of the bounding rectangle.

If no *objectfilter* parameter is specified, the default object filter is “GraphObj”.

MoveObject constructors

```
public static newMoveObj(component: ChartView): MoveObj

public static newMoveObjFilter(component: ChartView, object1filter: string): MoveObj
```

component	A reference to the **ChartView** object that the chart is placed in.
objectfilter	The fully qualified class name of the base class that is used to filter the desired class objects. The string "ChartText" causes the routine to move only objects derived from the **ChartText** class. If you want to move only specific objects of a given class, create a special subclass of that class. Then create your moveable objects using that subclass. Then specify your class name, i.e. **MyTextClass,** using the string “MyTextClass”.

Create the **MoveObj** object and then install it using the **ChartView.setCurrentMouseListener** method. This adds the **MoveObj** object as a **MouseListener** to the **ChartView** object. Enable/Disable the function using **MoveObj.setEnable** method. Call the **ChartView removeMouseListener(mouselistener)** to remove the object as a mouse listener for the chart view.

Not all **GraphObj** derived object are moveable. Call the **GraphObj.getMoveableType** method and check to see if it returns **ChartObj.OBJECT_MOVEABLE**. Alternatively, you can call the **MoveObj.isMoveableObject** method, passing in a reference to the object.

Most moveable objects move unrestricted in the x- and y direction. There are exceptions though. Axis objects move in the direction parallel to their current position, effectively changing the axis intercept, but not the extents of the axis endpoints. Axis labels always track their reference axis. The base axis defines the position of an **AxisTitle** text object and the chart view defines the position of a **ChartTitle** text object. Attempt to move these objects and they revert to their original centered positions. If you require moveable chart and axis titles, use the generic **ChartText** class instead of the title classes.

Moving objects example (Adapted from the MouseListeners.MoveObjects class)

[TypeScript]

```
let pTransform1: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 10, 20);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);

.
.
.

let mouselistener: QCChartTS.MoveObj = QCChartTS.MoveObj.newMoveObj(chartVu);
mouselistener.setEnable(true);
mouselistener.setMoveObjectFilter("GraphObj");
chartVu.setCurrentMouseListener(mouselistener);
```

[JavaScript]

```
let pTransform1 = QCChartTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 10, 20);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
.
.
.

let mouselistener = QCChartTS.MoveObj.newMoveObj(chartVu);
mouselistener.setEnable(true);
mouselistener.setMoveObjectFilter("GraphObj");
chartVu.setCurrentMouseListener(mouselistener);
```

Moving Simple Plot Object Data Points

Class MoveData

MouseListener

```
    |
    +--MoveData
```

The **MoveData** mouse listener traps a mouse pressed event and searches through all of the data points of the plot objects in the view. The data point closest to the mouse cursor location is compared against a threshold value, 10 device units, or pixels, by default. If the data point is within the threshold, and as long as the moue buttons is held down, it tracks the mouse. Release the mouse button and the data value associated with the selected data point updates to reflect the new physical coordinates of the data point, and the plot is redrawn. Since the algorithm searches through every data point of every plot object in the view, do not expect it to work particularly fast with millions or even thousands of data points. The practical number of data points that can be searched is obviously dependent on the speed of the host computer.

MoveData constructors

```
public static newMoveData(component: ChartView, transform: PhysicalCoordinates): MoveData
```

component	A reference to the **ChartView** object that the chart is placed in.
transform	The **PhysicalCoordinates** object associated with the **MoveData** object.

Create the **MoveData** object and then install it using the **ChartView.setCurrentMouseListener** method. This adds the **MoveData** object as a **MouseListener** to the **ChartView** object. Enable/Disable the function using **MoveData.setEnable** method. Call the **ChartView removeMouseListener(mouselistener)** to remove the object as a mouse listener for the chart view. Set the threshold distance for deciding if the nearest data point found is a “hit” using the **MoveData.setHitTestThreshold** method.

Moving datapoints example (See the class MouseListeners.MoveDatapoints for a more complicated example)

[TypeScript]

```
let pTransform1: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 10, 20);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
.
.
.
let mouselistener: QCChartTS.MoveData = QCChartTS.MoveData.newMoveData(chartVu, pTransform1);
mouselistener.setMarkerType( QCChartTS.ChartConstants.MARKER_CROSS);
mouselistener.setMarkerSize(12);
mouselistener.setMoveMode(QCChartTS.ChartConstants.MOVE_Y);
mouselistener.setEnable(true);
chartVu.setCurrentMouseListener(mouselistener);
```

[JavaScript]

```
let pTransform1 = QCChartTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 10, 20);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
.
.
.
let mouselistener = QCChartTS.MoveData.newMoveData(chartVu, pTransform1);
mouselistener.setMarkerType(QCChartTS.ChartConstants.MARKER_CROSS);
mouselistener.setMarkerSize(12);
mouselistener.setMoveMode(QCChartTS.ChartConstants.MOVE_Y);
mouselistener.setEnable(true);
```

```
chartVu.setCurrentMouseListener(mouselistener);
```

Moving the Chart Coordinate System

Class MoveCoordinates

```
MouseListener
   |
   +--DataCursor
         |
         +--MoveCoordinates
```

The **MoveCoordinates** mouse listener traps a mouse pressed event. While the mouse is button is held down, the underlying coordinate system will track the move movements. Release the mouse button and the chart redraws one final time using the extents of the final coordinate system.

MoveCoordinates constructors

```
public static newMoveCoordinates(component: ChartView, transform: PhysicalCoordinates):
MoveCoordinates
```

```
public static newMoveCoordinatesCoordsArray(component: ChartView, transforms:
PhysicalCoordinates[]): MoveCoordinates
```

component	A reference to the **ChartView** object that the chart is placed in.
transform	The coordinate system underlying the chart.

Create the **MoveCoordinates** object and then install it using the **ChartView.setCurrentMouseListener** method. This adds the **MoveCoordinates** object as a **MouseListener** to the **ChartView** object. Enable/ Disable the function using MoveCoordinates**.setEnable** method. Call the **ChartView removeMouseListener(mouselistener)** to remove the object as a mouse listener for the chart view.

You can restrict the movement of the coordinate system to the x-dimension (`MOVE_X`), or the y-dimension (`MOVE_Y`), using the **MoveMode** property. Set **MoveMode** to MOVE_XY for unrestricted movement.

If you have multiple coordinate systems in the chart, you can move them all simultaneously by setting the **MultiTransformMove** property true. Otherwise, only the coordinate system passed into the constructor is updated with the move information.

Moving the coordinate system (Extracted from the MouseListeners.MoveCoordinates class)

[TypeScript]

```
let Dataset1: QCChartTS.TimeSimpleDataset =
QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("First", tradingDay, stockPrice);
let pTransform1: QCChartTS.TimeCoordinates = new QCChartTS.TimeCoordinates();
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
.
.
.
let movecoords: QCChartTS.MoveCoordinates = QCChartTS.MoveCoordinates.newMoveCoordinates(chartVu,
pTransform1);
movecoords.setEnable(true);
chartVu.setCurrentMouseListener(movecoords);
```

[JavaScript]

```
let Dataset1 = QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("First", tradingDay, stockPrice);
let pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_NEAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
.
.
.
let movecoords = QCChartTS.MoveCoordinates.newMoveCoordinates(chartVu, pTransform1);
movecoords.setEnable(true);
chartVu.setCurrentMouseListener(movecoords);
```

15. Zooming

ChartZoom

Zooming is the interactive re-scaling of a charts physical coordinate system and the related axes based on limits defined by clicking and dragging a mouse inside the current graph window. A typical use of zooming is in applications where the initial chart displays a large number of data points. The user interacts with the chart, defining smaller and smaller zoom rectangles, zeroing in on the region of interest. The final chart displays axis limits that have a very small range compared to the range of the original, un-zoomed, chart.

Important zoom features include:

- Automatic recalculation of axis properties for tick mark spacing and axis labels..
- Zooming of time coordinates with smooth transitions between major scale changes: years->months->weeks->days->hours->minutes->seconds.
- Zooming of time coordinates that use a 5-day week and a non-24 hour day.
- Simultaneous zooming of an unlimited number of x- and y-coordinate systems and axes (super zooming).
- The user can recover previous zoom levels using a zoom stack.
- The zoomed coordinate system can be forced to maintain a fixed aspect ratio.
- User-defineable zoom limits prevent numeric under and overflows

Simple Zooming of a single physical coordinate system

Class ChartZoom

```
MouseListener
   |
   +--ChartZoom
```

The **ChartZoom** class implements delegates for mouse/touch events. It implements and uses the mouse/touch events: onMouseDown, onDoubleClick, onMouseDown, onMouseUp and onClick. The default operation of the **ChartZoom** class starts the zoom operation on the onMouseDown event; it draws the zoom rectangle during the onMouseDown event; and terminates the zoom operation on the mouse released event. During the mouse released event, the zoom rectangle is converted from device units into the chart physical coordinates and this information is stored and optionally used to rescale the chart scale and all axis objects that reference the chart scale. If four axis objects reference a single chart scale, for example when axes bound a chart on all for sides, all four axes re-scale to match the new chart scale.

Note: We use the term mouse throughout the manual when it would be more accurate to use mouse/touch. In order to simplify programming for mouse applications (desktop) and touch applications (phone and tablet) our MouseListener class processes both mouse and touch events in one core set of event handlers (onMouseDown, onMouseMove, onMouseUp, and onClick events). That way you/we don't have to constantly write two sets of code, one for mouse driven applications and one for touch driven applications. While the software has some button processing built-in for left, right and middle buttons, in order to maintain compatibility with touch devices, should refrain for using anything other than the left button. The right button is hijacked by the browser for context-sensitive menu options and we found no straightforward means of disabling it which works across all browser.

ChartZoom constructor

The constructor below creates a zoom object for a single chart coordinate system.

```
public static newChartZoom(component: ChartView, transform: PhysicalCoordinates, brescale:
boolean): ChartZoom
```

component	A reference to the **ChartView** object that the chart is placed in.
transform	The **PhysicalCoordinates** object associated with the scale being zoomed.
brescale	True designates that the scale should be re-scaled, once the final zoom rectangle is ascertained.

Enable the zoom object after creation using the **ChartZoom.setEnable(true)** method.

Retrieve the physical coordinates of the zoom rectangle using the **ChartZoom** getZoomMin and getZoomMax methods. Restrict zooming in the x- or y-direction using the setZoomXEnable and setZoomYEnable methods. Set the rounding mode associated with rescale operations using the setZoomXRoundMode and setZoomYRoundMode methods. Call the **ChartZoom.popZoomStack** method at any time and the chart scale reverts to the minimum and maximum values of the previous zoom operation. Repeated calls to the popZoomStack method return the chart scale is to its original condition, after which the popZoomStack method has no effect.

Integrated zoom stack processing

.

```
zoomObj.InternalZoomStackProcesssing = true;
```

Return to a previous zoom level by left clicking the mouse (or a quick tap for a touch device).

Aspect Ratio Correction
Starting with Revision 2.0, you can force the zoom rectangle to maintain a fixed aspect ratio. Use the ChartZoom.ArCorrectionMode property to specify the aspect ratio correction mode.

ZOOM_NO_AR_CORRECTION	Allow the x- and y-dimension of the zoom rectangle to change the overall charts physical aspect ratio. This is the default mode, and the only mode supported prior to Revision 2.0.
ZOOM_X_AR_CORRECTION	Track the x-dimension of the zoom rectangle and calculate the y-dimension in order to maintain a fixed aspect ratio.
ZOOM_Y_AR_CORRECTION	Track the y-dimension of the zoom rectangle and calculate the x-dimension in order to maintain a fixed aspect ratio.

The target aspect ratio is the aspect ratio of the coordinate system(s) at the time the **ChartZoom** object is intitialized.

```
zoomObj.ArCorrectionMode = QCChartTS.ChartConstants.ZOOM_X_AR_CORRECTION
```

Simple zoom example (Adapted from the ZoomExamples.BuildSimpleZoom example)
In this example, a new class derives from the **ChartZoom** class and the internal zoom stack processing is enabled.

[TypeScript]

```
.
.
.
let zoomObj: QCChartTS.ChartZoom = QCChartTS.ChartZoom.newChartZoom(chartVu, pTransform1, true);
zoomObj.setZoomYEnable(true);
zoomObj.setZoomXEnable(true);
zoomObj.setZoomXRoundMode(QCChartTS.ChartConstants.AUTOAXES_FAR);
zoomObj.setZoomYRoundMode(QCChartTS.ChartConstants.AUTOAXES_FAR);
zoomObj.InternalZoomStackProcesssing = true;
zoomObj.setEnable(true);
//  set range limits to 1000 ms, 1 degree
zoomObj.setZoomRangeLimitsRatio( QCChartTS.ChartDimension.newChartDimension(0.001, 0.001));
chartVu.setCurrentMouseListener(zoomObj);
```

[JavaScript]

```
.
.
.
let zoomObj = QCChartTS.ChartZoom.newChartZoom(chartVu, pTransform1, true);
zoomObj.setZoomYEnable(true);
zoomObj.setZoomXEnable(true);
zoomObj.setZoomXRoundMode(QCChartTS.ChartConstants.AUTOAXES_FAR);
zoomObj.setZoomYRoundMode(QCChartTS.ChartConstants.AUTOAXES_FAR);
zoomObj.InternalZoomStackProcesssing = true;
zoomObj.setEnable(true);
//  set range limits to 1000 ms, 1 degree
```

```
zoomObj.setZoomRangeLimitsRatio(QCChartTS.ChartDimension.newChartDimension(0.001, 0.001));
chartVu.setCurrentMouseListener(zoomObj);
```

Super Zooming of multiple physical coordinate systems

The **ChartZoom** class also supports the zooming of multiple physical coordinate systems (*super zooming*). During the mouse released event, the zoom rectangle is converted from device units into the physical coordinates of each scale, and this information is used to re-scale each coordinate system, and the axis objects associated with them.

Use the constructor below in order to super zoom a chart that has multiple coordinate systems and axes.

ChartZoom constructor

```
public static newChartSuperZoomN(component: ChartView, transforms: PhysicalCoordinates[],
        numtransforms: number, brescale: boolean): ChartZoom

public static newChartSuperZoom(component: ChartView, transforms: PhysicalCoordinates[],
        brescale: boolean): ChartZoom ale
);
```

component	A reference to the **ChartView** object that the chart is placed in.
transforms	An array, size numtransforms, of the **PhysicalCoordinates** objects associated with the zoom operation.
brescale	True designates that the all of the scales should be re-scaled, once the final zoom rectangle is ascertained.

Call the **ChartZoom.setEnable(true)** method to enable the zoom object.

Restrict zooming in the x- or y-direction using the setZoomXEnable and setZoomYEnable methods. Set the rounding mode associated with rescale operations using the setZoomXRoundMode and setZoomYRoundMode methods. Call the **ChartZoom.popZoomStack** method at any time and the chart scale reverts to the minimum and maximum values of the previous zoom operation. Repeated calls to the PopZoomStack method return the chart scale is to its original condition, after which the PopZoomStack method has no effect.

Integrated zoom stack processing

```
zoomObj.InternalZoomStackProcesssing = true;
```

Return to a previous zoom level by left clicking the mouse.

Super zoom example (Adapted from the ZoomExamples.BuildSuperZoom example)

[TypeScript]

```
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 Dataset4: QCChartTS.SimpleDataset = QCChartTS.SimpleDataset.newSimpleDataset("Fourth", x1,
y4);
let Dataset5: QCChartTS.SimpleDataset = QCChartTS.SimpleDataset.newSimpleDataset("Fifth", x1,
y5);

let pTransform1: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
let pTransform2: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform2.autoScaleDatasetRoundMode(Dataset2, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
let pTransform3: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform3.autoScaleDatasetRoundMode(Dataset3, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
let pTransform4: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform4.autoScaleDatasetRoundMode(Dataset4, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
let pTransform5: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);

.
.
.
let transformArray: QCChartTS.CartesianCoordinates[] = [
pTransform1,
pTransform2,
pTransform3,
pTransform4,
pTransform5];
let zoomObj: QCChartTS.ChartZoom = QCChartTS.ChartZoom.newChartSuperZoom(chartVu, transformArray,
true);
zoomObj.ArCorrectionMode = QCChartTS.ChartConstants.ZOOM_X_AR_CORRECTION;
zoomObj.setZoomYEnable(true);
zoomObj.setZoomXEnable(true);
zoomObj.setZoomXRoundMode(QCChartTS.ChartConstants.AUTOAXES_EXACT);
zoomObj.setZoomYRoundMode(QCChartTS.ChartConstants.AUTOAXES_EXACT);
zoomObj.setEnable(true);
zoomObj.InternalZoomStackProcesssing = true;
zoomObj.ArCorrectionMode = QCChartTS.ChartConstants.ZOOM_NO_AR_CORRECTION;
chartVu.setCurrentMouseListener(zoomObj);


.
```

```
.
```

[JavaScript]

```
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 Dataset4 = QCChartTS.SimpleDataset.newSimpleDataset("Fourth", x1, y4);
let Dataset5 = QCChartTS.SimpleDataset.newSimpleDataset("Fifth", x1, y5);
let pTransform1 =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
let pTransform2 =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform2.autoScaleDatasetRoundMode(Dataset2, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
let pTransform3 =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform3.autoScaleDatasetRoundMode(Dataset3, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
let pTransform4 =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform4.autoScaleDatasetRoundMode(Dataset4, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
let pTransform5 =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
.
.
.

chartVu.addChartObject(footer);
let transformArray = [
    pTransform1,
    pTransform2,
    pTransform3,
    pTransform4,
    pTransform5
];
let zoomObj = QCChartTS.ChartZoom.newChartSuperZoom(chartVu, transformArray, true);
zoomObj.ArCorrectionMode = QCChartTS.ChartConstants.ZOOM_X_AR_CORRECTION;
zoomObj.setZoomYEnable(true);
zoomObj.setZoomXEnable(true);
zoomObj.setZoomXRoundMode(QCChartTS.ChartConstants.AUTOAXES_EXACT);
zoomObj.setZoomYRoundMode(QCChartTS.ChartConstants.AUTOAXES_EXACT);
zoomObj.setEnable(true);
zoomObj.InternalZoomStackProcesssing = true;
zoomObj.ArCorrectionMode = QCChartTS.ChartConstants.ZOOM_NO_AR_CORRECTION;
chartVu.setCurrentMouseListener(zoomObj);
```

Limiting the Zoom Range

A zoom window needs to have zoom limits placed on the minimum allowable zoom range for the x- and y-coordinates. Unrestricted, or infinte zooming can result in numeric under and overflows. The default minimum allowable range resulting from a zoom operation is 1/1000 of the original coordinate range. Change this value using the **ChartZoom.setZoomRangeLimitsRatio** method. The minimum allowable range for this value is approximately 1.0e-9. Another way to set the minimum allowable range is to specify explicit values for the x- and y-range using the **ChartZoom.setZoomRangeLimits** method.

Specify the minimum allowable zoom range for a time axis in milliseconds, for example **ChartZoom.setZoomRangeLimits(new ChartDimension(1000, 0.01))** sets the minimum zoom range for the time axis to 1 second and for the y-axis to 0.01. The utility method **ChartCalendar.getCalendarWidthValue** is useful for calculating the milliseconds for any time base and any number of units. The code below sets a minimum zoom range of 45 minutes.

[TypeScript]

```
let minZoomTimeRange =
QCChartTS.ChartCalendar.getCalendarWidthValue(QCChartTS.ChartConstants.MINUTE, 45);
let minZoomYRange = 0.01;
let zoomLimits = QCChartTS.ChartDimension.newChartDimension(minZoomTimeRange, minZoomYRange);
zoomObj.setZoomRangeLimits(zoomLimits);
```

[JavaScript]

```
let minZoomTimeRange: number   =
QCChartTS.ChartCalendar.getCalendarWidthValue(QCChartTS.ChartConstants.MINUTE, 45);
let minZoomYRange:  number  = 0.01;
let zoomLimits:  QCChartTS.ChartDimension  =
QCChartTS.ChartDimension.newChartDimension(minZoomTimeRange, minZoomYRange);
zoomObj.setZoomRangeLimits(zoomLimits);
```

the mouse/touch

16. Data Tooltips

DataToolTip

Tooltip is a catchall phrase for a popup window that displays useful information about an object. A data tooltip is a popup box that displays the value of a data point in a chart. The data value can consist of the x-value, the y-value, or both values for a given point in a chart. The tooltip values are displayed using the numeric and time formats supported by the **NumericLabel** and **TimeLabel** classes.

Simple Data Tooltips

Class DataToolTip

```
MouseListener
   |
   +-DataToolTip
```

The **DataToolTip** class implements mouse event delegates. It implements and uses the mouse/touch events: onMouseDown, onDoubleClick, onMouseDown, onMouseUp, and onClick. The default operation of the **DataToolTip** class traps the mouse pressed event. It calculates which chart object intersects the mouse cursor and which data points for the intersecting object are closest. Next, it pops up a window and displays the x-value and/or y-value representing the data point. When the mouse button is released, the onMouseUp event, the tooltip popup window is deleted.

The tooltip data point search algorithm is complicated by the situation that many chart objects occupy a much larger area than the data point that is represented. Bars are a good example. A bar can occupy a large area, yet the actual data value represented by the bar is only a small point at the top. The tooltip algorithm searches for an intersection of the bar and the mouse cursor, not an intersection of the data point and the mouse cursor. If a hit on the bar is detected, the data values represented by the bar are used as the values displayed in the tooltip. The tooltip symbol will highlight the actual data point at the top of the bar. The tooltip window will always popup at the cursor location, not the data point location. If you click anywhere on a bar, the tooltip window will popup at that location and display the data value represented by the top of the bar.

The tooltip data point search algorithm works with both simple and group data. When used with simple plot objects (**SimpleLinePlot**, **SimpleBarPlot**, etc.) it locates the xy data point associated with the mouse event. When used with group plot objects it locates the x-value and the y-group value associated with the mouse event. It is able to differentiate between “stacked” group plot objects (**StackedBarPlot**, **StackedLinePlot**) and the other group plot objects that are not stacked (**GroupBarPlot**, **MultiLinePlot**, **OHLCPlot**, **CandlestickPlot**, etc.). The tooltip values displayed in the tooltip window reflect the actual data values stored in the associated dataset and do not reflect the implicit summation that goes on in the display of stacked plot objects. You should not use symbols to highlight the tooltip data point for stacked objects since the position of the tooltip symbol in the chart will not take into account the stacked object summation.

DataToolTip constructors

The constructors below create a **DataToolTip** object.

```
public static newDataToolTip(component: ChartView): DataToolTip
```

component	A reference to the **ChartView** object that the chart is placed in.

Create the **DataToolTip** object using the newDataToolTip constructor. The DataToolTip is the exception among the MouseListener classes, in that it does **NOT** have to be explicitly added as a MouseListener (using ChartView.setCurrentMouseListener). It is automatically added as a mouse listener, and enabled, when you create it. You can disable it using the setEnable(false) function. and then install it using the **ChartView.setCurrentMouseListener** method. This adds the **DataToolTip** object as a **MouseListener** to the **ChartView** object. Call the **ChartView removeMouseListener(mouselistener)** to remove the object as a mouse listener for the chart view.

Set the threshold distance for deciding if the nearest data point found is a “hit” using the **DataToolTip.setHitTestThreshold** method.

The default values for the **DataToolTip** class assume the following:

- The left mouse button pressed (or tap of the touch screen) event is the trigger for the tooltip.
- The numeric format for the x- and y-values are controlled by separate NumericLabel class templates that are both initially set to the ChartObj.DECIMALFORMAT format with a decimal precision of 1. Change this by creating a NumericLabel or TimeLabel object that specifies how you want the number formatted. Set the x- and y-value templates independently using the DataToolTip.setXValueTemplate and DataToolTip.setYValue template methods.
- The tooltip will display just the y-value of the selected object. Change this to display the x-value or both the x and y-values using the DataToolTip.setDataToolTipFormat method, specifying one of the data tooltip format constants: DATA_TOOLTIP_CUSTOM, DATA_TOOLTIP_X, DATA_TOOLTIP_Y, DATA_TOOLTIP_XY_ONELINE,

DATA_TOOLTIP_TWOLINE, DATA_TOOLTIP_GROUP_MULTILINE, DATA_TOOLTIP_OHLC.

- The tooltip popup window uses a default Sans Serif font with a size of 12. The text is justified above and to the right of the mouse cursor. The default background color of the data tooltip is a pale yellow, RGB (255,255,204). Change this by replacing the ChartText object used as the template for the tooltip popup window. Use the DataToolTip.setTextTemplate method to replace the default template with your own.
- The selected data point is highlighted using a ChartSymbol object, set to a default shape of ChartConstants.SQUARE, a size of 8 and a color of black. Change this by replacing the ChartSymbol used as the template with one of your own.

Another useful mode of the DataToolTip is the passover mode. Rather than waste a mouse click to retrieve a points data value, the DataToolTip can always be looking for the nearest plot point when you just hover the mouse cursor over the chart. If you linger on a data point more than a second, and you are within the hitThreshold, the tooltip will display, event without clicking the mouse button. Enable the passover move using the tooltip PassOverToolTip property as I the example below. The example SimpleLinePlots.BuildLinePlotSales demonstrates its use.

```
datatooltip.PassOverToolTip = true;
```

Simple data tooltip example – almost every example uses DataToolTip

[TypeScript]

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

[JavaScript]

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

Medium complex data tooltip example (Adapted from the SimpleLinePlots.BuildLineFill example)
In this example, the tooltip will display the x-value of the data point as a date, and the y-value as currency. The x- and y-values are displayed on two separate lines, one above the other.

[TypeScript]

```
let toolTipFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Arial",
QCChartTS.ChartFont.REGULAR, 12);
let datatooltip: QCChartTS.DataToolTip = QCChartTS.DataToolTip.newDataToolTip(chartVu);
let xValueTemplate: QCChartTS.TimeLabel =
QCChartTS.TimeLabel.newTimeLabelFormat(QCChartTS.ChartConstants.TIMEDATEFORMAT_MDY);
let yValueTemplate: QCChartTS.NumericLabel =
QCChartTS.NumericLabel.newNumericLabelFormatDecs(QCChartTS.ChartConstants.CURRENCYFORMAT, 0);
datatooltip.getToolTipSymbol().setColor(QCChartTS.ChartColor.GREEN);
datatooltip.setXValueTemplate(xValueTemplate);
datatooltip.setYValueTemplate(yValueTemplate);
datatooltip.setDataToolTipFormat(QCChartTS.ChartConstants.DATA_TOOLTIP_XY_TWOLINE);
```

[JavaScript]

```
let toolTipFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.REGULAR, 12);
let datatooltip = QCChartTS.DataToolTip.newDataToolTip(chartVu);
let xValueTemplate =
QCChartTS.TimeLabel.newTimeLabelFormat(QCChartTS.ChartConstants.TIMEDATEFORMAT_MDY);
let yValueTemplate =
QCChartTS.NumericLabel.newNumericLabelFormatDecs(QCChartTS.ChartConstants.CURRENCYFORMAT, 0);
datatooltip.getToolTipSymbol().setColor(QCChartTS.ChartColor.GREEN);
datatooltip.setXValueTemplate(xValueTemplate);
datatooltip.setYValueTemplate(yValueTemplate);
datatooltip.setDataToolTipFormat(QCChartTS.ChartConstants.DATA_TOOLTIP_XY_TWOLINE);
```

Complex data tooltip example (Adapted from the FinancialExamples.BuildOpeningScreen example)

In this example, the tooltip will display the x-value of the data point as a date, and the y-value as currency. The x- and y-values are displayed on one line, side by side.

[TypeScript]

```
let toolTipFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.REGULAR, 12);
let datatooltip: QCChartTS.DataToolTip = QCChartTS.DataToolTip.newDataToolTip(chartVu);
let xValueTemplate: QCChartTS.TimeLabel =
QCChartTS.TimeLabel.newTimeLabelFormat(QCChartTS.ChartConstants.TIMEDATEFORMAT_MDYHMS);
//  use minimal constructor
let yValueTemplate: QCChartTS.NumericLabel =
QCChartTS.NumericLabel.newNumericLabelFormatDecs(QCChartTS.ChartConstants.CURRENCYFORMAT, 2);
//  use minimal constructor
let textTemplate: QCChartTS.ChartText = QCChartTS.ChartText.newChartTextFontString(toolTipFont,
"");
textTemplate.setTextBgColor(QCChartTS.ChartColor.fromRgb(255, 255, 204));
textTemplate.setTextBgMode(true);
let toolTipSymbol: QCChartTS.ChartSymbol = QCChartTS.ChartSymbol.newChartSymbol(pTransform1,
QCChartTS.ChartConstants.SQUARE,
QCChartTS.ChartAttribute.newChartAttributeColor(QCChartTS.ChartColor.BLACK));
toolTipSymbol.setSymbolSize(5);
datatooltip.setTextTemplate(textTemplate);
datatooltip.setXValueTemplate(xValueTemplate);
datatooltip.setYValueTemplate(yValueTemplate);
datatooltip.setDataToolTipFormat(QCChartTS.ChartConstants.DATA_TOOLTIP_OHLC);
datatooltip.setToolTipSymbol(toolTipSymbol);
```

[JavaScript]

```
let toolTipFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.REGULAR, 12);
```

```
let datatooltip = QCChartTS.DataToolTip.newDataToolTip(chartVu);
let xValueTemplate =
QCChartTS.TimeLabel.newTimeLabelFormat(QCChartTS.ChartConstants.TIMEDATEFORMAT_MDYHMS);
//  use minimal constructor
let yValueTemplate =
QCChartTS.NumericLabel.newNumericLabelFormatDecs(QCChartTS.ChartConstants.CURRENCYFORMAT, 2);
//  use minimal constructor
let textTemplate = QCChartTS.ChartText.newChartTextFontString(toolTipFont, "");
textTemplate.setTextBgColor(QCChartTS.ChartColor.fromRgb(255, 255, 204));
textTemplate.setTextBgMode(true);
let toolTipSymbol = QCChartTS.ChartSymbol.newChartSymbol(pTransform1,
QCChartTS.ChartConstants.SQUARE,
QCChartTS.ChartAttribute.newChartAttributeColor(QCChartTS.ChartColor.BLACK));
toolTipSymbol.setSymbolSize(5);
datatooltip.setTextTemplate(textTemplate);
datatooltip.setXValueTemplate(xValueTemplate);
datatooltip.setYValueTemplate(yValueTemplate);
datatooltip.setDataToolTipFormat(QCChartTS.ChartConstants.DATA_TOOLTIP_OHLC);
datatooltip.setToolTipSymbol(toolTipSymbol);
```

Custom Tooltip displays

It would be impossible to provide options for all possible tooltip displays. The **DataToolTip** class includes an option that enables the programmer to override the existing behavior of the class. The programmer is able to use the built in search routines to identify what plot object is selected, the dataset, the coordinate system and the actual data values associated with the plot object. Using this information the programmer can customize the text displayed in the **ChartText** object used to display the tooltip text.

Use the following steps to create a custom tooltip.

- Subclass the **DataToolTip** class with one of your own.
- Enable the custom mode by calling **DataToolTip.setDataToolTipFormat** method.
- Override the **onMouseDown** and **onMouseUp** events and add the code needed to customize the display. In your custom **onMouseDown** event, make sure you call **super**.**onMouseDown** first, since this selects the plot object for you, and makes the plot objects coordinate system, dataset, and selected data point available using get methods. You can place your tooltip text in the **ChartText** object internal to the **DataToolTip** class. Get a reference to this object using the **DataToolTip.getTextTemplate**() method. In your custom **onMouseUp** event call **super.onMouseUp** followed by a call to the **ChartView** repaint method (chartVu.updateDraw() in the example below);

Custom DataToolTip example (Adapted from the OHLChart example)

In this example, a new class is derived from the **DataToolTip** class and the onMouseDown and onMouseUp events are overridden.

[TypeScript]

```
class CustomOHLCToolTip extends QCChartTS.DataToolTip {

  stockpanel: QCChartTS.ChartText;

  OHLCObj: OHLCChart| null = null;

  public constructor()
  {
      super();
      this.stockpanel = this.getTextTemplate();
  }


  public static newCustomOHLCToolTip(component: OHLCChart): CustomOHLCToolTip
  {
      let result: CustomOHLCToolTip = new CustomOHLCToolTip();
      result.OHLCObj = component;
      return result;
  }

  public onMouseUp(mouseevent: MouseEvent) {
class CustomOHLCToolTip2 extends QCChartTS.DataToolTip {

    stockpanel: QCChartTS.ChartText;

    OHLCObj: OHLCChart | null = null;

    public constructor() {
        super();
        this.stockpanel = this.getTextTemplate();
    }


    public static newCustomOHLCToolTip(component: OHLCChart): CustomOHLCToolTip {
        let result: CustomOHLCToolTip = new CustomOHLCToolTip();
        result.OHLCObj = component;
        return result;
    }

    public onMouseUp(mouseevent: MouseEvent) {
        super.onMouseUp(mouseevent);
        //  Redraws the chart. Since the stockpanel object has not been added to the chart
        //  (using addChartObject) it will not be redrawn when the chart is redrawn
        if (this.ChartObjComponent)
            this.ChartObjComponent.updateDraw();
    }

    public onMouseDown(mouseevent: MouseEvent) {
        let mousepos: QCChartTS.ChartPoint2D = this.getAdjMouseCoords(mouseevent);

        super.onMouseDown(mouseevent);
        let selectedPlot: QCChartTS.ChartPlot | null = this.getSelectedPlotObj();
        if ((selectedPlot)) {
            let selectedindex: number = this.getNearestPoint().getNearestPointIndex();
            let transform: QCChartTS.PhysicalCoordinates | null =
this.getSelectedCoordinateSystem();
            if (!transform) return;
            this.stockpanel.setChartObjScale(transform);
            this.stockpanel.setLocationPointType(mousepos, QCChartTS.ChartConstants.DEV_POS);
            this.stockpanel.setTextString("Stock Data");
            //  Looking to the original arrays, because we just have the selectedindex,
            //  yet we want to display stock O-H-L-C data, volume and NASDAQ. Only one
            //  of these datasets can be selected at a time by the tooltip.
            if (!this.OHLCObj) return;

            let open: number = this.OHLCObj.stockPriceData[0][selectedindex];
            let high: number = this.OHLCObj.stockPriceData[1][selectedindex];
            let low: number = this.OHLCObj.stockPriceData[2][selectedindex];
            let close: number = this.OHLCObj.stockPriceData[3][selectedindex];
            let nasdaq: number = this.OHLCObj.NASDAQData[selectedindex];
            let volume: number = this.OHLCObj.stockVolumeData[selectedindex];
            let openObj: string = QCChartTS.ChartSupport.numToString(open,
QCChartTS.ChartConstants.DECIMALFORMAT, 2, "");
            let highObj: string = QCChartTS.ChartSupport.numToString(high,
QCChartTS.ChartConstants.DECIMALFORMAT, 2, "");
            let lowObj: string = QCChartTS.ChartSupport.numToString(low,
QCChartTS.ChartConstants.DECIMALFORMAT, 2, "");
            let closeObj: string = QCChartTS.ChartSupport.numToString(close,
QCChartTS.ChartConstants.DECIMALFORMAT, 2, "");
            let volumeObj: string = QCChartTS.ChartSupport.numToString(volume,
QCChartTS.ChartConstants.DECIMALFORMAT, 0, "");
            let nasdaqObj: string = QCChartTS.ChartSupport.numToString(nasdaq,
QCChartTS.ChartConstants.DECIMALFORMAT, 2, "");
            let timelabel: QCChartTS.TimeLabel =
QCChartTS.TimeLabel.newTimeLabelCoordsFormat(transform, this.OHLCObj.xValues[selectedindex],
QCChartTS.ChartConstants.TIMEDATEFORMAT_STANDARD);
            this.stockpanel.addNewLineTextString(timelabel.getTextString());
            this.stockpanel.addNewLineTextString(("Open " + openObj));
            this.stockpanel.addNewLineTextString(("High " + highObj));
            this.stockpanel.addNewLineTextString(("Low " + lowObj));
            this.stockpanel.addNewLineTextString(("Close " + closeObj));
            this.stockpanel.addNewLineTextString(("Volume " + volumeObj));
            this.stockpanel.addNewLineTextString(("NASDAQ " + nasdaqObj));
            this.stockpanel.setChartObjEnable(QCChartTS.ChartConstants.OBJECT_ENABLE);
            if (!this.ChartObjComponent) return;
            let g2: QCChartTS.QCGraphics | null = this.ChartObjComponent.theGraphics;
            if (!g2) return;
            //  Precalculates the text bounding box so that the size is
            //  known before it is drawn
            this.stockpanel.preCalcTextBoundingBox(g2);
            let boundingbox: QCChartTS.ChartRectangle2D = this.stockpanel.getTextBox();
            //  Reposition tooltip text box if top of box near top of graph window
            //  You can do the same thing for all four sides of the graph window
            if (((mousepos.getY() - boundingbox.getHeight())
                < 1)) {
                mousepos.setLocation(mousepos.getX(), (mousepos.getY() +
boundingbox.getHeight()));
                this.stockpanel.setLocationPointType(mousepos, QCChartTS.ChartConstants.DEV_POS);
            }

            //  Draws the tooltip text panel to the chart graphics context
            this.stockpanel.draw(g2);
        }
    }
}
.
.
.



let stocktooltip: CustomOHLCToolTip = CustomOHLCToolTip.newCustomOHLCToolTip(this);
stocktooltip.XValueTemplate = new QCChartTS.TimeLabel();
stocktooltip.setDataToolTipFormat(QCChartTS.ChartConstants.DATA_TOOLTIP_CUSTOM);
```

[JavaScript]

```
class CustomOHLCToolTip extends QCChartTS.DataToolTip {

    constructor() {
        super();
        this.stockpanel = this.getTextTemplate();
    }
    static newCustomOHLCToolTip() {
        let result = new CustomOHLCToolTip();
        return result;
    }



    onMouseUp(mouseevent) {
        super.onMouseUp(mouseevent);
        //  Redraws the chart. Since the stockpanel object has not been added to the chart
        //  (using addChartObject) it will not be redrawn when the chart is redrawn
        if (this.ChartObjComponent)
            this.ChartObjComponent.updateDraw();
    }

    onMouseDown(mouseevent) {
        let mousepos = this.getAdjMouseCoords(mouseevent);

        super.onMouseDown(mouseevent);
        let selectedPlot = this.getSelectedPlotObj();
        if ((selectedPlot)) {
            let selectedindex = this.getNearestPoint().getNearestPointIndex();
            let transform = this.getSelectedCoordinateSystem();
            if (!transform) return;
            this.stockpanel.setChartObjScale(transform);
            this.stockpanel.setLocationPointType(mousepos, QCChartTS.ChartConstants.DEV_POS);
            this.stockpanel.setTextString("Stock Data");
            //  Looking to the original arrays, because we just have the selectedindex,
            //  yet we want to display stock O-H-L-C data, volume and NASDAQ. Only one
            //  of these datasets can be selected at a time by the tooltip.

            let open =stockPriceData[0][selectedindex];
            let high =stockPriceData[1][selectedindex];
            let low =stockPriceData[2][selectedindex];
            let close =stockPriceData[3][selectedindex];
            let nasdaq =NASDAQData[selectedindex];
            let volume =stockVolumeData[selectedindex];
            let openObj = QCChartTS.ChartSupport.numToString(open,
QCChartTS.ChartConstants.DECIMALFORMAT, 2, "");
            let highObj = QCChartTS.ChartSupport.numToString(high,
QCChartTS.ChartConstants.DECIMALFORMAT, 2, "");
            let lowObj = QCChartTS.ChartSupport.numToString(low,
QCChartTS.ChartConstants.DECIMALFORMAT, 2, "");
            let closeObj = QCChartTS.ChartSupport.numToString(close,
QCChartTS.ChartConstants.DECIMALFORMAT, 2, "");
            let volumeObj = QCChartTS.ChartSupport.numToString(volume,
QCChartTS.ChartConstants.DECIMALFORMAT, 0, "");
            let nasdaqObj = QCChartTS.ChartSupport.numToString(nasdaq,
QCChartTS.ChartConstants.DECIMALFORMAT, 2, "");
            let timelabel =
QCChartTS.TimeLabel.newTimeLabelCoordsFormat(transform,xValues[selectedindex],
QCChartTS.ChartConstants.TIMEDATEFORMAT_STANDARD);
            this.stockpanel.addNewLineTextString(timelabel.getTextString());
            this.stockpanel.addNewLineTextString(("Open " + openObj));
            this.stockpanel.addNewLineTextString(("High " + highObj));
            this.stockpanel.addNewLineTextString(("Low " + lowObj));
            this.stockpanel.addNewLineTextString(("Close " + closeObj));
            this.stockpanel.addNewLineTextString(("Volume " + volumeObj));
            this.stockpanel.addNewLineTextString(("NASDAQ " + nasdaqObj));
            this.stockpanel.setChartObjEnable(QCChartTS.ChartConstants.OBJECT_ENABLE);
            if (!this.ChartObjComponent) return;
            let g2 = this.ChartObjComponent.theGraphics;
            if (!g2) return;
            //  Precalculates the text bounding box so that the size is
            //  known before it is drawn
            this.stockpanel.preCalcTextBoundingBox(g2);
            let boundingbox = this.stockpanel.getTextBox();
            //  Reposition tooltip text box if top of box near top of graph window
            //  You can do the same thing for all four sides of the graph window
            if (((mousepos.getY() - boundingbox.getHeight())
                < 1)) {
                mousepos.setLocation(mousepos.getX(), (mousepos.getY() +
boundingbox.getHeight()));
                this.stockpanel.setLocationPointType(mousepos, QCChartTS.ChartConstants.DEV_POS);
            }

            //  Draws the tooltip text panel to the chart graphics context
            this.stockpanel.draw(g2);
        }
    }
}
.
.
.


let stocktooltip = CustomOHLCToolTip.newCustomOHLCToolTip();
stocktooltip.XValueTemplate = new QCChartTS.TimeLabel();
stocktooltip.setDataToolTipFormat(QCChartTS.ChartConstants.DATA_TOOLTIP_CUSTOM);

.
.
```

17. Pie and Ring Charts

PieChart
RingChart

Everyone is familiar with the ubiquitous pie chart. Pie charts are 1-dimensional, not because they are shallow, but because they represent a simple 1-dimensional series of numbers, {3, 5, 2, 7, 3, ...], rather than the parametric set of data points { (3,2), (6,3), (7,3)…} used in the other plot types described in this software. The best use of pie charts involves data that has 10 or fewer elements. Otherwise, the text used to label the pie charts starts to overlap in adjacent, small pie wedges. The x-values of a dataset represent the datavalues for each pie wedge. The y-values of the dataset explode a pie wedge from its normal centered position.

A ring chart is a variant of the pie chart. Instead an entire circle (or pie) being the basis of the chart, a ring is used instead.

Using the Pie Chart Class

Class PieChart

```
GraphObj
    |
    +--ChartPlot
         |
         +--SimplePlot
              |
              +-- PieChart
                   |
                   +-- RingChart
```

The **PieChart** and **RingChart** classes extends the **ChartPlot** class and displays pie/ring charts. The x-values of the simple dataset used for data storage specify the pie/ring wedge values. The y-values of the dataset specify the "explode" percentage for each pie/ring wedge.

PieChart Constructor

```
public static newPieChart(transform: PhysicalCoordinates, dataset: SimpleDataset,
        spiestring1s: string[], attribs: ChartAttribute[], labelinout1: number, pielabelformat:
number): PieChart
```

RingChart Constructor

```
public static newRingChart(transform: PhysicalCoordinates, dataset: SimpleDataset,
        spiestring1s: string[], attribs: ChartAttribute[], labelinout1: number, pielabelformat:
number): RingChart
```

transform	The pie/ring chart is placed in the coordinate system defined by transform.
dataset	The pie/ring chart represents the values in this dataset. The x-values of the simple dataset used for data storage specify the pie/ring wedge values. The y-values of the dataset specify the "explode" percentage for each pie/ring wedge.
spiestrings	An array of strings, size dataset.getNumberDatapoints(), used as labels for the pie/ ring slices.
attribs	An array of **ChartAttribute** objects, size dataset.getNumberDatapoints() that specify the attributes (outline color and fill color) for each wedge of a pie/ring chart.
labelinout	An array of integer, size dataset.getNumberDatapoints(), specifying if a specific pie/ring slice text label is drawn inside the pie/ring slice, or outside of the pie/ring slice. Use one of the constants: PIELABEL_OUTSLICE or PIELABEL_INSLICE.
pielabelformat	All pie/ring slice labels share the same format. Use one of the pie/ring slice label format constants:

PIELABEL_NONE Do not display and pie/ring slice text

PIELABEL_STRING Display only the pie/ring text strings, no numeric values

PIELABEL_NUMVALUE Display the pie/ring numeric value only, no pie text strings.

PIELABEL_STRINGNUMVAL Display the pie/ring text string and numeric value.

A pie/ring chart uses a default **CartesianCoordinates** object. Center it in the window using the **CartesianCoordinates.setGraphBorderDiagonal** method. Format the text used to label the pie/ring chart, both the strings and the numeric values, using a **NumericLabel** template set using the **PieChart.setPlotLabelTemplate** method. Change the starting position of the first pie/ring wedge from the default value of 0.0 (3:00 position) using the **PieChart.setStartPieSliceAngle** method. The **PieChart.calcNearestPoint** method can find the pie/ring wedge nearest a specified point, usually the result of a mouse click. See the **LabeledPieChart** example program.

Simple pie chart (extracted from the example program PieCharts.BuildPieAndLineChart). A simlar example for a RingChart is found in the example program PieCharts.BuildRingChart.

[TypeScript]

```
let sPieStrings: string[] = [
  "Tech",
  "Retail",
  "Bank",
  "Auto",
  "Energy",
  "Aero"];
let attribs: QCChartTS.ChartAttribute[] = new Array<QCChartTS.ChartAttribute>(6);
let theFont: QCChartTS.ChartFont;
let colorArray: number[] = [
  QCChartTS.ChartColor.RED,
  QCChartTS.ChartColor.BLUE,
  QCChartTS.ChartColor.CYAN,
  QCChartTS.ChartColor.YELLOW,
  QCChartTS.ChartColor.GREEN,
  QCChartTS.ChartColor.ORANGE,
  QCChartTS.ChartColor.LIGHTGRAY,
  QCChartTS.ChartColor.MAGENTA,
  QCChartTS.ChartColor.DARKGRAY,
  QCChartTS.ChartColor.PINK];
theFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif", QCChartTS.ChartFont.BOLD,
12);
let numPoints: number = 6;
let x1: number[] = new Array(numPoints);
let y1: number[] = new Array(numPoints);
let i: number;
for (i = 0; (i < numPoints); i++) {
  attribs[i] = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, colorArray[i]);
}

x1[0] = 5.8;
y1[0] = 0;
x1[1] = 2.2;
y1[1] = 0;
x1[2] = 3.5;
y1[2] = 0;
x1[3] = 4.2;
y1[3] = 0;
x1[4] = 0.2;
y1[4] = 0;
x1[5] = 3.7;
y1[5] = 0;
let Dataset1: QCChartTS.SimpleDataset = QCChartTS.SimpleDataset.newSimpleDataset("First", x1,
y1);
let pTransform1: QCChartTS.CartesianCoordinates = new QCChartTS.CartesianCoordinates();
pTransform1.setGraphBorderDiagonal(0.01, .2, .3, 0.75);
let background1: QCChartTS.Background =
QCChartTS.Background.newBackgroundCoordsTypeColorGradient(pTransform1,
QCChartTS.ChartConstants.GRAPH_BACKGROUND, QCChartTS.ChartColor.LIGHTGRAY,
QCChartTS.ChartColor.WHITE, QCChartTS.ChartConstants.Y_AXIS);
chartVu.addChartObject(background1);
let thePlot1: QCChartTS.PieChart = QCChartTS.PieChart.newPieChart(pTransform1, Dataset1,
sPieStrings, attribs, QCChartTS.ChartConstants.PIELABEL_OUTSLICE,
QCChartTS.ChartConstants.PIELABEL_STRINGNUMVAL);
thePlot1.setStartPieSliceAngle(-45);
let labeltemplate: QCChartTS.NumericLabel = new QCChartTS.NumericLabel();
labeltemplate.setNumericFormat(QCChartTS.ChartConstants.CURRENCYFORMAT);
labeltemplate.setDecimalPos(1);
labeltemplate.setTextFont(theFont);
thePlot1.setPlotLabelTemplate(labeltemplate);
thePlot1.setShowDatapointValue(false);
thePlot1.setLabelInOut(0, QCChartTS.ChartConstants.PIELABEL_INSLICE);
thePlot1.setLabelInOut(1, QCChartTS.ChartConstants.PIELABEL_INSLICE);
thePlot1.setLabelInOut(2, QCChartTS.ChartConstants.PIELABEL_INSLICE);
thePlot1.setLabelInOut(3, QCChartTS.ChartConstants.PIELABEL_INSLICE);
thePlot1.setLabelInOut(4, QCChartTS.ChartConstants.PIELABEL_OUTSLICE);
```

```
thePlot1.setLabelInOut(5, QCChartTS.ChartConstants.PIELABEL_INSLICE);
chartVu.addChartObject(thePlot1);
```

[JavaScript]

```
let sPieStrings = [
    "Tech",
    "Retail",
    "Bank",
    "Auto",
    "Energy",
    "Aero"
];
let attribs = new Array(6);
let theFont;
let colorArray = [
    QCChartTS.ChartColor.RED,
    QCChartTS.ChartColor.BLUE,
    QCChartTS.ChartColor.CYAN,
    QCChartTS.ChartColor.YELLOW,
    QCChartTS.ChartColor.GREEN,
    QCChartTS.ChartColor.ORANGE,
    QCChartTS.ChartColor.LIGHTGRAY,
    QCChartTS.ChartColor.MAGENTA,
    QCChartTS.ChartColor.DARKGRAY,
    QCChartTS.ChartColor.PINK
];
theFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif", QCChartTS.ChartFont.BOLD,
12);
let numPoints = 6;
let x1 = new Array(numPoints);
let y1 = new Array(numPoints);
let i;
for (i = 0; (i < numPoints); i++) {
    attribs[i] = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, colorArray[i]);
}
x1[0] = 5.8;
y1[0] = 0;
x1[1] = 2.2;
y1[1] = 0;
x1[2] = 3.5;
y1[2] = 0;
x1[3] = 4.2;
y1[3] = 0;
x1[4] = 0.2;
y1[4] = 0;
x1[5] = 3.7;
y1[5] = 0;
let Dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("First", x1, y1);
let pTransform1 = new QCChartTS.CartesianCoordinates();
pTransform1.setGraphBorderDiagonal(0.01, .2, .3, 0.75);
let background1 = QCChartTS.Background.newBackgroundCoordsTypeColorGradient(pTransform1,
QCChartTS.ChartConstants.GRAPH_BACKGROUND, QCChartTS.ChartColor.LIGHTGRAY,
QCChartTS.ChartColor.WHITE, QCChartTS.ChartConstants.Y_AXIS);
chartVu.addChartObject(background1);
let thePlot1 = QCChartTS.PieChart.newPieChart(pTransform1, Dataset1, sPieStrings, attribs,
QCChartTS.ChartConstants.PIELABEL_OUTSLICE, QCChartTS.ChartConstants.PIELABEL_STRINGNUMVAL);
thePlot1.setStartPieSliceAngle(-45);
let labeltemplate = new QCChartTS.NumericLabel();
labeltemplate.setNumericFormat(QCChartTS.ChartConstants.CURRENCYFORMAT);
labeltemplate.setDecimalPos(1);
labeltemplate.setTextFont(theFont);
```

```
thePlot1.setPlotLabelTemplate(labeltemplate);
thePlot1.setShowDatapointValue(false);
thePlot1.setLabelInOut(0, QCChartTS.ChartConstants.PIELABEL_INSLICE);
thePlot1.setLabelInOut(1, QCChartTS.ChartConstants.PIELABEL_INSLICE);
thePlot1.setLabelInOut(2, QCChartTS.ChartConstants.PIELABEL_INSLICE);
thePlot1.setLabelInOut(3, QCChartTS.ChartConstants.PIELABEL_INSLICE);
thePlot1.setLabelInOut(4, QCChartTS.ChartConstants.PIELABEL_OUTSLICE);
thePlot1.setLabelInOut(5, QCChartTS.ChartConstants.PIELABEL_INSLICE);
chartVu.addChartObject(thePlot1);
```

18. Polar and Antenna Plots

PolarPlot
- **PolarLinePlot**
- **PolarScatterPlot**

AntennaPlot
- **AntennaLinePlot**
- **AntennaScatterPlot**
- **AntennaLineMarkerPlot**

Polar charts play an important in engineering applications involving electronics and advanced control systems. Polar charts give a visual interpretation to mathematical problems involving trigonometric functions and complex numbers. Antenna charts are used to display the operating characteristics of antennas.

The **PolarPlot** class is an abstract class representing plot types that use data organized as arrays of x- and y-values, where an x-value represents the magnitude of a point in polar coordinates, and the y-value represents the angle of a point in polar coordinates. Polar plots types include: line plots and scatter plots.

The polar angle values stored as the y-values in the **SimpleDataset** should be in radians. If the raw data is in degrees, convert the data to radians using the **ChartSupport**.**toRadians** method.

The **AntennaPlot** class is an abstract class representing plot types that use data organized as arrays of x- and y-values, where an x-value represents the radial value of a point in antenna coordinates, and the y-value represents the angle of a point in antenna coordinates. Antenna plots types include: line plots, scatter plots, line marker plots, and annotations.

The antenna angle values stored as the y-values in the **SimpleDataset** should be specified in degrees. If the raw data is in radians, convert the data to degrees using the **ChartSupport.toDegrees** method.

If you plan to create polar charts, you need to also familiarize yourself with the polar charting classes describe in the other chapters of this manual. These are the chapters on coordinate systems (**PolarCoordinates** in Chapter 4), axes (**PolarAxes** in Chapter 7), axis labels (**PolarAxesLabels** in Chapter 8) and grids (**PolarGrids** in Chapter 9). The same is true if you plan to create antenna charts. Refer to the documentation in the chapters on coordinate systems (**AntennaCoordinates** in Chapter 4) , axes (**AntennaAxes** in Chapter 7), axis labels (**AntennaAxesLabels** in Chapter 8) and grids (**AntennaGrids** in Chapter 9).

The **ChartZoom** and **MoveCoordinates** classes require a rectangular coordinate system and will not work with polar and antenna charts. The **MoveData** class does work with polar and antenna charts.

Polar Plots

Class PolarLinePlot

```
GraphObj
   |
   +--ChartPlot
         |
         +--PolarPlot
               |
               +-- PolarLinePlot
```

The **PolarLinePlot** class is a concrete implementation of the **PolarPlot** class and displays data in a simple line plot format. The lines drawn between adjacent data points use polar coordinate interpolation.

PolarLinePlot constructor

```
public static newPolarLinePlotCoordinates(transform: PolarCoordinates): PolarLinePlot

public static newPolarLinePlot(transform: PolarCoordinates, dataset: SimpleDataset,
        attrib: ChartAttribute): PolarLinePlot
```

transform	The coordinate system for the new **PolarLinePlot** object.
dataset	The polar line plot represents the polar coordinate values in this dataset. The x-values of the dataset represent the magnitudes of the points and the y-values the polar angles in radians.
attrib	Specifies the attributes (line color and line style) for the line plot.

The polar line plot class interpolates between adjacent data points in polar coordinates and not using straight lines as in the Cartesian coordinate plotting functions. This gives the lines between adjacent data points in a polar plot a curved look.

Polar line plot and scatter plot chart (extracted from the example program PolarCharts.BuildPolarLineChart)

[TypeScript]

```
let nump1: number = 100;
let mag1: number[] = new Array(nump1);
let ang1: number[] = new Array(nump1);

let i: number;
```

```
for (i = 0; i < nump1; i++) {
    ang1[i] = QCChartTS.ChartSupport.toRadians(i * (360.0 / nump1));
    mag1[i] = 10 * Math.abs(30 * (Math.sin(2 * (ang1[i])) * Math.cos(2 * (ang1[i]))));
}

theFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif", QCChartTS.ChartFont.REGULAR,
10);

let Dataset1: QCChartTS.SimpleDataset = QCChartTS.SimpleDataset.newSimpleDataset("First", mag1,
ang1);
let pPolarTransform: QCChartTS.PolarCoordinates =
QCChartTS.PolarCoordinates.newPolarCoordinatesDataset(Dataset1);

pPolarTransform.setGraphBorderDiagonal(0.25, .2, .75, 0.8);

let background: QCChartTS.Background = QCChartTS.Background.newBackground(pPolarTransform,
QCChartTS.ChartConstants.GRAPH_BACKGROUND,
    QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(background);

pPolarTransform.autoScaleDataset(Dataset1);
let pPolarAxis: QCChartTS.PolarAxes = <QCChartTS.PolarAxes>pPolarTransform.getCompatibleAxes();
chartVu.addChartObject(pPolarAxis);

let pPolarGrid: QCChartTS.PolarGrid = QCChartTS.PolarGrid.newPolarGrid(pPolarAxis,
QCChartTS.ChartConstants.GRID_MAJOR);
chartVu.addChartObject(pPolarGrid);

let pPolarAxisLabels: QCChartTS.PolarAxesLabels =
<QCChartTS.PolarAxesLabels>pPolarAxis.getCompatibleAxesLabels();
chartVu.addChartObject(pPolarAxisLabels);

let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 2, 0);
let thePlot1: QCChartTS.PolarLinePlot = QCChartTS.PolarLinePlot.newPolarLinePlot(pPolarTransform,
Dataset1, attrib1);
chartVu.addChartObject(thePlot1);


let attrib2: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute(QCChartTS.ChartColor.RED, 1, 0,
QCChartTS.ChartColor.RED);
attrib2.setFillFlag(true);
let thePlot2: QCChartTS.PolarScatterPlot =
QCChartTS.PolarScatterPlot.newPolarScatterPlot(pPolarTransform, Dataset1,
QCChartTS.ChartConstants.CIRCLE, attrib2);
chartVu.addChartObject(thePlot2);
```

[JavaScript]

```
let nump1 = 100;
let mag1 = new Array(nump1);
let ang1 = new Array(nump1);

let i;
for (i = 0; i < nump1; i++) {
    ang1[i] = QCChartTS.ChartSupport.toRadians(i * (360.0 / nump1));
    mag1[i] = 10 * Math.abs(30 * (Math.sin(2 * (ang1[i])) * Math.cos(2 * (ang1[i]))));
}

theFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif", QCChartTS.ChartFont.REGULAR,
10);

let Dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("First", mag1, ang1);
let pPolarTransform = new QCChartTS.PolarCoordinates();

pPolarTransform.setGraphBorderDiagonal(0.25, .2, .75, 0.8);
```

```
let background = QCChartTS.Background.newBackground(pPolarTransform,
QCChartTS.ChartConstants.GRAPH_BACKGROUND,
    QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(background);

pPolarTransform.autoScaleDataset(Dataset1);
let pPolarAxis = pPolarTransform.getCompatibleAxes();
chartVu.addChartObject(pPolarAxis);

let pPolarGrid = QCChartTS.PolarGrid.newPolarGrid(pPolarAxis,
QCChartTS.ChartConstants.GRID_MAJOR);
chartVu.addChartObject(pPolarGrid);

let pPolarAxisLabels = pPolarAxis.getCompatibleAxesLabels();
chartVu.addChartObject(pPolarAxisLabels);

let attrib1 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 2, 0);
let thePlot1 = QCChartTS.PolarLinePlot.newPolarLinePlot(pPolarTransform, Dataset1, attrib1);
chartVu.addChartObject(thePlot1);


let attrib2 = QCChartTS.ChartAttribute.newChartAttribute(QCChartTS.ChartColor.RED, 1, 0,
QCChartTS.ChartColor.RED);
attrib2.setFillFlag(true);
let thePlot2 = QCChartTS.PolarScatterPlot.newPolarScatterPlot(pPolarTransform, Dataset1,
QCChartTS.ChartConstants.CIRCLE, attrib2);
chartVu.addChartObject(thePlot2);
```

Class PolarScatterPlot

```
GraphObj
   |
   +--ChartPlot
        |
        +--PolarPlot
             |
             +-- PolarScatterPlot
```

The **PolarScatterPlot** class is a concrete implementation of the **PolarPlot** class and displays data in a simple scatter plot format. This means that lines drawn between points in an antenna chart will look curved, which is how they should look.

PolarScatterPlot constructor

```
public static newPolarScatterPlotCoords(transform: PolarCoordinates): PolarScatterPlot

public static newPolarScatterPlot(transform: PolarCoordinates,
        dataset: SimpleDataset, symtype: number,
        attrib: ChartAttribute): PolarScatterPlot
```

transform	The coordinate system for the new **PolarScatterPlot** object.

dataset	The polar scatter plot represents the polar coordinate values in this dataset. The x-values of the dataset represent the magnitudes of the points and the y-values the polar angles in radians.
symtype	The symbol used in the scatter plot. Use one of the scatter plot symbol constants: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE.
attrib	Specifies the attributes (size, line and fill color) for the scatter plot.

See previous example for a programming example using PolarScatterPlot.

Antenna Plots

Class AntennaLinePlot

```
GraphObj
   |
   +--ChartPlot
         |
         +--AntennaPlot
               |
               +-- AntennaLinePlot
```

The **AntennaLinePlot** class is a concrete implementation of the **AntennaPlot** class and displays data in a simple line plot format. The lines drawn between adjacent data points use antenna coordinate interpolation.

AntennaLinePlot constructor

```
public static newAntennaLinePlotCoords(transform: AntennaCoordinates): AntennaLinePlot
public static newAntennaLinePlot(transform: AntennaCoordinates, dataset: SimpleDataset,
        attrib: ChartAttribute): AntennaLinePlot
```

transform	The coordinate system for the new **AntennaLinePlot** object.
dataset	The antenna line plot represents the antenna coordinate values in this dataset. The x-values of the dataset represent the radial values and the y-values represent the antenna angular values in degrees.

attrib	Specifies the attributes (line color and line style) for the line plot.

The antenna line plot class interpolates between adjacent data points in antenna coordinates and not using straight lines as in the Cartesian coordinate plotting functions. This gives the lines between adjacent data points in a antenna plot a curved look.

Antenna line plot and scatter plot chart (extracted from the example program AntennaCharts.BuildAntennaLineChart)

[TypeScript]

```
let Dataset1: QCChartTS.SimpleDataset;
let Dataset2: QCChartTS.SimpleDataset;

let theFont: QCChartTS.ChartFont;
let num: number = 61;

let mag1: number[] = new Array(num);
let mag2: number[] = new Array(num);
let ang1: number[] = new Array(num);
let i: number;
for (i = 0; i < num - 1; i++) {
.
.
.

}
// close loop
mag1[num - 1] = mag1[0];
mag2[num - 1] = mag2[0];
ang1[num - 1] = ang1[0];



theFont = QCChartTS.ChartFont.newChartFont("Microsoft Sans Serif", QCChartTS.ChartFont.BOLD, 10);
Dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("First", mag1, ang1);
Dataset2 = QCChartTS.SimpleDataset.newSimpleDataset("Second", mag2, ang1);

let datasetarray: QCChartTS.SimpleDataset[] = [Dataset1, Dataset2];
this.pAntennaTransform = new QCChartTS.AntennaCoordinates();
this.pAntennaTransform.autoScaleDatasetsRoundMode(datasetarray,
QCChartTS.ChartConstants.AUTOAXES_FAR);

this.pAntennaTransform.setGraphBorderDiagonal(0.25, .15, .75, 0.85);

let background: QCChartTS.Background = QCChartTS.Background.newBackground(this.pAntennaTransform,
QCChartTS.ChartConstants.GRAPH_BACKGROUND,
  QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(background);

let pAntennaAxis: QCChartTS.AntennaAxes = this.pAntennaTransform.getCompatibleAxes();
pAntennaAxis.setLineColor(QCChartTS.ChartColor.BLACK);

let axestickspace: number = 1;
let axesntickspermajor: number = 5;
let angletickspace: number = 5;
let anglentickspermajor: number = 6;
let minorticlength: number = 5;
let majorticlength: number = 10;
let tickdir: number = QCChartTS.ChartConstants.AXIS_CENTER;

pAntennaAxis.setAntennaAxesTicksFull(axestickspace, axesntickspermajor,
  angletickspace, anglentickspermajor,
  minorticlength, majorticlength,
  tickdir);

chartVu.addChartObject(pAntennaAxis);

let pAntennaGrid: QCChartTS.AntennaGrid = QCChartTS.AntennaGrid.newAntennaGrid(pAntennaAxis,
QCChartTS.ChartConstants.GRID_ALL);
pAntennaGrid.setChartObjAttributes(QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartCol
or.LIGHTBLUE, 1, QCChartTS.ChartConstants.LS_SOLID));
chartVu.addChartObject(pAntennaGrid);

let axisLabelsFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont("Microsoft Sans
Serif", QCChartTS.ChartFont.REGULAR, 24);

let pAntennaAxisLabels: QCChartTS.AntennaAxesLabels =
<QCChartTS.AntennaAxesLabels>pAntennaAxis.getCompatibleAxesLabels();
//  pAntennaAxisLabels.setTextFont(axisLabelsFont);
chartVu.addChartObject(pAntennaAxisLabels);

let transparentRed: number = QCChartTS.ChartSupport.fromArgb(180, 255, 0, 0);
let transparentBlue: number = QCChartTS.ChartSupport.fromArgb(180, 0, 0, 255);


let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(transparentRed, 1,
QCChartTS.ChartConstants.LS_SOLID);
attrib1.setSymbolSize(7);
let attrib2: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.BLUE);
attrib2.setSymbolSize(7);

let attrib3: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute(QCChartTS.ChartColor.YELLOW, 3,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.YELLOW);
let attrib4: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.MEDIUMPURPLE, 2,
QCChartTS.ChartConstants.LS_DOT_1_4);


let thePlot1: QCChartTS.AntennaLinePlot  =
    QCChartTS.AntennaLinePlot.newAntennaLinePlotCoordinates(this.pAntennaTransform);
    thePlot1.initAntennaLinePlot(Dataset1, attrib1);
    chartVu.addChartObject(thePlot1);

let thePlot2: QCChartTS.AntennaScatterPlot  =
    QCChartTS.AntennaScatterPlot.newAntennaScatterPlotCoordinates(this.pAntennaTransform);
    thePlot2.initAntennaScatterPlot(Dataset2, QCChartTS.ChartConstants.SQUARE, attrib2);
    chartVu.addChartObject(thePlot2);



let thePlot3: QCChartTS.AntennaAnnotation =
QCChartTS.AntennaAnnotation.newAntennaAnnotation(this.pAntennaTransform,
QCChartTS.ChartConstants.ANTENNA_ANNOTATION_ANGULAR, 180, attrib3);
chartVu.addChartObject(thePlot3);


let thePlot4: QCChartTS.AntennaAnnotation =
QCChartTS.AntennaAnnotation.newAntennaAnnotation(this.pAntennaTransform,
QCChartTS.ChartConstants.ANTENNA_ANNOTATION_RADIUS, 12, attrib4);
chartVu.addChartObject(thePlot4);
```

[JavaScript]

```
 var Dataset1;
  var Dataset2;
  var theFont;
  var num = 61;

  var mag1 = new Array(num);
  var mag2 = new Array(num);
  var ang1 = new Array(num);
  var i;
  for (i = 0; i < num - 1; i++) {
  .
  .
  .

  }
  // close loop
  mag1[num - 1] = mag1[0];
  mag2[num - 1] = mag2[0];
  ang1[num - 1] = ang1[0];



  theFont = QCChartTS.ChartFont.newChartFont("Microsoft Sans Serif", QCChartTS.ChartFont.BOLD,
10);
  Dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("First", mag1, ang1);
  Dataset2 = QCChartTS.SimpleDataset.newSimpleDataset("Second", mag2, ang1);

  var datasetarray = [Dataset1, Dataset2];
  var pAntennaTransform = new QCChartTS.AntennaCoordinates();
  pAntennaTransform.autoScaleDatasetsRoundMode(datasetarray,
QCChartTS.ChartConstants.AUTOAXES_FAR);

  pAntennaTransform.setGraphBorderDiagonal(0.25, .15, .75, 0.85);

  var background = QCChartTS.Background.newBackground(pAntennaTransform,
QCChartTS.ChartConstants.GRAPH_BACKGROUND,
    QCChartTS.ChartColor.WHITE);
  chartVu.addChartObject(background);

  var pAntennaAxis = pAntennaTransform.getCompatibleAxes();
  pAntennaAxis.setLineColor(QCChartTS.ChartColor.BLACK);

  var axestickspace = 1;
  var axesntickspermajor = 5;
  var angletickspace = 5;
  var anglentickspermajor = 6;
  var minorticlength = 5;
  var majorticlength = 10;
  var tickdir = QCChartTS.ChartConstants.AXIS_CENTER;

  pAntennaAxis.setAntennaAxesTicksFull(axestickspace, axesntickspermajor,
    angletickspace, anglentickspermajor,
    minorticlength, majorticlength,
    tickdir);

  chartVu.addChartObject(pAntennaAxis);

  var pAntennaGrid = QCChartTS.AntennaGrid.newAntennaGrid(pAntennaAxis,
QCChartTS.ChartConstants.GRID_ALL);

pAntennaGrid.setChartObjAttributes(QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartCol
or.LIGHTBLUE, 1, QCChartTS.ChartConstants.LS_SOLID));
  chartVu.addChartObject(pAntennaGrid);

  var axisLabelsFont = QCChartTS.ChartFont.newChartFont("Microsoft Sans Serif",
QCChartTS.ChartFont.REGULAR, 24);

  var pAntennaAxisLabels = pAntennaAxis.getCompatibleAxesLabels();
  //      pAntennaAxisLabels.setTextFont(axisLabelsFont);
  chartVu.addChartObject(pAntennaAxisLabels);

  var transparentRed = QCChartTS.ChartSupport.fromArgb(180, 255, 0, 0);
  var transparentBlue = QCChartTS.ChartSupport.fromArgb(180, 0, 0, 255);
  var attrib1 = QCChartTS.ChartAttribute.newChartAttribute3(transparentRed, 1,
QCChartTS.ChartConstants.LS_SOLID);
  attrib1.setSymbolSize(7);
  var attrib2 = QCChartTS.ChartAttribute.newChartAttribute(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.BLUE);
  attrib2.setSymbolSize(7);

  var attrib3 = QCChartTS.ChartAttribute.newChartAttribute(QCChartTS.ChartColor.YELLOW, 3,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.YELLOW);
  var attrib4 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.MEDIUMPURPLE, 2,
QCChartTS.ChartConstants.LS_DOT_1_4);


  var thePlot1  =      QCChartTS.AntennaLinePlot.newAntennaLinePlotCoordinates(pAntennaTransform);
            thePlot1.initAntennaLinePlot(Dataset1, attrib1);
            chartVu.addChartObject(thePlot1);

  var thePlot2  =
    QCChartTS.AntennaScatterPlot.newAntennaScatterPlotCoordinates(pAntennaTransform);
            thePlot2.initAntennaScatterPlot(Dataset2, QCChartTS.ChartConstants.SQUARE, attrib2);
            chartVu.addChartObject(thePlot2);


  var thePlot3 = QCChartTS.AntennaAnnotation.newAntennaAnnotation(pAntennaTransform,
QCChartTS.ChartConstants.ANTENNA_ANNOTATION_ANGULAR, 180, attrib3);
  chartVu.addChartObject(thePlot3);


  var thePlot4 = QCChartTS.AntennaAnnotation.newAntennaAnnotation(pAntennaTransform,
QCChartTS.ChartConstants.ANTENNA_ANNOTATION_RADIUS, 12, attrib4);
  chartVu.addChartObject(thePlot4);
```

Class AntennaScatterPlot

```
GraphObj
       |
       +--ChartPlot
              |
              +--AntennaPlot
                     |
                     +-- AntennaScatterPlot
```

The **AntennaScatterPlot** class is a concrete implementation of the **AntennaPlot** class and displays data in a simple scatter plot format.

AntennaScatterPlot constructor

```
public static newAntennaScatterPlotCoords(transform: AntennaCoordinates): AntennaScatterPlot

public static newAntennaScatterPlot(transform: AntennaCoordinates,
        dataset: SimpleDataset, symtype: number,
        attrib: ChartAttribute): AntennaScatterPlot
```

transform	The coordinate system for the new **AntennaScatterPlot** object.
dataset	The antenna scatter plot represents the antenna coordinate values in this dataset. The x-values of the dataset represent the radial values of the points and the y-values the angular values in degrees.
symtype	The symbol used in the scatter plot. Use one of the scatter plot symbol constants: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE.
attrib	Specifies the attributes (size, line and fill color) for the scatter plot.

See previous example for a programming example using AttennaScatterPlot.

Class AntennaLineMarkerPlot

```
GraphObj
       |
       +--ChartPlot
              |
              +--AntennaPlot
                     |
                     +-- AntennaLineMarkerPlot
```

The **AntennaLineMarkerPlot** class is a concrete implementation of the **AntennaPlot** class and displays data in a simple line marker plot format.

AntennaScatterPlot constructor

```
public static newAntennaLineMarkerPlot(transform: PhysicalCoordinates,
        dataset: SimpleDataset, symtype: number, lineattrib: ChartAttribute,
        symbolattrib: ChartAttribute): AntennaLineMarkerPlot

public static newAntennaLineMarkerPlotCoords(transform: AntennaCoordinates):
AntennaLineMarkerPlot

public static newAntennaLineMarkerPlotCoordsDatasetAttribute(transform: AntennaCoordinates,
dataset: SimpleDataset, attrib: ChartAttribute): AntennaLineMarkerPlot

);
```

transform	The coordinate system for the new **AntennaLineMarkerPlot** object.

dataset	The line marker plot represents the values in this dataset.
symtype	The symbol used in the line marker plot. Use one of the scatter plot symbol constants: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE.
lineattrib	Specifies the attributes (line color and line style) for the line part of the line marker plot.
symbolattrib	Specifies the attributes (line and fill color) for the symbol part of the line marker plot.

Antenna line marker plot (extracted from the example program AntennaCharts.BuildAntennaLineChart)

[TypeScript]

```
let Dataset1: QCChartTS.SimpleDataset;
let Dataset2: QCChartTS.SimpleDataset;

let theFont: QCChartTS.ChartFont;
let num: number = 61;

let mag1: number[] = new Array(num);
let mag2: number[] = new Array(num);
let ang1: number[] = new Array(num);
let i: number;
for (i = 0; i < num - 1; i++) {
.
.
.

}
// close loop
mag1[num - 1] = mag1[0];
mag2[num - 1] = mag2[0];
ang1[num - 1] = ang1[0];


theFont = QCChartTS.ChartFont.newChartFont("Microsoft Sans Serif", QCChartTS.ChartFont.BOLD, 10);
Dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("First", mag1, ang1);
Dataset2 = QCChartTS.SimpleDataset.newSimpleDataset("Second", mag2, ang1);

let datasetarray: QCChartTS.SimpleDataset[] = [Dataset1, Dataset2];
this.pAntennaTransform = new QCChartTS.AntennaCoordinates();
this.pAntennaTransform.autoScaleDatasetsRoundMode(datasetarray,
QCChartTS.ChartConstants.AUTOAXES_FAR);

this.pAntennaTransform.setGraphBorderDiagonal(0.25, .15, .75, 0.85);

let background: QCChartTS.Background = QCChartTS.Background.newBackground(this.pAntennaTransform,
QCChartTS.ChartConstants.GRAPH_BACKGROUND,
  QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(background);

let pAntennaAxis: QCChartTS.AntennaAxes = this.pAntennaTransform.getCompatibleAxes();
pAntennaAxis.setLineColor(QCChartTS.ChartColor.BLACK);

let axestickspace: number = 1;
let axesntickspermajor: number = 5;
let angletickspace: number = 5;
let anglentickspermajor: number = 6;
let minorticlength: number = 5;
let majorticlength: number = 10;
let tickdir: number = QCChartTS.ChartConstants.AXIS_CENTER;

pAntennaAxis.setAntennaAxesTicksFull(axestickspace, axesntickspermajor,
  angletickspace, anglentickspermajor,
  minorticlength, majorticlength,
  tickdir);

chartVu.addChartObject(pAntennaAxis);

let pAntennaGrid: QCChartTS.AntennaGrid = QCChartTS.AntennaGrid.newAntennaGrid(pAntennaAxis,
QCChartTS.ChartConstants.GRID_ALL);
pAntennaGrid.setChartObjAttributes(QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartCol
or.LIGHTBLUE, 1, QCChartTS.ChartConstants.LS_SOLID));
chartVu.addChartObject(pAntennaGrid);

let axisLabelsFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont("Microsoft Sans
Serif", QCChartTS.ChartFont.REGULAR, 24);

let pAntennaAxisLabels: QCChartTS.AntennaAxesLabels =
<QCChartTS.AntennaAxesLabels>pAntennaAxis.getCompatibleAxesLabels();
//  pAntennaAxisLabels.setTextFont(axisLabelsFont);
chartVu.addChartObject(pAntennaAxisLabels);

let transparentRed: number = QCChartTS.ChartSupport.fromArgb(180, 255, 0, 0);
let transparentBlue: number = QCChartTS.ChartSupport.fromArgb(180, 0, 0, 255);


let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(transparentRed, 1,
QCChartTS.ChartConstants.LS_SOLID);
attrib1.setSymbolSize(7);
let attrib2: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.BLUE);
attrib2.setSymbolSize(7);

let attrib3: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute(QCChartTS.ChartColor.YELLOW, 3,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.YELLOW);
let attrib4: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.MEDIUMPURPLE, 2,
QCChartTS.ChartConstants.LS_DOT_1_4);


let thePlot1: QCChartTS.AntennaLineMarkerPlot =
QCChartTS.AntennaLineMarkerPlot.newAntennaLineMarkerPlotCoordsDatasetAttribute(this.pAntennaTrans
form, Dataset1, attrib1);
chartVu.addChartObject(thePlot1);

let thePlot2: QCChartTS.AntennaLineMarkerPlot =
QCChartTS.AntennaLineMarkerPlot.newAntennaLineMarkerPlotCoordsDatasetAttribute(this.pAntennaTrans
form, Dataset2, attrib2);
chartVu.addChartObject(thePlot2);

let thePlot3: QCChartTS.AntennaAnnotation =
QCChartTS.AntennaAnnotation.newAntennaAnnotation(this.pAntennaTransform,
QCChartTS.ChartConstants.ANTENNA_ANNOTATION_ANGULAR, 180, attrib3);
chartVu.addChartObject(thePlot3);
```

```
let thePlot4: QCChartTS.AntennaAnnotation =
QCChartTS.AntennaAnnotation.newAntennaAnnotation(this.pAntennaTransform,
QCChartTS.ChartConstants.ANTENNA_ANNOTATION_RADIUS, 12, attrib4);
chartVu.addChartObject(thePlot4);
```

[JavaScript]

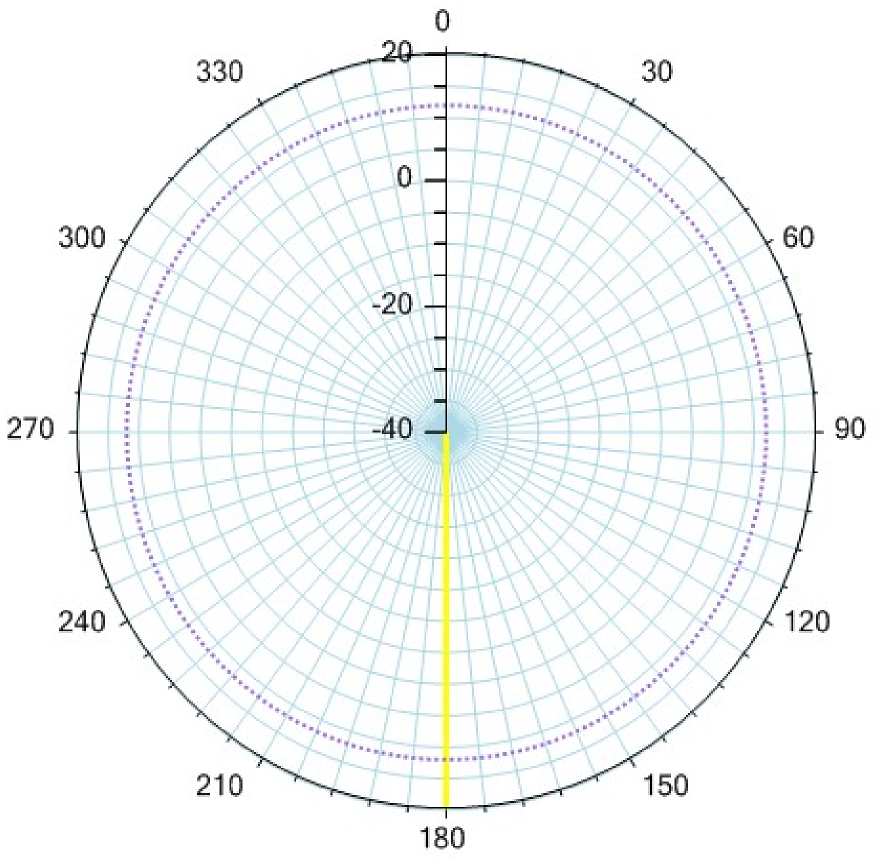
```
 var Dataset1;
  var Dataset2;

  var theFont;
  var num = 61;

  var mag1 = new Array(num);
  var mag2 = new Array(num);
  var ang1 = new Array(num);
  var i;
  for (i = 0; i < num - 1; i++) {
  .
  .
  .

  }
  // close loop
  mag1[num - 1] = mag1[0];
  mag2[num - 1] = mag2[0];
  ang1[num - 1] = ang1[0];



  theFont = QCChartTS.ChartFont.newChartFont("Microsoft Sans Serif", QCChartTS.ChartFont.BOLD,
10);
  Dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("First", mag1, ang1);
  Dataset2 = QCChartTS.SimpleDataset.newSimpleDataset("Second", mag2, ang1);

  var datasetarray = [Dataset1, Dataset2];
  var pAntennaTransform = new QCChartTS.AntennaCoordinates();
  pAntennaTransform.autoScaleDatasetsRoundMode(datasetarray,
QCChartTS.ChartConstants.AUTOAXES_FAR);

  pAntennaTransform.setGraphBorderDiagonal(0.25, .15, .75, 0.85);

  var background = QCChartTS.Background.newBackground(pAntennaTransform,
QCChartTS.ChartConstants.GRAPH_BACKGROUND,
    QCChartTS.ChartColor.WHITE);
  chartVu.addChartObject(background);

  var pAntennaAxis = pAntennaTransform.getCompatibleAxes();
  pAntennaAxis.setLineColor(QCChartTS.ChartColor.BLACK);

  var axestickspace = 1;
  var axesntickspermajor = 5;
  var angletickspace = 5;
  var anglentickspermajor = 6;
  var minorticlength = 5;
  var majorticlength = 10;
  var tickdir = QCChartTS.ChartConstants.AXIS_CENTER;

  pAntennaAxis.setAntennaAxesTicksFull(axestickspace, axesntickspermajor,
    angletickspace, anglentickspermajor,
    minorticlength, majorticlength,
    tickdir);

  chartVu.addChartObject(pAntennaAxis);

  var pAntennaGrid = QCChartTS.AntennaGrid.newAntennaGrid(pAntennaAxis,
QCChartTS.ChartConstants.GRID_ALL);
```

```
pAntennaGrid.setChartObjAttributes(QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartCol
or.LIGHTBLUE, 1, QCChartTS.ChartConstants.LS_SOLID));
  chartVu.addChartObject(pAntennaGrid);

  var axisLabelsFont = QCChartTS.ChartFont.newChartFont("Microsoft Sans Serif",
QCChartTS.ChartFont.REGULAR, 24);

  var pAntennaAxisLabels = pAntennaAxis.getCompatibleAxesLabels();
  //      pAntennaAxisLabels.setTextFont(axisLabelsFont);
  chartVu.addChartObject(pAntennaAxisLabels);

  var transparentRed = QCChartTS.ChartSupport.fromArgb(180, 255, 0, 0);
  var transparentBlue = QCChartTS.ChartSupport.fromArgb(180, 0, 0, 255);


  var attrib1 = QCChartTS.ChartAttribute.newChartAttribute3(transparentRed, 1,
QCChartTS.ChartConstants.LS_SOLID);
  attrib1.setSymbolSize(7);
  var attrib2 = QCChartTS.ChartAttribute.newChartAttribute(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.BLUE);
  attrib2.setSymbolSize(7);

  var attrib3 = QCChartTS.ChartAttribute.newChartAttribute(QCChartTS.ChartColor.YELLOW, 3,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.YELLOW);
  var attrib4 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.MEDIUMPURPLE, 2,
QCChartTS.ChartConstants.LS_DOT_1_4);


  var thePlot1 =
QCChartTS.AntennaLineMarkerPlot.newAntennaLineMarkerPlotCoordsDatasetAttribute(pAntennaTransform,
Dataset1, attrib1);
  chartVu.addChartObject(thePlot1);

  var thePlot2 =
QCChartTS.AntennaLineMarkerPlot.newAntennaLineMarkerPlotCoordsDatasetAttribute(pAntennaTransform,
Dataset2, attrib2);
  chartVu.addChartObject(thePlot2);

  var thePlot3 = QCChartTS.AntennaAnnotation.newAntennaAnnotation(pAntennaTransform,
QCChartTS.ChartConstants.ANTENNA_ANNOTATION_ANGULAR, 180, attrib3);
  chartVu.addChartObject(thePlot3);


  var thePlot4 = QCChartTS.AntennaAnnotation.newAntennaAnnotation(pAntennaTransform,
QCChartTS.ChartConstants.ANTENNA_ANNOTATION_RADIUS, 12, attrib4);
  chartVu.addChartObject(thePlot4);
```

Class AntennaAnnotation

GraphObj
```
   |
   +--AntennaAnnotation
```

The **AntennaAnnotation** class is used to highlight either a specific radius or angular value in an antenna chart. The radius is highlighted using a circle, and the angular value is highlighted using a line draw from the origin to the outer edge of the antenna chart.



Two annotations – a radius annotations (dotted blue line) at the radial value 12, and an angular annotations (solid yellow line) at the angular value 180 degrees.

AntennaAnnotationPlot constructor

```
public static newAntennaAnnotation(transform: AntennaCoordinates, annotationtype: number, value:
number, attrib: ChartAttribute): AntennaAnnotation
```

transform	The coordinate system for the new **AntennaScatterPlot** object.
annotationtype	The annotation type. Use one of the annotation type constants: ANTENNA_ANNOTATION_ANGULAR (draws a radial line at the specified angular value, from the origin to the outer edge of the antenna chart, or ANTENNA_ANNOTATION_RADIUS (draws a circle at the specified radius value).
value	The value of the annotation. For an angular annotation, specify the value in degrees. For a radial annotation, specify a value within the range of the antenna minimum and maximum radial values.
attrib	Specifies the attributes (size, line and fill color) for the annotation.

See previous example for a programming example using AntennaAnnotationPlot.

19. Legends

Legend
    StandardLegend
    BubblePlotLegend

Charts containing multiple chart objects, line plots, bar graphs and scatter plots for example, usually require a legend. The legend provides a key so that the viewer of the chart can figure out what data is associated with what chart object. The bounding box of the legend is rectangular and can reside anywhere in the chart window: inside the plot area, overlapping it or completely outside. The legend rectangle can have a border and can be filled with a solid color or left transparent. The legend object can hold one or more legend items, where each legend item is a symbol-text string combination providing the key for one of the plot objects in the graph. The legend can also have a title and footer.

The **Legend** class is the abstract base class for chart legends. It organizes a collection of legend items as a rectangular object.

The **StandardLegend** is a concrete implementation of the **Legend** class and it is the primary legend class for all plot objects except for bubble plots. The legend items objects display in a row or column format. Each legend item contains a symbol and descriptive string. The symbol normally associates the legend item to a particular plot object, and the descriptive string describes what the plot object represents.

The **BubblePlotLegend** is a concrete implementation of the **Legend** class and it is the legend class for bubble plots. The legend items objects display as offset, concentric circles with descriptive text giving the key for the value associated with a bubble of this size.

Standard Legends

Class StandardLegend

GraphObj
```
    |
    +-- StandardLegend
```

The **StandardLegend** is the primary legend class for all plot objects except for bubble plots. The class manages a list of **LegendItems** that holds the symbols and descriptive text for the symbols.

StandardLegend constructors

```
public static newStandardLegend(rx: number, ry: number, rwidth: number, rheight: number, attrib:
ChartAttribute, nlayout1mode: number): StandardLegend

public static newStandardLegendPosWHAttribLayout(rx: number, ry: number, rwidth: number, rheight:
number, attrib: ChartAttribute, nlayout1mode: number): StandardLegend

public static newStandardLegendPosAttribLayout(rx: number, ry: number,
        attrib: ChartAttribute, nlayout1mode: number): StandardLegend
```

rx	The x-position, in chart normalized coordinates, of the legend rectangle.
ry	The y-position, in chart normalized coordinates, of the legend rectangle.
rwidth	The width, in chart normalized coordinates, of the legend rectangle.
rheight	The height, in chart normalized coordinates, of the legend rectangle.
attrib	Specifies the outline color, outline line style, and fill color for the legend rectangle.
nlayoutmode	Specifies if the legend has a horizontal, or vertical layout. Use one of the orientation constants: HORIZ_DIR (row major) or VERT_DIR (column major).

Add legend items to a legend using one of the **addLegendItem** methods.

addLegendItem methods

```
public addLegendItemStringSymbolNumAttribFont(stext: string, nsymbol: number, attrib: 
ChartAttribute, thefont: ChartFont): number

public addLegendItemStringSymbolPathAttribFont(stext: string, symbolshape: GPath, attrib: 
ChartAttribute, thefont: ChartFont): number 

public addLegendItemStringSymbolNumGraphObjFont(stext: string, nsymbol: number, chartobj: 
GraphObj, thefont: ChartFont): number

public addLegendItemStringGraphObjFont(stext: string, chartobj: GraphObj, thefont: ChartFont): 
number

public addLegendItemStringSymbolPathGraphObjFont(stext: string, symbolshape: GPath, chartobj: 
GraphObj, thefont: ChartFont): number 

public addLegendItemStringSymbolNumChartPlotGroupFont(stext: string, nsymbol: number, chartobj: 
ChartPlot, ngroup: number, thefont: ChartFont): number
```

stext	Specifies the text string for the legend item.
nsymbol	Specifies the symbol for the legend item. Use one of the chart symbol constants: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, or CIRCLE.
chartobj	The color and fill attributes for the legend item are copied from the attributes of this **ChartPlot** object.
symbolshape	Specifies a user defined shape to use as the legend item symbol.
attrib	Specifies the **ChartAttribute** object to get the color and fill attributes of the legend item.
thefont	Specifies the text font for the legend item.

The addLegendItem returns the current number of legend items.

Simple legend example (extracted from the example program SimpleLinePlots.BuildLineFill.
[TypeScript]

```
.
.
.
let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 3,
QCChartTS.ChartConstants.LS_SOLID);
Dataset1.sortByX(true);
thePlot1 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, Dataset1, attrib1);
thePlot1.setLineStyle(QCChartTS.ChartConstants.LS_DASH_DOT);
chartVu.addChartObject(thePlot1);
let attrib2: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.YELLOW, 3,
QCChartTS.ChartConstants.LS_SOLID);
Dataset2.sortByX(true);
thePlot2 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, Dataset2, attrib2);
chartVu.addChartObject(thePlot2);
.
.
.
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.fromRgb(155, 155, 155));
legendAttributes.setFillFlag(true);
legendAttributes.setLineFlag(true);
let legend: QCChartTS.StandardLegend =
QCChartTS.StandardLegend.newStandardLegendPosWHAttribLayout(0.1, 0.9, 0.6, 0.075,
legendAttributes, QCChartTS.ChartConstants.HORIZ_DIR);
legend.addLegendItemStringSymbolNumGraphObjFont("Expenses", QCChartTS.ChartConstants.LINE,
thePlot1, legendFont);
legend.addLegendItemStringSymbolNumGraphObjFont("Revenue", QCChartTS.ChartConstants.LINE,
thePlot2, legendFont);
```

```
chartVu.addChartObject(legend);
```

[JavaScript]

```
let attrib1 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 3,
QCChartTS.ChartConstants.LS_SOLID);
Dataset1.sortByX(true);
thePlot1 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, Dataset1, attrib1);
thePlot1.setLineStyle(QCChartTS.ChartConstants.LS_DASH_DOT);
chartVu.addChartObject(thePlot1);
let attrib2 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.YELLOW, 3,
QCChartTS.ChartConstants.LS_SOLID);
Dataset2.sortByX(true);
thePlot2 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, Dataset2, attrib2);
chartVu.addChartObject(thePlot2);
let transparentRed = QCChartTS.ChartColor.fromArgb(200, 255, 0, 0);
let transparentGreen = QCChartTS.ChartColor.fromArgb(200, 0, 255, 0);
let lossAttrib = QCChartTS.ChartAttribute.newChartAttribute4(transparentRed, 1,
QCChartTS.ChartConstants.LS_SOLID, transparentRed);
let profitAttrib = QCChartTS.ChartAttribute.newChartAttribute4(transparentGreen, 1,
QCChartTS.ChartConstants.LS_SOLID, transparentGreen);
profitAttrib.setFillFlag(true);
lossAttrib.setFillFlag(true);
profitAttrib.setLineFlag(false);
lossAttrib.setLineFlag(false);
.
.
.


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.fromRgb(155, 155, 155));
legendAttributes.setFillFlag(true);
legendAttributes.setLineFlag(true);
let legend = QCChartTS.StandardLegend.newStandardLegendPosWHAttribLayout(0.1, 0.9, 0.6, 0.075,
legendAttributes, QCChartTS.ChartConstants.HORIZ_DIR);
legend.addLegendItemStringSymbolNumGraphObjFont("Expenses", QCChartTS.ChartConstants.LINE,
thePlot1, legendFont);
legend.addLegendItemStringSymbolNumGraphObjFont("Revenue", QCChartTS.ChartConstants.LINE,
thePlot2, legendFont);

chartVu.addChartObject(legend);
```

Bubble Plot Legends

Class BubblePlotLegend

```
GraphObj
   |
   +-- BubblePlotLegend
```

The **BubblePlotLegend** is the primary legend class for bubble plots. The class manages a list of **BubblePlotLegendItem** that holds the symbols and descriptive text for the symbols.

BubblePlotLegend constructors

```
public static newBubblePlotLegendWH(plot: BubblePlot, rx: number, ry: number, rwidth: number,
rheight: number, attrib: ChartAttribute): BubblePlotLegend

public static newBubblePlotLegend(plot: BubblePlot, rx: number, ry: number, attrib:
ChartAttribute): BubblePlotLegend
```

plot	The bubble plot object the legend is associated with.
rx	The x-position, in chart normalized coordinates, of the legend rectangle.
ry	The y-position, in chart normalized coordinates, of the legend rectangle.
rwidth	The width, in chart normalized coordinates, of the legend rectangle.
rheight	The height, in chart normalized coordinates, of the legend rectangle.
attrib	Specifies the outline color, outline line style, and fill color for the legend rectangle.

Add legend items to a legend using one of the addLegendItem methods.

addLegendItem methods

```
public addLegendItemTextSizeAttrib(stext: string, rsize: number, attrib: ChartAttribute, thefont:
ChartFont): number

public addLegendItem(legenditem: BubblePlotLegendItem): number

public addLegendItemTextSizeObjFont(stext: string, rsize: number, chartobj: BubblePlot, thefont:
ChartFont): number
```

stext	Specifies the text string for the legend item.
rsize	Specifies the size of the bubble for this item, in the same units as the coordinate system the bubble plot is placed in.
chartobj	The color and fill attributes for the legend item are copied from the attributes of this chart object.
thefont	Specifies the text font for the legend item.

The method returns the current number of legend items.

Simple legend example (extracted from the example program ScatterPlots.BuildRealGDPGrowth)

[TypeScript]

```
.
.
.


let attrib2: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLACK, 3,
QCChartTS.ChartConstants.LS_SOLID);
let legendFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Arial",
QCChartTS.ChartFont.BOLD, 12);
let legendAttributes: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 3,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.WHITE);
legendAttributes.setFillFlag(true);
legendAttributes.setLineFlag(false);

let bubblelegend: QCChartTS.BubblePlotLegend =
QCChartTS.BubblePlotLegend.newBubblePlotLegendWH(thePlot1, 0.79, 0.11, 0.16, 0.27,
legendAttributes);
bubblelegend.addLegendItemTextSizeAttrib("1B ", 10, attrib2, legendFont);
bubblelegend.addLegendItemTextSizeAttrib("500M", 5, attrib2, legendFont);
bubblelegend.addLegendItemTextSizeAttrib("100M", 1, attrib2, legendFont);
bubblelegend.addLegendGeneralTextPosStringColorFont(QCChartTS.ChartConstants.LEGEND_HEADER,
"Population Key", QCChartTS.ChartColor.BLACK, legendFont);
chartVu.addChartObject(bubblelegend);
```

[JavaScript]

```
.
.
.
let attrib2 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLACK, 3,
QCChartTS.ChartConstants.LS_SOLID);
let legendFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
let legendAttributes = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 3,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.WHITE);
legendAttributes.setFillFlag(true);
legendAttributes.setLineFlag(false);

let bubblelegend = QCChartTS.BubblePlotLegend.newBubblePlotLegendWH(thePlot1, 0.79, 0.11, 0.16,
0.27, legendAttributes);
bubblelegend.addLegendItemTextSizeAttrib("1B ", 10, attrib2, legendFont);
bubblelegend.addLegendItemTextSizeAttrib("500M", 5, attrib2, legendFont);
bubblelegend.addLegendItemTextSizeAttrib("100M", 1, attrib2, legendFont);
bubblelegend.addLegendGeneralTextPosStringColorFont(QCChartTS.ChartConstants.LEGEND_HEADER,
"Population Key", QCChartTS.ChartColor.BLACK, legendFont);
chartVu.addChartObject(bubblelegend);
```

20. Text Classes

ChartText
- **ChartTitle**
- **AxisTitle**
- **ChartLabel**
  - **StringLabel**
  - **TimeLabel**
  - **NumericLabel**
  - **ElapsedTimeLabel**

The software uses the **ChartText** classes to position and format text in a chart. Examples of classes derived from the **ChartText** include the **ChartLabel**, **AxisLabels**, **ChartTitle**, and **AxisTitle** classes. The **Legend, PieChart** and **ChartPlot** classes, while not derived from the text classes, use them internally.

ChartFont

The standard text font handling of HTML5 does not match that of other platforms we support: .Net, WPF, and Java. So we create a wrapper class (ChartFont) for HTML5 font properties (name, style and size) to make the font handling similar to these other languages. So text objects will always contain a reference to a ChartFont object to use when drawing the text.

Class ChartFont

```
public static newChartFont(family: string, style: number, size: number): ChartFont
public static newChartFont3(family: string, style: number, size: number): ChartFont
public static newChartFont1(family: string): ChartFont
public static newChartFont2(family: string, size: number): ChartFont
```

family	the font name
style	the font style (ChartFont.NORMAL, BOLD, PLAIN, ITALIC, BOLD_ITALIC)
size	the font size
src	the source font to copy

Note: ChartFont argument order – The argument order for the default ChartFont constructor is (*name, style, size)*. This is similar to the Java Font class, but different than the .Net Font class, which is *(name, size, style)*. Because the last two arguments (style and size) are both of type number, it is easy to swap their order. If you do that the text will not display correctly, with the spacing between lines especially bad. But correcting the argument order will correct it.

Simple Text Classes

Class ChartText

GraphObj
```
|
+--ChartText
```

The **ChartText** class is the base class for all text output classes. The **ChartText** class formats and places text in a chart. Position the **ChartText** objects using any of the coordinate systems. Rotate and justify the text vertically and horizontally. Insert a CR (carriage return, ASCII 13) character at line breaks for multiline text. The most common constructors are:

ChartText constructors

```
public static newChartText(transform: PhysicalCoordinates, tfont: ChartFont, tstring: string,
    x: number, y: number, npostype: number): ChartText

public static newChartTextCoords(transform: PhysicalCoordinates): ChartText

public static newChartText9(transform: PhysicalCoordinates, tfont: ChartFont, tstring: string,
        x: number, y: number,
        npostype: number,
        xjust: number, yjust: number, rotation: number): ChartText

public static newChartTextCoordsFontStringJustRot(transform: PhysicalCoordinates, tfont:
ChartFont, tstring: string,
        x: number, y: number,
        npostype: number,
        xjust: number, yjust: number, rotation: number): ChartText

public static newChartTextCoordsFontStringPosType(transform: PhysicalCoordinates, tfont:
ChartFont, tstring: string,
        x: number, y: number, npostype: number): ChartText

public static newChartTextCoordsFontString(transform: PhysicalCoordinates, tfont: ChartFont,
tstring: string): ChartText

public static newChartTextFontString(tfont: ChartFont, tstring: string): ChartText
```

transform	Places the text in the coordinate system defined by transform.
tfont	A reference to a ChartFont object.
tstring	A reference to a string object.

x	Specifies the x-value of the text position
y	Specifies the y-value of the text position
npostype	Specifies the if the position of the text is specified in physical coordinates, normalized coordinates or Canvas device coordinates. Use one of the position constants: DEV_POS, PHYS_POS, NORM_GRAPH_POS, NORM_PLOT_POS.
xjust	Specifies the horizontal justification of the text. Use one of the text justification constants: JUSTIFY_MIN, JUSTIFY_CENTER or JUSTIFY_MAX.
yjust	Specifies the vertical justification of the text. Use one of the text justification constants: JUSTIFY_MIN, JUSTIFY_CENTER or JUSTIFY_MAX.
rotation	The rotation (-360 to 360 degrees) of the text in the normal viewing plane.

Place text in a time coordinate system (**TimeCoordinates**) by converting the time x-position to milliseconds and using the milliseconds as the x-position value.

ChartText example (extracted from the example program MultiLinePlots.BuildMultiLines)

[TypeScript]

```
.
.
.

let theLabelFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Arial",
QCChartTS.ChartFont.BOLD, 12);
let currentLabel1: QCChartTS.ChartText =
QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1, theLabelFont, "I(b) = 50uA",
15.5, (y1[0][85] + 1), QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(currentLabel1);
let currentLabel2: QCChartTS.ChartText =
QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1, theLabelFont, "I(b) =
100uA", 15.5, (y1[1][85] + 1), QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(currentLabel2);
let currentLabel3: QCChartTS.ChartText =
QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1, theLabelFont, "I(b) =
150uA", 15.5, (y1[2][85] + 1), QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(currentLabel3);
let currentLabel4: QCChartTS.ChartText =
QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1, theLabelFont, "I(b) =
200uA", 15.5, (y1[3][85] + 1), QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(currentLabel4);
let currentLabel5: QCChartTS.ChartText =
QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1, theLabelFont, "I(b) =
250uA", 15.5, (y1[4][85] + 1), QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(currentLabel5);
let currentLabel6: QCChartTS.ChartText =
QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1, theLabelFont, "I(b) =
300uA", 15.5, (y1[5][85] + 1), QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(currentLabel6);
let currentLabel7: QCChartTS.ChartText =
QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1, theLabelFont, "I(b) =
350uA", 15.5, (y1[6][85] + 1), QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(currentLabel7);
```

```
let regionLabel: QCChartTS.ChartText =
QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1, theLabelFont, "Linear" + "("
+ "Region", 4, 40, QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(regionLabel);
```

[JavaScript]

```
let theLabelFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
let currentLabel1 = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1,
theLabelFont, "I(b) = 50uA", 15.5, (y1[0][85] + 1), QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(currentLabel1);
let currentLabel2 = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1,
theLabelFont, "I(b) = 100uA", 15.5, (y1[1][85] + 1), QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(currentLabel2);
let currentLabel3 = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1,
theLabelFont, "I(b) = 150uA", 15.5, (y1[2][85] + 1), QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(currentLabel3);
let currentLabel4 = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1,
theLabelFont, "I(b) = 200uA", 15.5, (y1[3][85] + 1), QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(currentLabel4);
let currentLabel5 = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1,
theLabelFont, "I(b) = 250uA", 15.5, (y1[4][85] + 1), QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(currentLabel5);
let currentLabel6 = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1,
theLabelFont, "I(b) = 300uA", 15.5, (y1[5][85] + 1), QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(currentLabel6);
let currentLabel7 = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1,
theLabelFont, "I(b) = 350uA", 15.5, (y1[6][85] + 1), QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(currentLabel7);
let regionLabel = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1,
theLabelFont, "Linear" + "(" + "Region", 4, 40, QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(regionLabel);
```

ChartText time coordinates example (extracted from the example program MiscCharts.BuildLineGapPlot)

[TypeScript]

```
let theLabelFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.BOLD, 16);
let chartLabel1: QCChartTS.ChartText =
QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1, theLabelFont, "Sales",
QCChartTS.ChartCalendar.getCalendarMsecs(xValues[1]), groupBarData[1]
[1],QCChartTS.ChartConstants.PHYS_POS);
chartLabel1.setColor(QCChartTS.ChartColor.WHITE);
chartLabel1.setYJust(QCChartTS.ChartConstants.AXIS_MIN);
chartVu.addChartObject(chartLabel1);
let chartLabel2: QCChartTS.ChartText =
QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1, theLabelFont, "Costs",
QCChartTS.ChartCalendar.getCalendarMsecs(xValues[1]), groupBarData[0]
[ 1],QCChartTS.ChartConstants.PHYS_POS);
chartLabel2.setColor(QCChartTS.ChartColor.WHITE);
chartLabel2.setYJust(QCChartTS.ChartConstants.AXIS_MAX);
chartVu.addChartObject(chartLabel2);
let chartLabel3: QCChartTS.ChartText =
QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1, theLabelFont, "Profits",
QCChartTS.ChartCalendar.getCalendarMsecs(xValues[1]), ((groupBarData[0][ 1] + groupBarData[1][1])
/ 2),QCChartTS.ChartConstants.PHYS_POS);
chartLabel3.setColor(QCChartTS.ChartColor.WHITE);
chartLabel3.setYJust(QCChartTS.ChartConstants.AXIS_CENTER);
chartVu.addChartObject(chartLabel3);
let chartLabel4: QCChartTS.ChartText =
QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1, theLabelFont, "Loss",
QCChartTS.ChartCalendar.getCalendarMsecs(xValues[3]), ((groupBarData[0][ 3] + groupBarData[1][3])
/ 2),QCChartTS.ChartConstants.PHYS_POS);
chartLabel4.setColor(QCChartTS.ChartColor.WHITE);
```

```
chartLabel4.setYJust(QCChartTS.ChartConstants.AXIS_CENTER);
chartVu.addChartObject(chartLabel4);
```

[JavaScript]

```
let theLabelFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.BOLD, 16);
let chartLabel1 = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1,
theLabelFont, "Sales", QCChartTS.ChartCalendar.getCalendarMsecs(xValues[1]), groupBarData[1][1],
QCChartTS.ChartConstants.PHYS_POS);
chartLabel1.setColor(QCChartTS.ChartColor.WHITE);
chartLabel1.setYJust(QCChartTS.ChartConstants.AXIS_MIN);
chartVu.addChartObject(chartLabel1);
let chartLabel2 = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1,
theLabelFont, "Costs", QCChartTS.ChartCalendar.getCalendarMsecs(xValues[1]), groupBarData[0][1],
QCChartTS.ChartConstants.PHYS_POS);
chartLabel2.setColor(QCChartTS.ChartColor.WHITE);
chartLabel2.setYJust(QCChartTS.ChartConstants.AXIS_MAX);
chartVu.addChartObject(chartLabel2);
let chartLabel3 = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1,
theLabelFont, "Profits", QCChartTS.ChartCalendar.getCalendarMsecs(xValues[1]), ((groupBarData[0]
[1] + groupBarData[1][1]) / 2), QCChartTS.ChartConstants.PHYS_POS);
chartLabel3.setColor(QCChartTS.ChartColor.WHITE);
chartLabel3.setYJust(QCChartTS.ChartConstants.AXIS_CENTER);
chartVu.addChartObject(chartLabel3);
let chartLabel4 = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1,
theLabelFont, "Loss", QCChartTS.ChartCalendar.getCalendarMsecs(xValues[3]), ((groupBarData[0][3]
+ groupBarData[1][3]) / 2), QCChartTS.ChartConstants.PHYS_POS);
chartLabel4.setColor(QCChartTS.ChartColor.WHITE);
chartLabel4.setYJust(QCChartTS.ChartConstants.AXIS_CENTER);
chartVu.addChartObject(chartLabel4);
```

Chart Title Classes

Class ChartTitle

ChartText

```
   |
   +--ChartTitle
```

The **ChartTitle** class creates a header, subheader or footer for a chart. The most common constructors are:

ChartTitle constructors

```
public static newChartTitleCoordsFontString(transform: PhysicalCoordinates, tfont: ChartFont,
tstring: string): ChartTitle

public static newChartTitle(transform: PhysicalCoordinates, tfont: ChartFont, tstring: string,
        ntitletype: number, ntitlepos: number): ChartTitle
```

transform	Places the text in the coordinate system defined by transform.
tfont	A reference to a **ChartFont** object.

tstring	A reference to a string object.
ntitletype	The title can be a header, subhead or footer. Use one of the title type constants: CHART_HEADER, CHART_SUBHEAD or CHART_FOOTER.
ntitlepos	The title can be centered with respect to the entire graph area, or the plot area. Use one of the title position constants: CENTER_GRAPH or CENTER_PLOT.

ChartTitle example (extracted from the example program SimpleLinePlots.BuildLineFill)

[TypeScript]

```
.
.
.
let theTitleFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Arial",
QCChartTS.ChartFont.BOLD, 16);
let mainTitle: QCChartTS.ChartTitle =
QCChartTS.ChartTitle.newChartTitleCoordsFontString(pTransform1, theTitleFont, "Profits are
Expected to Rise");
mainTitle.setTitleType(QCChartTS.ChartConstants.CHART_HEADER);
mainTitle.setTitlePosition(QCChartTS.ChartConstants.CENTER_GRAPH);
mainTitle.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(mainTitle);

let theFooterFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Arial",
QCChartTS.ChartFont.BOLD, 12);
let footer: QCChartTS.ChartTitle =
QCChartTS.ChartTitle.newChartTitleCoordsFontString(pTransform1, theFooterFont, "Graphs can have
background gradients, semi-transparent colors, legends, titles and data tooltips.");
footer.setTitleType(QCChartTS.ChartConstants.CHART_FOOTER);
footer.setTitlePosition(QCChartTS.ChartConstants.CENTER_GRAPH);
footer.setTitleOffset(8);
footer.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(footer);
```

[JavaScript]

```
let theTitleFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 16);
let mainTitle = QCChartTS.ChartTitle.newChartTitleCoordsFontString(pTransform1, theTitleFont,
"Profits are Expected to Rise");
mainTitle.setTitleType(QCChartTS.ChartConstants.CHART_HEADER);
mainTitle.setTitlePosition(QCChartTS.ChartConstants.CENTER_GRAPH);
mainTitle.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(mainTitle);

let theFooterFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
let footer = QCChartTS.ChartTitle.newChartTitleCoordsFontString(pTransform1, theFooterFont,
"Graphs can have background gradients, semi-transparent colors, legends, titles and data
tooltips.");
footer.setTitleType(QCChartTS.ChartConstants.CHART_FOOTER);
footer.setTitlePosition(QCChartTS.ChartConstants.CENTER_GRAPH);
footer.setTitleOffset(8);
footer.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(footer);
```

Class AxisTitle

ChartText

|
+--AxisTitle

The **AxisTitle** class creates a title for an axis. The text is horizontal for x-axis titles and vertical for y-axis titles. The most common constructor is:

AxisTitle Constructor

```
public static newAxisTitle(axis: Axis, thefont: ChartFont, s: string): AxisTitle
```

axis	The base axis this title is associated with.
thefont	The font object used to display the axis title.
s	Sets the title string.

ChartTitle example (extracted from the example program ScatterPlots.BuildLabeledDatapoints)

[TypeScript]

```
.
.
.
let xAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.X_AXIS);
chartVu.addChartObject(xAxis);
let yAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
chartVu.addChartObject(yAxis);
let xAxisLab: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(xAxis);
xAxisLab.setTextFont(theFont);
chartVu.addChartObject(xAxisLab);
let yAxisLab: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(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(yAxis, titleFont, "Test
Score");
chartVu.addChartObject(yaxistitle);
let xaxistitle: QCChartTS.AxisTitle = QCChartTS.AxisTitle.newAxisTitle(xAxis, titleFont, "Student
#");
```

[JavaScript]

```
let xAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.X_AXIS);
chartVu.addChartObject(xAxis);
let yAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.Y_AXIS);
chartVu.addChartObject(yAxis);
```

```
let xAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(xAxis);
xAxisLab.setTextFont(theFont);
chartVu.addChartObject(xAxisLab);
let yAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(yAxis);
yAxisLab.setTextFont(theFont);
chartVu.addChartObject(yAxisLab);
let titleFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
let yaxistitle = QCChartTS.AxisTitle.newAxisTitle(yAxis, titleFont, "Test Score");
chartVu.addChartObject(yaxistitle);
let xaxistitle = QCChartTS.AxisTitle.newAxisTitle(xAxis, titleFont, "Student #");
chartVu.addChartObject(xaxistitle);
```

Numeric, Time, Elapsed Time and String Label Classes

Class ChartLabel

```
ChartText
       |
       +-- ChartLabel
              |
              +--StringLabel
              |
              +--TimeLabel
              |
              +--ElapsedTimeLabel
              |
              +--NumericLabel
```

The **ChartLabel** class is the abstract base class for all of the formatted label classes. The axis label classes use formatted labels to label the axis tick marks. They are also useful for chart annotations. Position the objects using any of the coordinate systems. Rotate and justify the text vertically and horizontally.

NumericLabel constructors

```
public static newNumericLabelCoords(transform: PhysicalCoordinates): NumericLabel

public static newNumericLabel(transform: PhysicalCoordinates,
        tfont: ChartFont,
        initialvalue1: number,
        x: number, y: number,
        npostype: number,
        nnumformat: number,
        ndecimal: number): NumericLabel

public static newNumericLabelFormatDecs(
        nnumformat: number,
        ndecimal: number): NumericLabel
```

transform	Places the text in the coordinate system defined by transform.
tfont	A reference to a ChartFont object.
initialvalue	The initial value of the numeric label.
x	Specifies the x-value of the text position
y	Specifies the y-value of the text position
npostype	Specifies the if the position of the text is specified in physical coordinates, normalized coordinates or Canvas device coordinates. Use one of the position constants: DEV_POS, PHYS_POS, NORM_GRAPH_POS, NORM_PLOT_POS.
nnumformat	Specifies the numeric format of the label. Use one of the numeric format constants : DECIMALFORMAT, SCIENTIFICFORMAT, BUSINESSFORMAT, ENGINEERINGFORMAT, PERCENTFORMAT, CURRENCEYFORMAT and EXPONENTFORMAT.
ndecimal	The number of digits to display to the right of the decimal point.
xjust	Specifies the horizontal justification of the text. Use one of the text justification constants: JUSTIFY_MIN, JUSTIFY_CENTER or JUSTIFY_MAX.
yjust	Specifies the vertical justification of the text. Use one of the text justification constants: JUSTIFY_MIN, JUSTIFY_CENTER or JUSTIFY_MAX.
rotation	The rotation (-360 to 360 degrees) of the text in the normal viewing plane.

The **TimeLabel, ElapsedTimeLabel** and **StringLabel** classes are similar, unique properties for each are listed below.

TimeLabel constructors

```
public static newTimeLabelCoords(transform: PhysicalCoordinates): TimeLabel

public static newTimeLabelCoordsFormat(transform: PhysicalCoordinates, date: Date, timeformat:
number): TimeLabel

public static newTimeLabelFormat(timeformat: number): TimeLabel

public static newTimeLabelBase(transform: PhysicalCoordinates,
        tfont: ChartFont,
        date: Date,
        x: number, y: number,
        npostype: number,
        timeformat: number,
        xjust: number, yjust: number,
        rotation: number): TimeLabel
```

date	The calendar value used to initialize the label.
timeformat	The format used to convert the calendar value to a text string. Use one of the calendar format constants, TIMEDATEFORMAT_XXX.

ElapsedTimeLabel constructors

```
[JavaScript]
Overloads Public Sub New( _
   ByVal transform As PhysicalCoordinates, _
   ByVal timespan As ChartTimeSpan, _
   ByVal timeformat As Integer _
)


Visual Basic (Declaration)
Public Sub New ( _
    transform As PhysicalCoordinates, _
    tfont As Font, _
    timespan As ChartTimeSpan, _
    x, _
    y, _
    npostype , _
    timeformat , _
    xjust , _
    yjust , _
    rotation _
)

C#
public ElapsedTimeLabel (
   PhysicalCoordinates transform,
   ChartCalendar date,
   int timeformat
);

public ElapsedTimeLabel(
    PhysicalCoordinates transform,
    Font tfont,
    ChartTimeSpan timespan,
    double x,
    double y,
    int npostype,
    int timeformat,
    int xjust,
    int yjust,
    double rotation
)
```

timespan	The time span value used to initialize the label.

timeformat	The format used to convert the time span value to a text stringUse one of the TIMEDATAFORMAT_ constants: TIMEDATEFORMAT_NONE, TIMEDATEFORMAT_24HMS, TIMEDATEFORMAT_24HM, TIMEDATEFORMAT_MS.

StringLabel constructors

```
public static newStringLabelCoords(transform: PhysicalCoordinates): StringLabel

public static newStringLabel(transform: PhysicalCoordinates,
        tfont: ChartFont,
        tstring: string,
        x: number, y: number,
        npostype: number,
        xjust: number, yjust: number,
        rotation: number): StringLabel

public static newStringLabelCoordsFontPosType(transform: PhysicalCoordinates,
        tfont: ChartFont,
        tstring: string,
        x: number, y: number,
        npostype: number): StringLabel

public static newStringLabelFontString(
        tfont: ChartFont,
        tstring: string): StringLabel
```

tstring	A reference to a string object.

Place text in a time coordinate system (**TimeCoordinates**) by converting the time x-position to milliseconds and using the milliseconds as the x-position value.

StringLabel example (extracted from the example program FinancialExamples.BuildTechnicalAnalysis)

[TypeScript]

```
let TechnicalsFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans
Serif", QCChartTS.ChartFont.REGULAR, 12);
//  General variable to use to position various graph labels
let technicalspostime: Date = new Date(2002, QCChartTS.ChartConstants.NOVEMBER, 15);

//  Create the "Neckline" text label
let necklineLabel: QCChartTS.StringLabel =
QCChartTS.StringLabel.newStringLabelCoordsFontPosType(pTransform1, TechnicalsFont, "Neckline",
QCChartTS.ChartCalendar.getCalendarMsecs(technicalspostime), 9050,
QCChartTS.ChartConstants.PHYS_POS);
necklineLabel.setXJust(QCChartTS.ChartConstants.JUSTIFY_CENTER);
necklineLabel.setYJust(QCChartTS.ChartConstants.JUSTIFY_MIN);
chartVu.addChartObject(necklineLabel);
```

[JavaScript]

```
let TechnicalsFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.REGULAR, 12);
//  General variable to use to position various graph labels
let technicalspostime = new Date(2002, QCChartTS.ChartConstants.NOVEMBER, 15);

let necklineLabel = QCChartTS.StringLabel.newStringLabelCoordsFontPosType(pTransform1,
TechnicalsFont, "Neckline", QCChartTS.ChartCalendar.getCalendarMsecs(technicalspostime), 9050,
QCChartTS.ChartConstants.PHYS_POS);
necklineLabel.setXJust(QCChartTS.ChartConstants.JUSTIFY_CENTER);
necklineLabel.setYJust(QCChartTS.ChartConstants.JUSTIFY_MIN);
chartVu.addChartObject(necklineLabel);
technicalspostime = new Date(2002, QCChartTS.ChartConstants.JULY, 15);
```

NumericLable example (extracted from the example program FinancialExamples.BuildTechnicalAnalysis)

[TypeScript]

```
let labelpostime: Date = new Date(2002, QCChartTS.ChartConstants.OCTOBER, 1);
let RSILabelFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.BOLD, 18);


let overSoldTextLabel: QCChartTS.ChartString =
QCChartTS.ChartString.newChartTextCoordsFontStringPosType(pTransform2, RSILabelFont, "Oversold",
QCChartTS.ChartCalendar.getCalendarMsecs(labelpostime), 0, QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(overSoldTextLabel);

labelpostime = new Date(2002, QCChartTS.ChartConstants.DECEMBER, 1);
let overSoldNumericValue: QCChartTS.NumericLabel =
QCChartTS.NumericLabel.newNumericLabel(pTransform2, RSILabelFont,
    30, QCChartTS.ChartCalendar.getCalendarMsecs(labelpostime),
    0, QCChartTS.ChartConstants.PHYS_POS,  QCChartTS.ChartConstants.DECIMALFORMAT, 1);
chartVu.addChartObject(overSoldNumericValue);

//  Fill the plot from the data line to the 100.0 value, representing the overbought region
let thePlot8: QCChartTS.SimpleLinePlot = QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform2,
RSI70LimitDataset, RSI70LimitAttrib);
thePlot8.setFillBaseValue(100);
chartVu.addChartObject(thePlot8);

labelpostime = new Date(2002, QCChartTS.ChartConstants.OCTOBER, 1);
//  Define the overbought label
let overBoughtTextLabel: QCChartTS.ChartText =
QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform2, RSILabelFont, "Overbought",
QCChartTS.ChartCalendar.getCalendarMsecs(labelpostime), 70, QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(overBoughtTextLabel);

labelpostime = new Date(2002, QCChartTS.ChartConstants.DECEMBER, 1);
```

```
let overBoughtNumericValue: QCChartTS.NumericLabel =
QCChartTS.NumericLabel.newNumericLabel(pTransform2, RSILabelFont,
    70, QCChartTS.ChartCalendar.getCalendarMsecs(labelpostime),
    70, QCChartTS.ChartConstants.PHYS_POS,  QCChartTS.ChartConstants.DECIMALFORMAT, 1);
chartVu.addChartObject(overBoughtNumericValue);
```

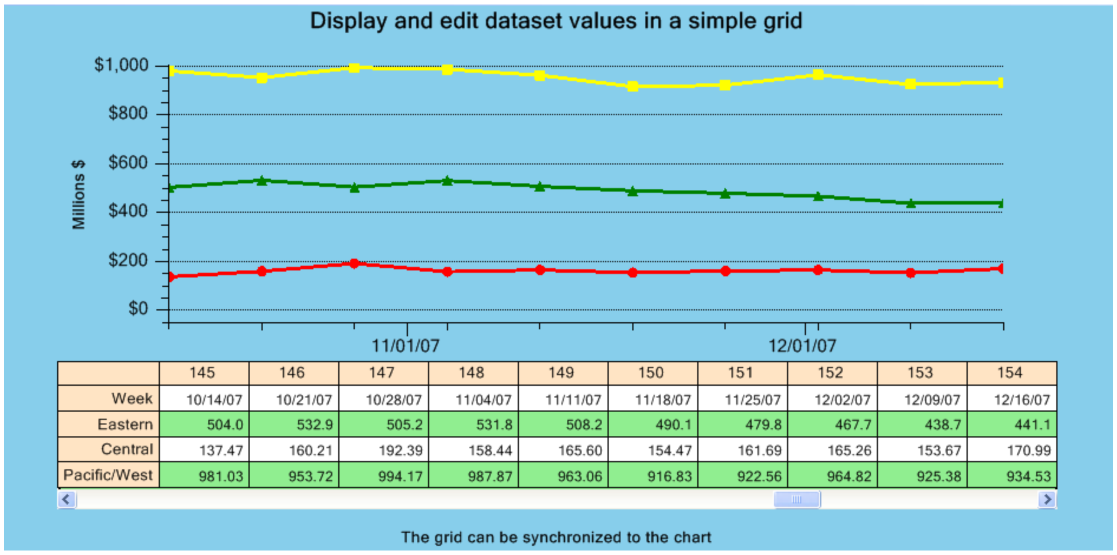
[JavaScript]

```
var labelpostime = new Date(2002, QCChartTS.ChartConstants.OCTOBER, 1);
var RSILabelFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.BOLD, 18);
var overSoldTextLabel = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform2,
RSILabelFont, "Oversold", QCChartTS.ChartCalendar.getCalendarMsecs(labelpostime), 0,
QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(overSoldTextLabel);
labelpostime = new Date(2002, QCChartTS.ChartConstants.DECEMBER, 1);
var overSoldNumericValue = QCChartTS.NumericLabel.newNumericLabel(pTransform2, RSILabelFont, 30,
QCChartTS.ChartCalendar.getCalendarMsecs(labelpostime), 0, QCChartTS.ChartConstants.PHYS_POS,
QCChartTS.ChartConstants.DECIMALFORMAT, 1);
chartVu.addChartObject(overSoldNumericValue);
//  Fill the plot from the data line to the 100.0 value, representing the overbought region
var thePlot8 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform2, RSI70LimitDataset,
RSI70LimitAttrib);
thePlot8.setFillBaseValue(100);
chartVu.addChartObject(thePlot8);
labelpostime = new Date(2002, QCChartTS.ChartConstants.OCTOBER, 1);
//  Define the overbought label
var overBoughtTextLabel = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform2,
RSILabelFont, "Overbought", QCChartTS.ChartCalendar.getCalendarMsecs(labelpostime), 70,
QCChartTS.ChartConstants.PHYS_POS);
chartVu.addChartObject(overBoughtTextLabel);
labelpostime = new Date(2002, QCChartTS.ChartConstants.DECEMBER, 1);
var overBoughtNumericValue = QCChartTS.NumericLabel.newNumericLabel(pTransform2, RSILabelFont,
70, QCChartTS.ChartCalendar.getCalendarMsecs(labelpostime), 70,
QCChartTS.ChartConstants.PHYS_POS, QCChartTS.ChartConstants.DECIMALFORMAT, 1);
chartVu.addChartObject(overBoughtNumericValue);
```

21. Dataset Viewers

DatasetViewer

Charts and data grids are probably the two most popular ways to display numeric data. We have created our own grid class that integrates with our chart dataset classes. The **DatasetViewer** can display simple numeric and time-based datasets (**SimpleDataset**, **TimeSimpleDataset, ElapsedTimeSimpleDataset**) and group numeric and time-based datasets (**GroupDataset**, **TimeGroupsDataset, ElapsedTimeGroupDataset**). When a **DatasetViewer** is added to a chart, it can be printed as part of that chart. Background colors, row and column headers, can be customized. The **DatasetViewer** can be scrolled, updated in real-time, and synchronized to the chart, so that scrolling of the DatasetViewer can scroll the chart.



	145	146	147	148	149	150	151	152	153	154
Week	10/14/07	10/21/07	10/28/07	11/04/07	11/11/07	11/18/07	11/25/07	12/02/07	12/09/07	12/16/07
Eastern	504.0	532.9	505.2	531.8	508.2	490.1	479.8	467.7	438.7	441.1
Central	137.47	160.21	192.39	158.44	165.60	154.47	161.69	165.26	153.67	170.99
Pacific/West	981.03	953.72	994.17	987.87	963.06	916.83	922.56	964.82	925.38	934.53

A DatasetViewer displaying three TimeSimpleDatasets

Class DatasetViewer

ChartView
        |
        +--DatasetViewer

The **DatasetViewer** is a **ChartView** derived object and as such is an independent **UserControl** object. Use it to view one or more datasets in a chart. Since it is usually not possible or practical to display the entire dataset, the **DatasetViewer** windows a rectangular section of the dataset for display. Scroll bars are used to scroll the rows and columns of the dataset. The **DatasetViewer** constructor defines the size, position, source matrix, the number of rows and columns of the **DatasetViewer** grid, and the starting position of the **DatasetViewer** scrollbar.

DatasetViewer constructor

```
public static newDatasetViewer(chartvu: ChartView, transform: PhysicalCoordinates, posrect:
ChartRectangle2D, mat: number[][], rows: number, cols: number, start: number): DatasetViewer

public static newDatasetViewerChartViewCoordsDataset(chartvu: ChartView, transform:
PhysicalCoordinates, posrect: ChartRectangle2D, dataset: ChartDataset, rows: number, cols:
number, start: number): DatasetViewer

public static newDatasetViewerChartViewCoordsDatasetOrient(chartvu: ChartView, transform:
PhysicalCoordinates, posrect: ChartRectangle2D, dataset: ChartDataset, rows: number, cols:
number, start: number, orient: number): DatasetViewer
```

chartvu	The **ChartView** object the **DatasetViewer** is placed in.
transform	The coordinate system the **DatasetViewer** is placed in.
posrect	A positioning rectangle (using normalized chart coordinates) for the dataset viewer, use null if not used.
dataset	A simple, or group, dataset to add to the dataset viewer.
rows	Number of rows to display
cols	Number of columns to display.
start	Starting column of the dataset viewer.

Set unique fonts for the column headers, row headers and grid cells using the ColumnHeaderFont, RowHearderFont and GridCellFont properties.

Turn on the edit feature of the grid cells using the EnableEdit property. Turn on the striped background color of the grid cells using the UseStripedGridBackground property.

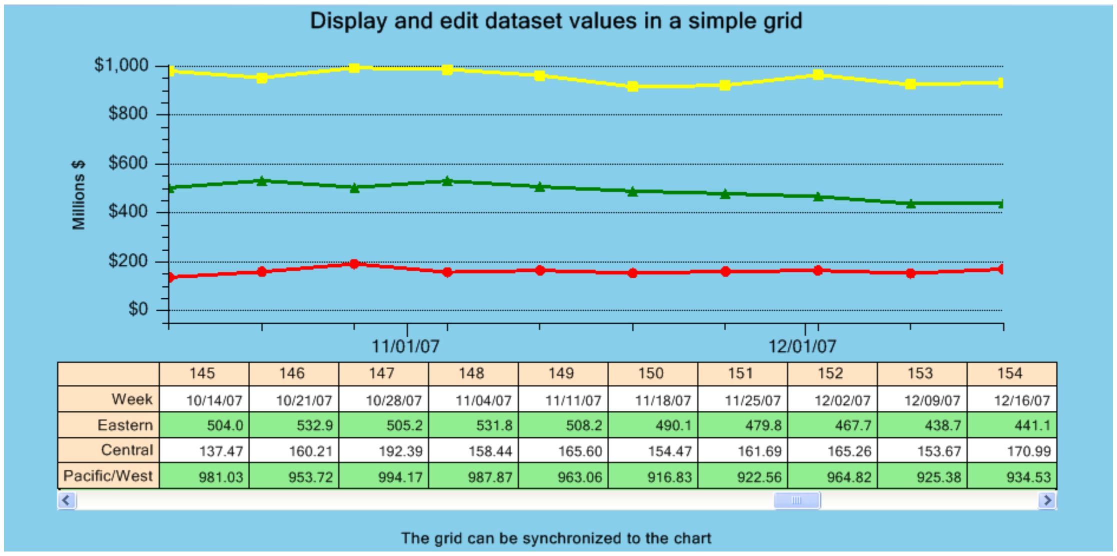
Foreground and background attributes of the column headers, row headers and grid cells can be set using the ColumnHeaderAttribute, RowHeaderAttribute, GridAttribute, and AltGridAttribute properties.

You can add multiple datasets to a **DatasetViewer** using the DatasetViewer.**addDataset** method. When adding additional datasets, it only adds the y-values of the dataset. It is assumed the x-values of the datasets are the same; otherwise, the columns would lose synchronization.

The row header string for the first grid row, the x-values, is picked up from the first dataset's XString property. If that is null, "X-Values" is displayed for numeric x-values, and "Time" for time-based x-values. Subsequent row header strings, for the y-values, are picked up from the main title string of each associated dataset. In the case of group datasets with multiple y-values for each x-value, row header strings are picked up from the datasets GroupStrings property, which stores one string for each group in the dataset.

You can change the default orientation of the **DatasetViewer** by calling a version of the **DatasetViewer** constructor that has an orientation property as the last parameter. See the NewDemosRev2.VerticalDatasetViewerChart for an example.

Simple DatasetViewer example (extracted from the example program NewDemosRev2.BuildSimpleDatasetViewer)


Display and edit dataset values in a simple grid
$1,000
$800
$600
$400
$200
$0
Millions $
11/01/07
12/01/07
The grid can be synchronized to the chart

	145	146	147	148	149	150	151	152	153	154
Week	10/14/07	10/21/07	10/28/07	11/04/07	11/11/07	11/18/07	11/25/07	12/02/07	12/09/07	12/16/07
Eastern	504.0	532.9	505.2	531.8	508.2	490.1	479.8	467.7	438.7	441.1
Central	137.47	160.21	192.39	158.44	165.60	154.47	161.69	165.26	153.67	170.99
Pacific/West	981.03	953.72	994.17	987.87	963.06	916.83	922.56	964.82	925.38	934.53

A DatasetViewer displaying three TimeSimpleDatasets

[TypeScript]

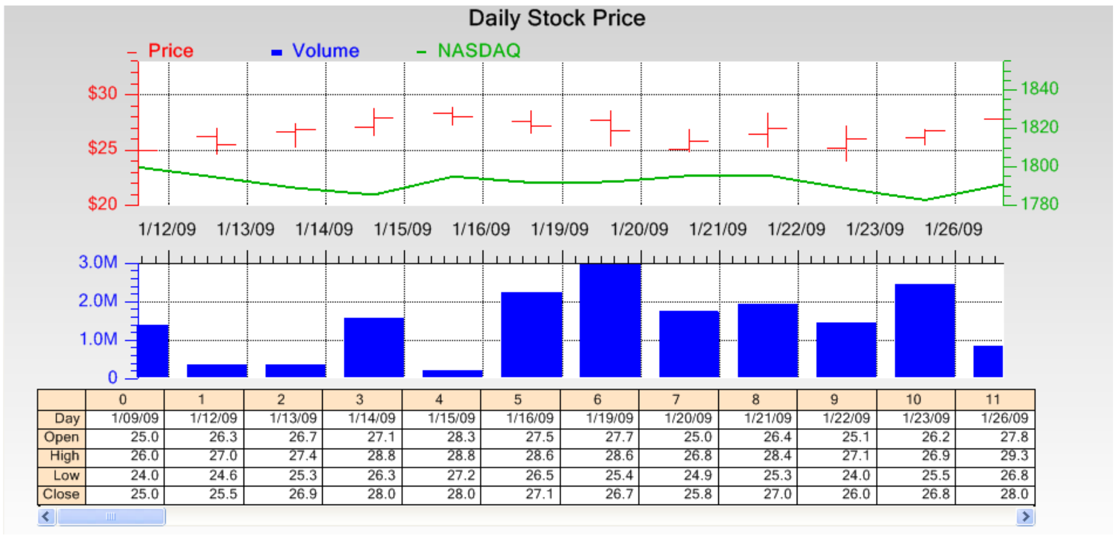
```
let posrect: QCChartTS.ChartRectangle2D = QCChartTS.ChartRectangle2D.newChartRectangle2D(0.05,
0.67, 0.9, 0.26);
let startindex: number = initialstartindex;
let rows: number = 3;
let columns: number = 8;
let datasetViewer1: QCChartTS.DatasetViewer =
QCChartTS.DatasetViewer.newDatasetViewerChartViewCoordsDataset(chartVu, pTransform1, posrect,
Dataset1, rows, columns, startindex);
datasetViewer1.addDataset(Dataset2);
datasetViewer1.addDataset(Dataset3);
datasetViewer1.enableEdit(true);
if (datasetViewer1.DatasetTable)
datasetViewer1.DatasetTable.TableBackgroundMode =
QCChartTS.GeneralizedTableDisplay.TABLE_SINGLE_COLOR_BACKGROUND_GRIDCELL;
datasetViewer1.UseStripedGridBackground = true;
datasetViewer1.RowHeaderFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.REGULAR, 12);
datasetViewer1.ColumnHeaderFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.REGULAR, 12);
datasetViewer1.GridCellFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.REGULAR, 12);
```

```
datasetViewer1.SyncChart = true;

chartVu.addChartObject(datasetViewer1);
```

[JavaScript]

```
let posrect = QCChartTS.ChartRectangle2D.newChartRectangle2D(0.05, 0.67, 0.9, 0.26);
let startindex = initialstartindex;
let rows = 3;
let columns = 8;
let datasetViewer1 = QCChartTS.DatasetViewer.newDatasetViewerChartViewCoordsDataset(chartVu,
pTransform1, posrect, Dataset1, rows, columns, startindex);
datasetViewer1.addDataset(Dataset2);
datasetViewer1.addDataset(Dataset3);
datasetViewer1.enableEdit(true);
if (datasetViewer1.DatasetTable)
datasetViewer1.DatasetTable.TableBackgroundMode =
QCChartTS.GeneralizedTableDisplay.TABLE_SINGLE_COLOR_BACKGROUND_GRIDCELL;
datasetViewer1.UseStripedGridBackground = true;
datasetViewer1.RowHeaderFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.REGULAR, 12);
datasetViewer1.ColumnHeaderFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.REGULAR, 12);
datasetViewer1.GridCellFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.REGULAR, 12);
datasetViewer1.SyncChart = true;


chartVu.addChartObject(datasetViewer1);
```

Group DatasetViewer example (extracted from the example program NewDemosRev2.BuildGroupDatasetViewer)



	0	1	2	3	4	5	6	7	8	9	10	11
Day	1/09/09	1/12/09	1/13/09	1/14/09	1/15/09	1/16/09	1/19/09	1/20/09	1/21/09	1/22/09	1/23/09	1/26/09
Open	25.0	26.3	26.7	27.1	28.3	27.5	27.7	25.0	26.4	25.1	26.2	27.8
High	26.0	27.0	27.4	28.8	28.8	28.6	28.6	26.8	28.4	27.1	26.9	29.3
Low	24.0	24.6	25.3	26.3	27.2	26.5	25.4	24.9	25.3	24.0	25.5	26.8
Close	25.0	25.5	26.9	28.0	28.0	27.1	26.7	25.8	27.0	26.0	26.8	28.0

A **DatasetViewer** displaying a **TimeGroupDataset** display open-high-low-close data

[TypeScript]

```
let posrect: QCChartTS.ChartRectangle2D =
QCChartTS.ChartRectangle2D.newChartRectangle2DFrame(0.03, 0.67, 0.9, 0.24);
QCChartTS.DatasetViewer.DefaultFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.REGULAR, 11);
let datasetViewer1: QCChartTS.DatasetViewer =
QCChartTS.DatasetViewer.newDatasetViewerChartViewCoordsDataset(chartVu, pTransform1, posrect,
Dataset1, 5, 12, 0);

datasetViewer1.enableEdit(true);
datasetViewer1.SyncTableRange = true;
// datasetViewer1.DatasetTable.TableBackgroundMode =
QCChartTS.GeneralizedTableDisplay.TABLE_SINGLE_COLOR_BACKGROUND_GRIDCELL;
datasetViewer1.SyncChart = true;
datasetViewer1.setArrayFormat(0, QCChartTS.ChartConstants.TIMEDATEFORMAT_MDY);
datasetViewer1.TransformList.push(pTransform2);
datasetViewer1.TransformList.push(pTransform3);

chartVu.addChartObject(datasetViewer1);
```

[JavaScript]

```
let posrect = QCChartTS.ChartRectangle2D.newChartRectangle2DFrame(0.03, 0.67, 0.9, 0.24);
QCChartTS.DatasetViewer.DefaultFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.REGULAR, 11);
let datasetViewer1 = QCChartTS.DatasetViewer.newDatasetViewerChartViewCoordsDataset(chartVu,
pTransform1, posrect, Dataset1, 5, 12, 0);

datasetViewer1.enableEdit(true);
datasetViewer1.SyncTableRange = true;
// datasetViewer1.DatasetTable.TableBackgroundMode =
QCChartTS.GeneralizedTableDisplay.TABLE_SINGLE_COLOR_BACKGROUND_GRIDCELL;
datasetViewer1.SyncChart = true;
datasetViewer1.setArrayFormat(0, QCChartTS.ChartConstants.TIMEDATEFORMAT_MDY);
datasetViewer1.TransformList.push(pTransform2);
datasetViewer1.TransformList.push(pTransform3);

chartVu.addChartObject(datasetViewer1);
```

22. Adding Lines, Shapes, Images and Arrows to a Chart

ChartShape
Arrow
ChartImage

It is not possible to take into account every possible graphical object that a programmer wants to add to a graph. Specialized applications require specialized objects. Rather than create a large group of classes that duplicate the functions of the **Arc2D**, **ChartRectangle2D** and other classes, a generalized class has been created, **ChartShape**, that can place and display in a chart any object that can be expressed as a **GPath**.

The **Arrow** defines an arrow shape useable with the **ChartShape** class. The class creates a base arrow with a custom arrowhead and shaft size. Scale, rotate and position the arrow in a chart.

The **ChartImage** class places a **HTMLImageElement** object anywhere in a chart. It can be a small element of the chart, inside or outside of the plot area, or it can be sized to fill the plot area or graph area and used as a background object.

Generic Shape Class

Class ChartShape

```
GraphObj
   |
   +--ChartShape
```

The **ChartShape** class places arbitrary **GPath** objects in a chart. If the shape includes absolute positioning information, use (0,0) as the xy position parameters of the shape. If the shape coordinates are relative coordinates with the object centered on (0,0), place the shape at the position you want using the xy position parameters. The xy position parameters are the rotation origin of shape.

ChartShape constructor

```
public static newChartShape(transform: PhysicalCoordinates,
        ashape: GPath, shapecoordstype: number,
        x: number, y: number, npositiontype: number,
        rotation: number): ChartShape
```

transform	The shape object is placed in the coordinate system defined by transform.
ashape	A reference to a **GPath** object.
shapecoordstype	Specifies if the coordinate system defining the shape is specified in physical coordinates, normalized coordinates or Canvas device coordinates. Use one of the position constants: DEV_POS, PHYS_POS, NORM_GRAPH_POS, NORM_PLOT_POS.
x	Specifies the x-value of the shape position.
y	Specifies the y-value of the shape position.
npostype	Specifies the if the position of the shape is specified in physical coordinates, normalized coordinates or Canvas device coordinates. Use one of the position constants: DEV_POS, PHYS_POS, NORM_GRAPH_POS, NORM_PLOT_POS.
rotation	The rotation, in degrees, of the shape in the normal viewing plane. The rotation will take place about the objects (0.0, 0.0) coordinate. If the object is not defined with a center of (0.0, 0.0) it may be rotated out of the current viewing plane.

ChartShape example (extracted from the example program MultiLinePlots,BuildMultiLines)

[TypeScript]

```
let alphaColor: number = QCChartTS.ChartColor.fromArgb(127, 170, 100, 50);
let attrib2: QCChartTS.ChartAttribute = QCChartTS.ChartAttribute.newChartAttribute4(alphaColor,
1, QCChartTS.ChartConstants.LS_SOLID, alphaColor);
attrib2.setFillFlag(true);
let linearRegionRect: QCChartTS.ChartRectangle2D =
QCChartTS.ChartRectangle2D.newChartRectangle2DFrame(0.1, 0.1, 1.5, 50);
let rectpath: QCChartTS.GPath = new QCChartTS.GPath();
rectpath.addRectangle(linearRegionRect.getRectangle());
let linearRegionShape: QCChartTS.ChartShape = QCChartTS.ChartShape.newChartShape(pTransform1,
rectpath, QCChartTS.ChartConstants.PHYS_POS, 0, 0, QCChartTS.ChartConstants.PHYS_POS, 0);
linearRegionShape.setChartObjAttributes(attrib2);
chartVu.addChartObject(linearRegionShape);
```

[JavaScript]

```
let alphaColor = QCChartTS.ChartColor.fromArgb(127, 170, 100, 50);
let attrib2 = QCChartTS.ChartAttribute.newChartAttribute4(alphaColor, 1,
QCChartTS.ChartConstants.LS_SOLID, alphaColor);
attrib2.setFillFlag(true);
let linearRegionRect = QCChartTS.ChartRectangle2D.newChartRectangle2DFrame(0.1, 0.1, 1.5, 50);
let rectpath = new QCChartTS.GPath();
rectpath.addRectangle(linearRegionRect.getRectangle());
let linearRegionShape = QCChartTS.ChartShape.newChartShape(pTransform1, rectpath,
QCChartTS.ChartConstants.PHYS_POS, 0, 0, QCChartTS.ChartConstants.PHYS_POS, 0);
linearRegionShape.setChartObjAttributes(attrib2);
chartVu.addChartObject(linearRegionShape);)
```

ChartShape example (extracted from the example program ScatterPlots.BuildLabeledDatapoints)

[TypeScript]

```
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);
```

[JavaScript]

```
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);
```

Chart Image Class

Class ChartImage

GraphObj
```
        |
        +-- ChartImage
```

The **ChartImage** class will place a **HTMLImageElement** object anywhere in a chart. It can be a small element of the chart, inside or outside of the plot area or it can be sized to fill the plot area or graph area and used as a background object. Which image formats are supported is browser dependent, but it looks like modern browsers support at least JPEG, PNG, BMP, SVG, and ICO. The HTMLImageElement used to access the image file can be loaded from a file located at the root (or subfolder) of your website, but not outside of it, because that would be a security violation.

ChartImage constructor

```
public static newChartImage(transform: PhysicalCoordinates,
        aimage: HTMLImageElement | null, x: number, y: number, npostype: number,
        rotation: number): ChartImage
```

transform	The coordinate system for the new **ChartImage** object.
aimage	A reference to the **Image** object that is to be placed in the chart.
x	The x-value for the position of the image in the chart.
y	The y-value for the position of the image in the chart.

npostype	Specifies whether the x- and y-position values are specified in normalized coordinates, or physical coordinates. Use one of the position constants: NORM_POS, PHYS_POS.
rotation	The rotation of the image specified in degrees.

ChartImage example (extracted from the example program ImageCharts.BuildImageBackground)

[TypeScript]

```
let filename: string = "Images/ChartClouds.jpg";
let aImage: HTMLImageElement | null = document.createElement('img');
aImage.src = filename;
aImage.onload = function () {
  if ((aImage != null)) {
  let chartImage: QCChartTS.ChartImage = QCChartTS.ChartImage.newChartImage(pTransform1, aImage,
0, 0, QCChartTS.ChartConstants.NORM_GRAPH_POS, 0);
  chartImage.setSizeMode(QCChartTS.ChartConstants.COORD_SIZE);
  chartImage.setImageSize(QCChartTS.ChartDimension.newChartDimension(1, 1));
  chartImage.ZOrder = thePlot1.ZOrder - 1;
  chartVu.addChartObject(chartImage);
  chartVu.updateDraw();
    }
};
```

[JavaScript]

```
let filename = "Images/ChartClouds.jpg";
let aImage = document.createElement('img');
aImage.src = filename;
aImage.onload = function () {
    if ((aImage != null)) {
       let chartImage = QCChartTS.ChartImage.newChartImage(pTransform1, aImage, 0, 0,
QCChartTS.ChartConstants.NORM_GRAPH_POS, 0);
       chartImage.setSizeMode(QCChartTS.ChartConstants.COORD_SIZE);
       chartImage.setImageSize(QCChartTS.ChartDimension.newChartDimension(1, 1));
       chartImage.ZOrder = thePlot1.ZOrder - 1;
       chartVu.addChartObject(chartImage);///
       chartVu.updateDraw();
    }
};
```

Generic Arrow Class

Class Arrow

```
ChartObj
       |
       +-- Arrow
```

The **Arrow** defines an arrow shape useable with the **ChartShape** class. The class creates a base arrow with a custom arrowhead and shaft size. Scale, rotate and position the arrow in a chart. The arrow is defined using device coordinates.

Arrow constructor

```
[public static newArrow(arrowshafthalfwidth: number, arrayshaftlength: number,
arrowheadhalfwidth: number, arrowheadlength: number): Arrow
```

arrowshafthalfwidth	Sets the half-width of the arrow shaft. (default 1)
arrayshaftlength	Sets the length of the arrow shaft. (default 7)
arrowheadhalfwidth	Sets the half-width of the arrow head. (default 2)
arrowheadlength	Sets the length of the arrow head. (default 3)

The default arrow has a length of about 10 pixels and a width of 4 pixels at the head. The size of the various parts can be set to whatever values you want to create an arrow of with an aspect ratio appropriate to your application. You can scale the arrow by setting the **ArrowScaleFactor** property. Get a **GPath** object defining the arrow shape by calling the **getArrowShape** method.

Arrow example (extracted from the example program MultiLinePlots.BuildMultiLines)

[TypeScript]

```
let regionArrow: QCChartTS.Arrow = QCChartTS.Arrow.newArrow(1, 40, 6, 15);
let arrowAttrib: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.BLACK);
arrowAttrib.setFillFlag(true);
let arrowShape: QCChartTS.ChartShape = QCChartTS.ChartShape.newChartShape(pTransform1,
regionArrow.getArrowShape(), QCChartTS.ChartConstants.DEV_POS, 1.5, 40,
QCChartTS.ChartConstants.PHYS_POS, 195);
arrowShape.setChartObjAttributes(arrowAttrib);
chartVu.addChartObject(arrowShape);
```

[JavaScript]

```
let regionArrow = QCChartTS.Arrow.newArrow(1, 40, 6, 15);
let arrowAttrib = QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.BLACK, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.BLACK);
```

```
arrowAttrib.setFillFlag(true);
let arrowShape = QCChartTS.ChartShape.newChartShape(pTransform1, regionArrow.getArrowShape(),
QCChartTS.ChartConstants.DEV_POS, 1.5, 40, QCChartTS.ChartConstants.PHYS_POS, 195);
arrowShape.setChartObjAttributes(arrowAttrib);
chartVu.addChartObject(arrowShape);
```

23. Printing and Copying the Canvas Image

The **QCChart2D for JavaScript/TypeScript** software does not have any internal printing options. You need to use the browsers built-in printing options. See the Setup printer options for the browser under consideration.

Copying the Canvas Image

You can retrieve an image representation of a chart as a URL string. This option uses the HTMLCanvasElement.toImageURL feature of the HTML5 canvas.

Class BufferedImage

```
ChartObj
         |
         +-- BufferedImage
```

The **BufferedImage** can return a URL string pointing to an image of the underlying ChartView canvas. That string can then be used as the data source for an HTML img tag to display the chart image.

It would be useful to be able to copy a chart image to the clipboard pro grammatically. While this has been added to the HTML5 specification, it is not yet been widely implemented. So our software won't do that. You should be able to right click on the image in the browser and copy it to the clipboard that way.

BufferedImage constructor

```
public static newBufferedImage(cv: ChartView): BufferedImage
```

cv	The **ChartView** object that is the source for the chart image.

Use the toImageURL function to retrieve a URL to the ChartView image. The image must have already been rendered in the ChartView before the URL is valid.

```
 public toImageURL(): string
```

Example

In the parent HTML

```
<img id="imageElement1" src="" alt="chart image" width="800" height="600">
```

The image ID can be any string that you want. You refer to the ID string in order to get the underlying HTMLImageElement object using getElementById. Then, somewhere in your JavaScript or Typescript code assign the value of the image data source to the string returned by **BufferedImage.toImageURL** function.

[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();
```

24. Using QCChart2D for JavaScript/TypeScript to Create Web Applications

JavaScript and TypeScript are folder oriented. When you access a JavaScript file from your host HTML page, you need to specify the *.js file location relative to the current HTML file location. If that file imports anything, such as our QCChart2DTS library file (qcchart2dts.js), it too must be referenced using the appropriate relative folder location. It is assumed that all working folders are within the root directory of the web server. In our examples, the host HTML page references the chart building javascript file using script like:

```
import { HelloChart} from './HelloChart.js';
```

The "./" in the folder specification above represents the current folder that contains the HTML file you are working with.

Even though the TypescriptExamplesQCChart2D folder contains the TypeScript example programs, you still end up referencing a .js file in your host HTML file. The current generation of browsers (circa 2020) cannot import TypeScript (.ts) file directly. Instead you must import the corresponding JavaScript that was made (transpiled) from the source TypeScript file. In order to simplify the references, when the TypeScript source files are transpiled, the corresponding *.js file is placed in the same folder, right next to the original *.ts file. This is specified using the tsconfig.json file, which is sort of a project file for TypeScript applications.

Note: In order to test the HTML files you create, you must use a real, or command line, server such as the npm "http-server" to serve up the html files you create. Because of the JavaScript and HTML5 features used in the software, you **cannot** just reference the HTML files using a browser and point to the physical file location. Instead you must start a local server in one of our folders and access the HTML files from there, using a http:// address like:

http://127.0.0.1:8080/JavascriptExamplesQCChart2D/HelloChart/HelloChart.html

where http://127.0.0.1:8080/ represents the local server. You can usually replace 127.0.0.1 with *localhost*.

http://localhost:8080/JavascriptExamplesQCChart2D/HelloChart/HelloChart.html

Start the npm "http-server" server using the batch file: startlocalserver.bat
The main folder containing both the JavaScript and TypeScript examples is Quinn-Curtis/JSTS. In that folder is a batch file (**startlocalserver.bat**) which will start the npm "http-server" local server, setting the directory it is located in as the root directory of the test server. All the batch file contains is:

```
http-server ./
```

where http-server starts the npm server, and "./" specifies that the current directory is to be the root folder of the server. In order to start the npm "http-server" that way it is assumed that you have installed the npm "http-server" command line server globally according to the instructions here: https://www.npmjs.com/package/http-server You will need a relatively current version of npm to properly install the http-server package.

Linux (Ubuntu) users: If you plan to use Linux (Ubuntu), see the section at the end of this chapter for issues unique to that development environment.

Visual Studio Code

Visual Studio Code is available for free from a Microsoft website: https://code.visualstudio.com . It is available for x64-based systems only. Once installed, you will also need to install the TypeScript compiler, if you plan to use TypeScript. JavaScript does not require a compiler. You can install the TypeScript compiler using the directions here: https://code.visualstudio.com/docs/typescript/typescript-compiling . It basically uses a command line from a Command Prompt window which looks like:

```
npm install -g typescript
```

You will need a relatively current version of npm to do this. We are using Version 5.6.0. You can check your npm version by entering:

```
npm -v
```

from the command prompt. If you need to install npm, or upgrade it, you will find resources on the web which tell you how to do that.

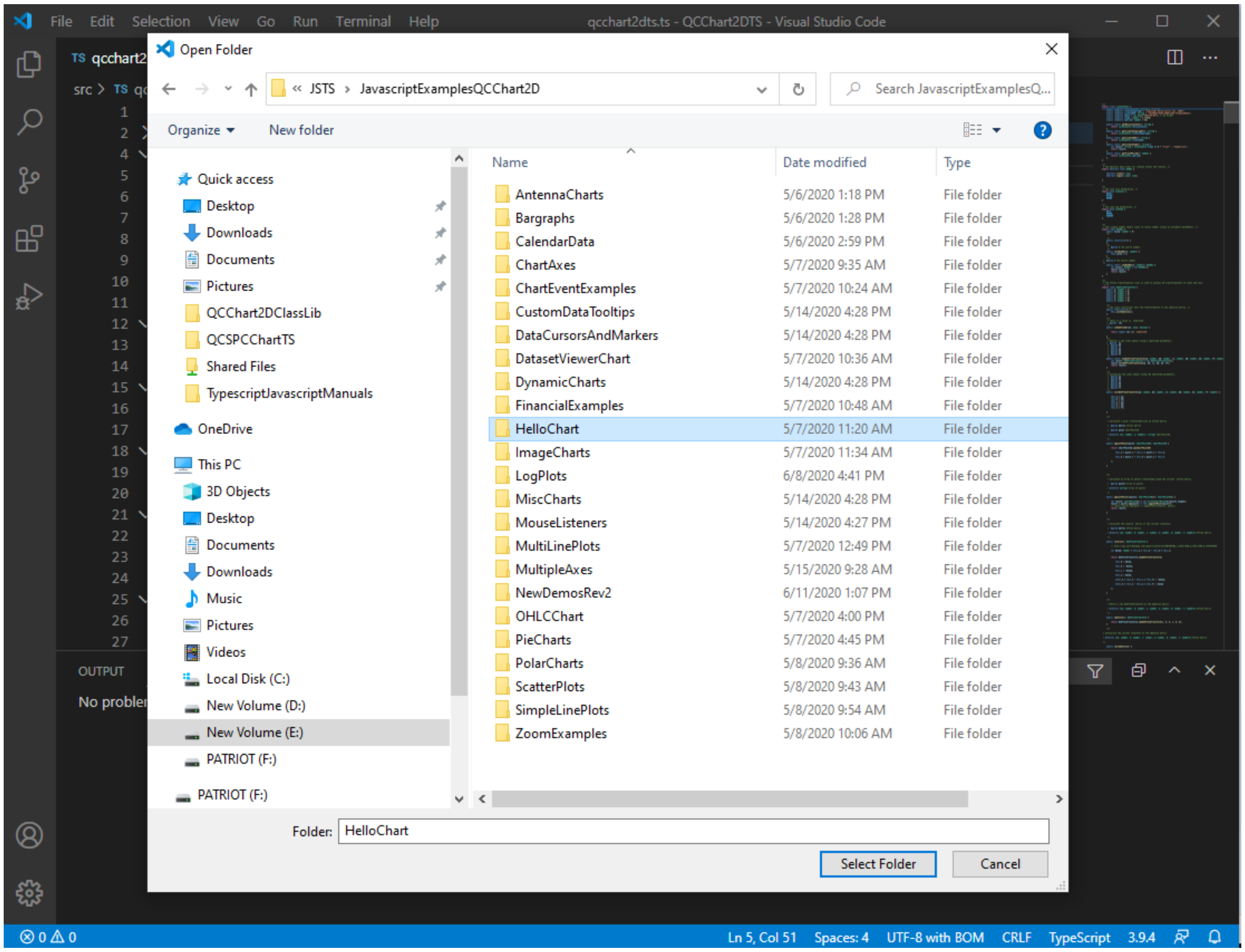
Using JavaScript Only

While we recommend that you use a TypeScript intermediary to interact with the QCChart2DTS library, you can also call it directly from JavaScript. While QCChart2DTS is written in TypeScript, the file that you reference inside your JavaScript program is the transpiled JavaScript version of the library. All of the class, property and function names inside the library are the same, regardless of whether you using TypeScript or JavaScript.

Start off by creating a folder unique to your intended code, under our JSTS/JavacriptExamplesQCChart2D/ folder. *We will use the folder name HelloChart. Since we include the HelloChart folder as one of our examples, if you plan on following this tutorial exactly, just rename the existing HelloChart folder something else, HelloChart2 for example.* In its simplest form, that folder will contain just two files: your test HTML page (HelloChart .html) and the JavaScript source page

(HelloChart .js). In this example, the HelloChart.html page can just be a bare-bones HTML page, the same as our JavascriptExamplesQCChart2D/HelloChart .html. The only difference is in the embedded filename used to access the HelloChart.js JavaScript file.

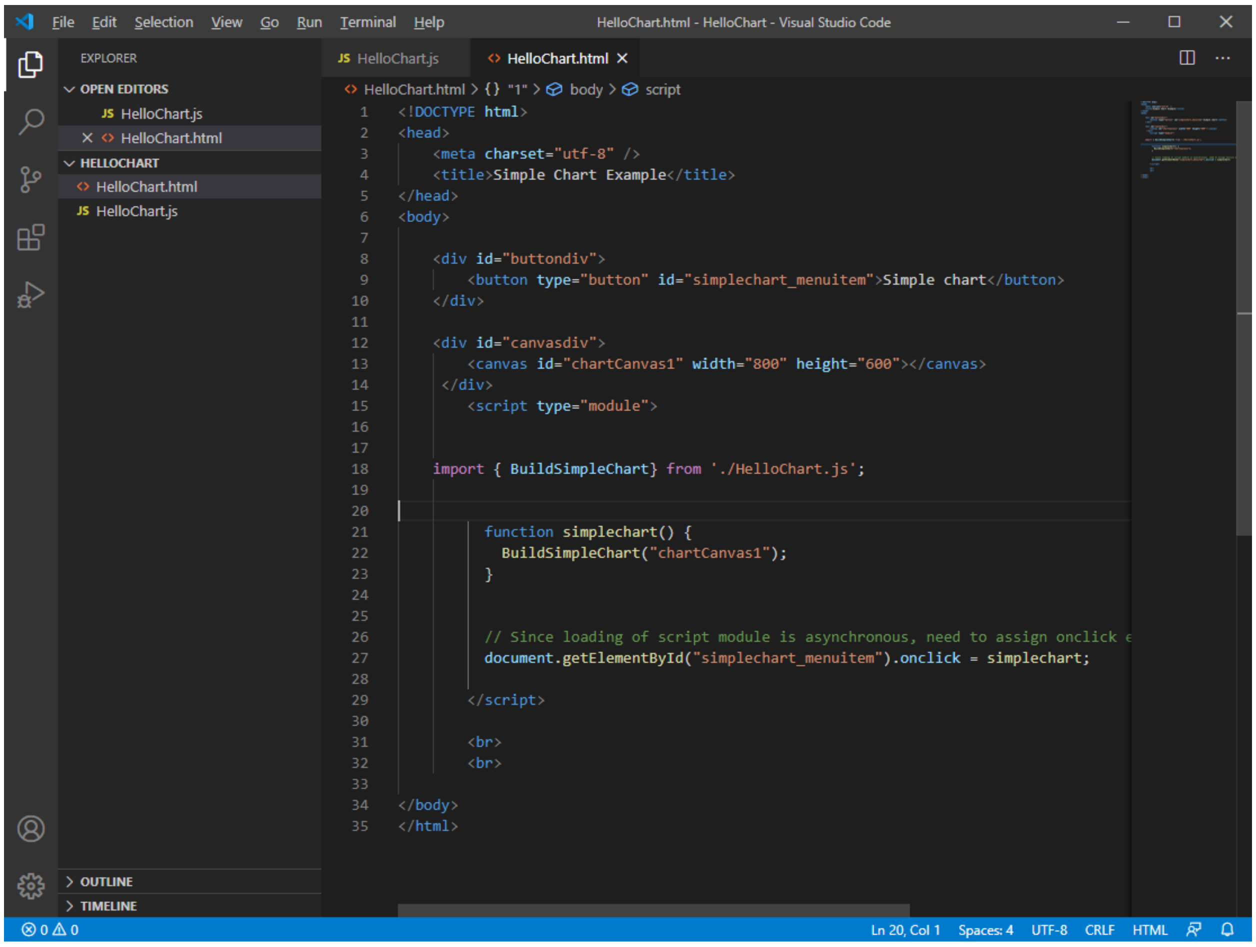
From your JavaScript Development Environment (Visual Studio Code in our case), open the HelloChart folder using the IDE File | Open Folder option, and select the HelloChart folder you created.



Name	Date modified	Type
AntennaCharts	5/6/2020 1:18 PM	File folder
Bargraphs	5/6/2020 1:28 PM	File folder
CalendarData	5/6/2020 2:59 PM	File folder
ChartAxes	5/7/2020 9:35 AM	File folder
ChartEventExamples	5/7/2020 10:24 AM	File folder
CustomDataTooltips	5/14/2020 4:28 PM	File folder
DataCursorsAndMarkers	5/14/2020 4:28 PM	File folder
DatasetViewerChart	5/7/2020 10:36 AM	File folder
DynamicCharts	5/14/2020 4:28 PM	File folder
FinancialExamples	5/7/2020 10:48 AM	File folder
HelloChart	5/7/2020 11:20 AM	File folder
ImageCharts	5/7/2020 11:34 AM	File folder
LogPlots	6/8/2020 4:41 PM	File folder
MiscCharts	5/14/2020 4:28 PM	File folder
MouseListeners	5/14/2020 4:27 PM	File folder
MultiLinePlots	5/7/2020 12:49 PM	File folder
MultipleAxes	5/15/2020 9:28 AM	File folder
NewDemosRev2	6/11/2020 1:07 PM	File folder
OHLCChart	5/7/2020 4:00 PM	File folder
PieCharts	5/7/2020 4:45 PM	File folder
PolarCharts	5/8/2020 9:36 AM	File folder
ScatterPlots	5/8/2020 9:43 AM	File folder
SimpleLinePlots	5/8/2020 9:54 AM	File folder
ZoomExamples	5/8/2020 10:06 AM	File folder

Folder: HelloChart

Next, create two new files in the HelloChart directory: HelloChart.html and HelloChart.js, using the File | New File options of the IDE. In VS Code they are initially named Untitled-1 and Untitled-2, but when you go to File | Save, or File | Save As you will be prompted for their proper names and then you can rename them HelloChart.html and HelloChart.js . So you should end up with a folder named HelloChart containing two blank files, HelloChart.html and HelloChart.js.


File Edit Selection View Go Run Terminal Help
HelloChart.html - HelloChart - Visual Studio Code
EXPLORER
OPEN EDITORS
HelloChart.js
HelloChart.html
HELLOCHART
HelloChart.html
HelloChart.js
OUTLINE
TIMELINE
HelloChart.html > {} "1" > body > script
Ln 20, Col 1 Spaces: 4 UTF-8 CRLF HTML

```
<!DOCTYPE html>
<head>
    <meta charset="utf-8" />
    <title>Simple Chart Example</title>
</head>
<body>

    <div id="buttondiv">
        <button type="button" id="simplechart_menuitem">Simple chart</button>
    </div>

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


    import { BuildSimpleChart} from './HelloChart.js';


          function simplechart() {
            BuildSimpleChart("chartCanvas1");
          }


          // Since loading of script module is asynchronous, need to assign onclick e
          document.getElementById("simplechart_menuitem").onclick = simplechart;

        </script>

        <br>
        <br>

</body>
</html>
```

Copy the code below into the HelloChart.html file.

```
<!DOCTYPE html>
<head>
    <meta charset="utf-8" />
    <title>Simple Chart Example</title>
</head>
<body>

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

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


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

     var hellochart = new HelloChart();


          function simplechart() {
```

```
          hellochart.BuildSimpleChart("chartCanvas1");
        }


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

      </script>

      <br>
      <br>

</body>
</html>
```

Note that the import statement looks to the current directory for the HelloChart.js file.

```
    import { HelloChart} from './HelloChart.js';
```

When the button (id = simplechart_menuitem) is pressed, it triggers a call to the BuildSimpleChart function imported from the HelloChart.js file. Note that the event handler (simplechart) for the button:

```
   document.getElementById("simplechart_menuitem").onclick = simplechart;
```

is installed inside the JavaScript block of code. Since JavaScript code marked type=”module” is loaded asynchronously, you **cannot** assign the event handler when the button is created in the HTML. That is because the JavaScript inside the <script type="module"> has not yet been loaded when the button is defined, and therefore the *simplechart* function contained within cannot be referenced. If you attempt to do so it will be assigned a null (or undefined) value. But if you do the event handler assignment inside the <script type="module"> code block, after the *simplechart* function has been defined, it will work.

Save the HelloChart.html file and go to the HelloChart.js file. You can copy the code below into that file.

```
import * as QCChartTS from '../../QCChart2DTS/qcchart2dts.js';

export async function BuildSimpleChart(canvasid) {
    let htmlcanvas = document.getElementById(canvasid);
    let chartVu = QCChartTS.ChartView.newChartView(htmlcanvas);
    if (!chartVu)
        return;
    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()));
    }
    y2[94] = 10;
    y3[0] = 95;
    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);
    let xAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.X_AXIS);
    chartVu.addChartObject(xAxis);
    let yAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.Y_AXIS);
    chartVu.addChartObject(yAxis);
    let xAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(xAxis);
    xAxisLab.setTextFont(theFont);
    chartVu.addChartObject(xAxisLab);
    let yAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(yAxis);
    yAxisLab.setTextFont(theFont);
    chartVu.addChartObject(yAxisLab);
    let titleFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
    let yaxistitle = QCChartTS.AxisTitle.newAxisTitle(yAxis, titleFont, "Measurable work
output");
    chartVu.addChartObject(yaxistitle);
    let xaxistitle = QCChartTS.AxisTitle.newAxisTitle(xAxis, titleFont, "# MBAs/1000 employees");
    chartVu.addChartObject(xaxistitle);
    let xgrid = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.X_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
    chartVu.addChartObject(xgrid);
    let ygrid = QCChartTS.Grid.newGrid(xAxis, 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 EnronLabel = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1,
theFont, "Eronix", x1[94], y2[94], QCChartTS.ChartConstants.PHYS_POS);
    EnronLabel.setXJust(QCChartTS.ChartConstants.JUSTIFY_MAX);
    EnronLabel.setYJust(QCChartTS.ChartConstants.JUSTIFY_MAX);
    EnronLabel.setTextNudge(-4, 4);
    chartVu.addChartObject(EnronLabel);
    let IGG = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1, theFont,
"Quinn-Curtis", x1[0], y3[0], QCChartTS.ChartConstants.PHYS_POS);
    IGG.setXJust(QCChartTS.ChartConstants.JUSTIFY_MIN);
    IGG.setYJust(QCChartTS.ChartConstants.JUSTIFY_MAX);
    IGG.setTextNudge(4, 4);
    chartVu.addChartObject(IGG);
    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 toolTipFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.REGULAR,
12);
    let datatooltip = QCChartTS.DataToolTip.newDataToolTip(chartVu);
    let xValueTemplate =
QCChartTS.NumericLabel.newNumericLabelFormatDecs(QCChartTS.ChartConstants.DECIMALFORMAT, 0);
    let yValueTemplate =
QCChartTS.NumericLabel.newNumericLabelFormatDecs(QCChartTS.ChartConstants.DECIMALFORMAT, 1);
    let textTemplate = QCChartTS.ChartText.newChartTextFontString(toolTipFont, "");
    textTemplate.setTextBgColor(QCChartTS.ChartColor.fromRgb(255, 255, 204));
    textTemplate.setTextBgMode(true);
    let toolTipSymbol = QCChartTS.ChartSymbol.newChartSymbol(pTransform1,
QCChartTS.ChartConstants.SQUARE,
QCChartTS.ChartAttribute.newChartAttributeColor(QCChartTS.ChartColor.GREEN));
    toolTipSymbol.setSymbolSize(10);
    datatooltip.setTextTemplate(textTemplate);
    datatooltip.setXValueTemplate(xValueTemplate);
    datatooltip.setYValueTemplate(yValueTemplate);
    datatooltip.setDataToolTipFormat(QCChartTS.ChartConstants.DATA_TOOLTIP_XY_ONELINE);
    datatooltip.setToolTipSymbol(toolTipSymbol);
    datatooltip.setEnable(true);
    //   chartVu.setCurrentMouseListener(datatooltip);
    chartVu.updateDraw();
}
```

Note that the QCChart2DTS library is being imported from its location in the file system, relative to the HelloChart.js file.

```
import * as  QCChartTS from '../../QCChart2DTS/qcchart2dts.js';
```

The "import * as QCChartTS" in the import statement assigns all classes, functions, properties and constants in the software to the namespace QCChartTS, so you must preface any reference to the formal name of any class, static method within a class, or static constants, found within the QCChart2D library, by explicitly referencing that namespace.

```
    let htmlcanvas = document.getElementById(canvasid);
    let chartVu = QCChartTS.ChartView.newChartView(htmlcanvas);
    if (!chartVu)
        return;
    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()));
    }
    y2[94] = 10;
    y3[0] = 95;
    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);
```

The HelloChart.js file consists of just one function, BuildSimpleChart.

In broad terms, first you create an instance of a ChartView object using a static constructor.:

```
export async function BuildSimpleChart(canvasid) {

    let htmlcanvas = document.getElementById(canvasid);
    let chartVu = QCChartTS.ChartView.newChartView(htmlcanvas);
    if (!chartVu)
        return;
```

passing in the required canvas object. Don't leave anything out because JavaScript will only tell you that you are missing a require parameter at runtime when web page crashes, and then only if you are looking at the web page using the debugger.

Next you get, or create your data and assign it to one or more of our Dataset objects.

```
    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()));
    }
    y2[94] = 10;
    y3[0] = 95;
    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);
```

Next, you define the coordinate system you want to display the data in. You can auto-scale the coordinate system to match the data, or scale the coordinate system manually to your own set of values.

```
  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);
```

Next you define the graphical objects you want displayed in the graph. This would include background objects, axes, axes labels, titles, footers, annotations, legends etc. Each graphical object can be customized according to its type. Each graphical object is added (using chartVu.addChartObj) to a display list maintained by the parent ChartView.

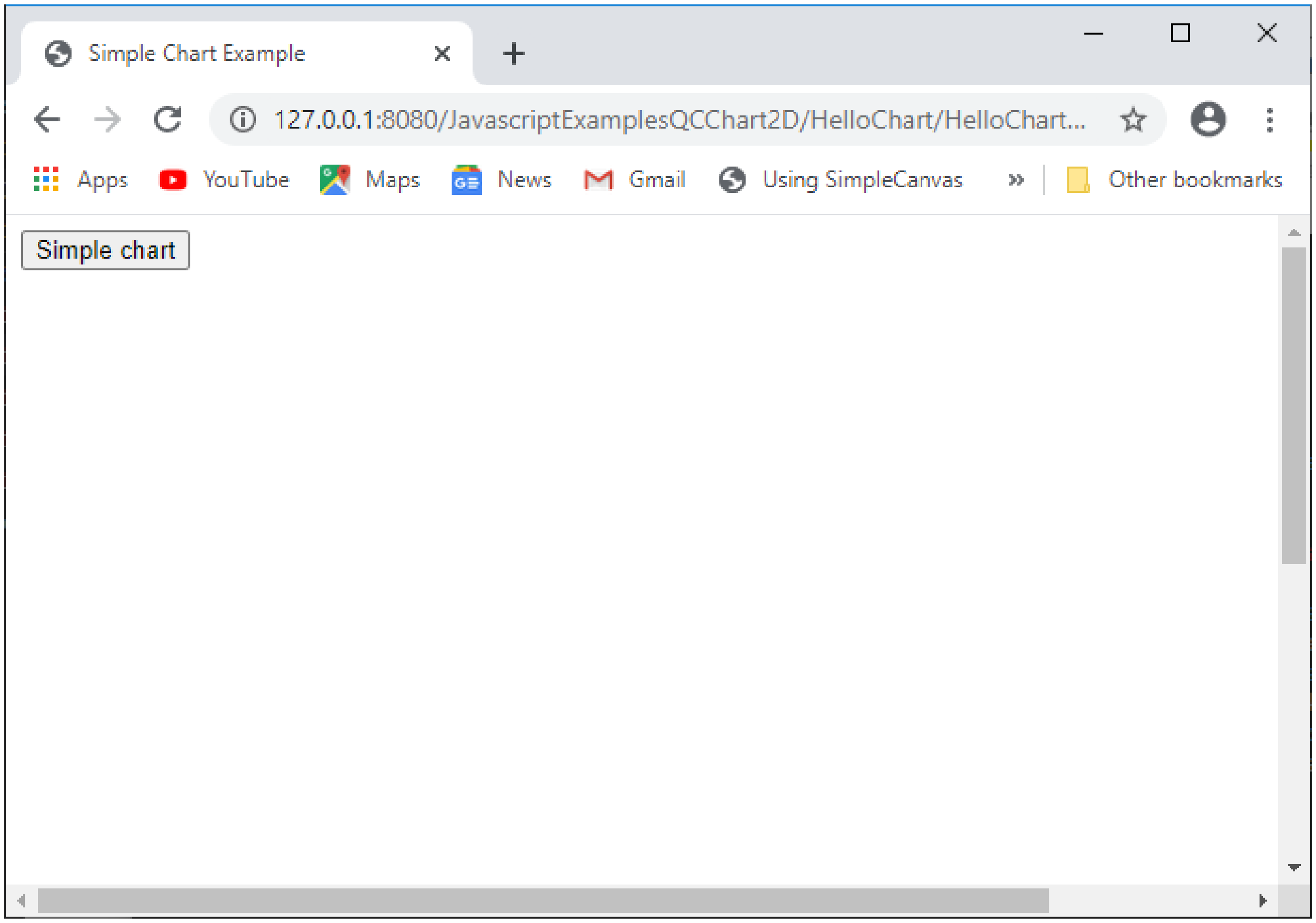
```
   let background = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.PLOT_BACKGROUND, QCChartTS.ChartColor.WHITE);
    chartVu.addChartObject(background);
    let xAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.X_AXIS);
    chartVu.addChartObject(xAxis);
    let yAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.Y_AXIS);
    chartVu.addChartObject(yAxis);
    let xAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(xAxis);
    xAxisLab.setTextFont(theFont);
    chartVu.addChartObject(xAxisLab);
    let yAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(yAxis);
    yAxisLab.setTextFont(theFont);
    chartVu.addChartObject(yAxisLab);
    let titleFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
    let yaxistitle = QCChartTS.AxisTitle.newAxisTitle(yAxis, titleFont, "Measurable work
output");
    chartVu.addChartObject(yaxistitle);
    let xaxistitle = QCChartTS.AxisTitle.newAxisTitle(xAxis, titleFont, "# MBAs/1000 employees");
    chartVu.addChartObject(xaxistitle);
    let xgrid = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.X_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
    chartVu.addChartObject(xgrid);
    let ygrid = QCChartTS.Grid.newGrid(xAxis, 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 EnronLabel = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1,
theFont, "Eronix", x1[94], y2[94], QCChartTS.ChartConstants.PHYS_POS);
    EnronLabel.setXJust(QCChartTS.ChartConstants.JUSTIFY_MAX);
    EnronLabel.setYJust(QCChartTS.ChartConstants.JUSTIFY_MAX);
    EnronLabel.setTextNudge(-4, 4);
    chartVu.addChartObject(EnronLabel);
    let IGG = QCChartTS.ChartText.newChartTextCoordsFontStringPosType(pTransform1, theFont,
"Quinn-Curtis", x1[0], y3[0], QCChartTS.ChartConstants.PHYS_POS);
```

```
    IGG.setXJust(QCChartTS.ChartConstants.JUSTIFY_MIN);
    IGG.setYJust(QCChartTS.ChartConstants.JUSTIFY_MAX);
    IGG.setTextNudge(4, 4);
    chartVu.addChartObject(IGG);
    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);
```

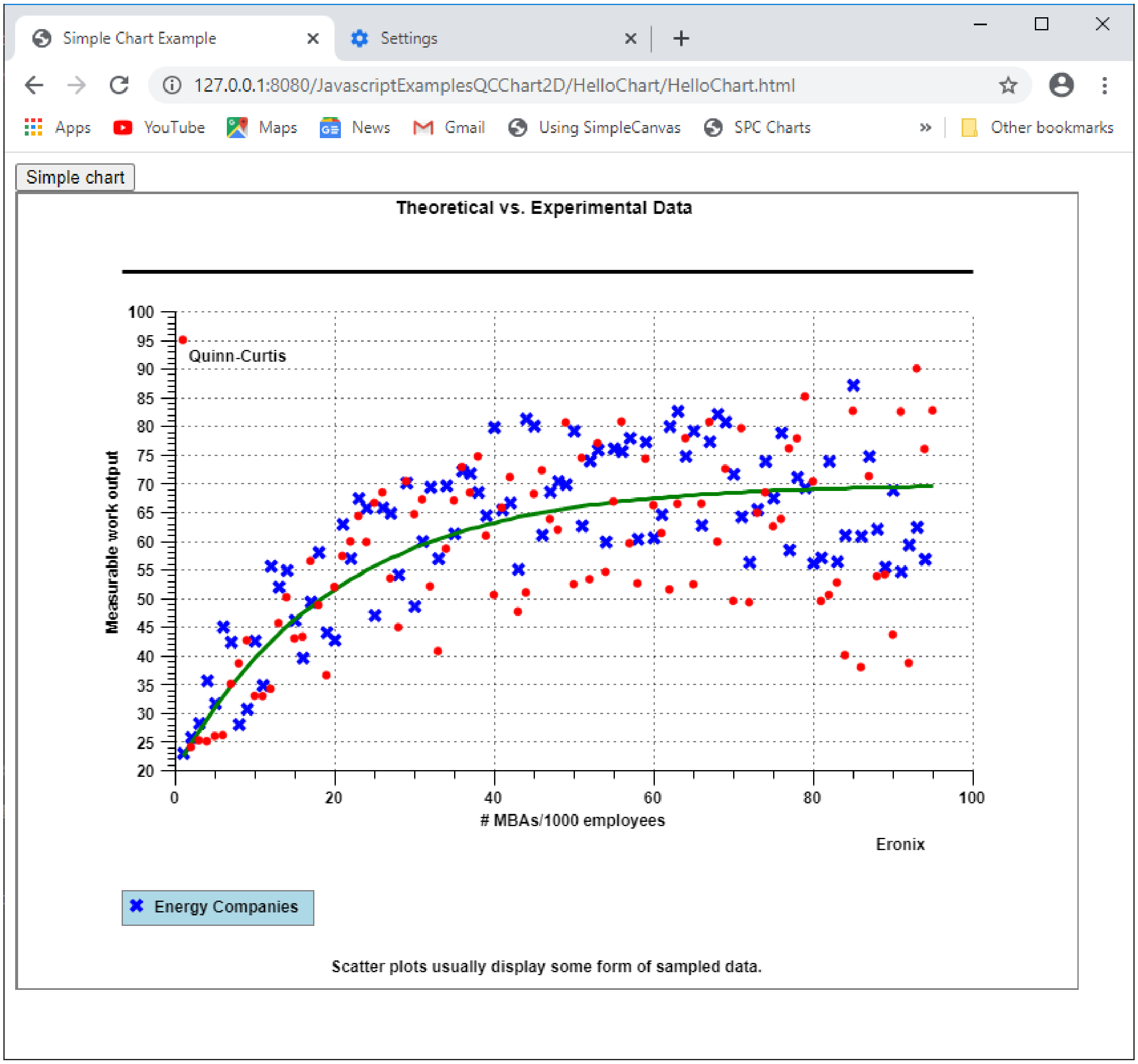
Assuming you have the two files setup correctly, and you have started some sort of test server which references the Quinn-Curtis/JSTS folder as its root directory, you should be able to run the test program by setting a browser to load the file at the URL:

http://127.0.0.1:8080/JavascriptExamplesQCChart2D/HelloChart/HelloChart.html

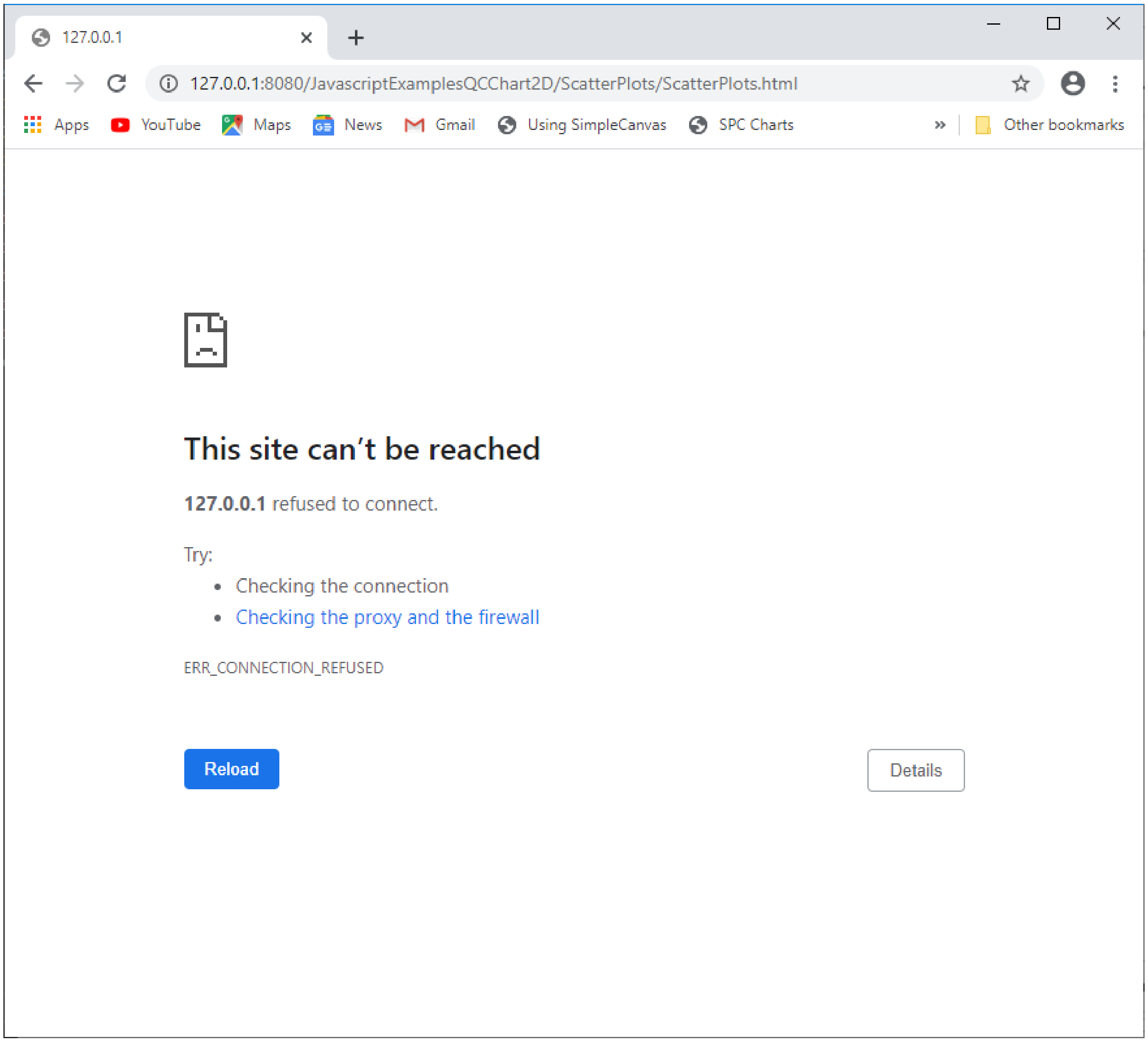
The result will be the following display:



showing just the button. Click the button and the chart will appear.



\
If you get a screen that looks like:



This site can't be reached

127.0.0.1 refused to connect.

Try:

- Checking the connection
- Checking the proxy and the firewall

ERR_CONNECTION_REFUSED

you probably did not start the http-server local server

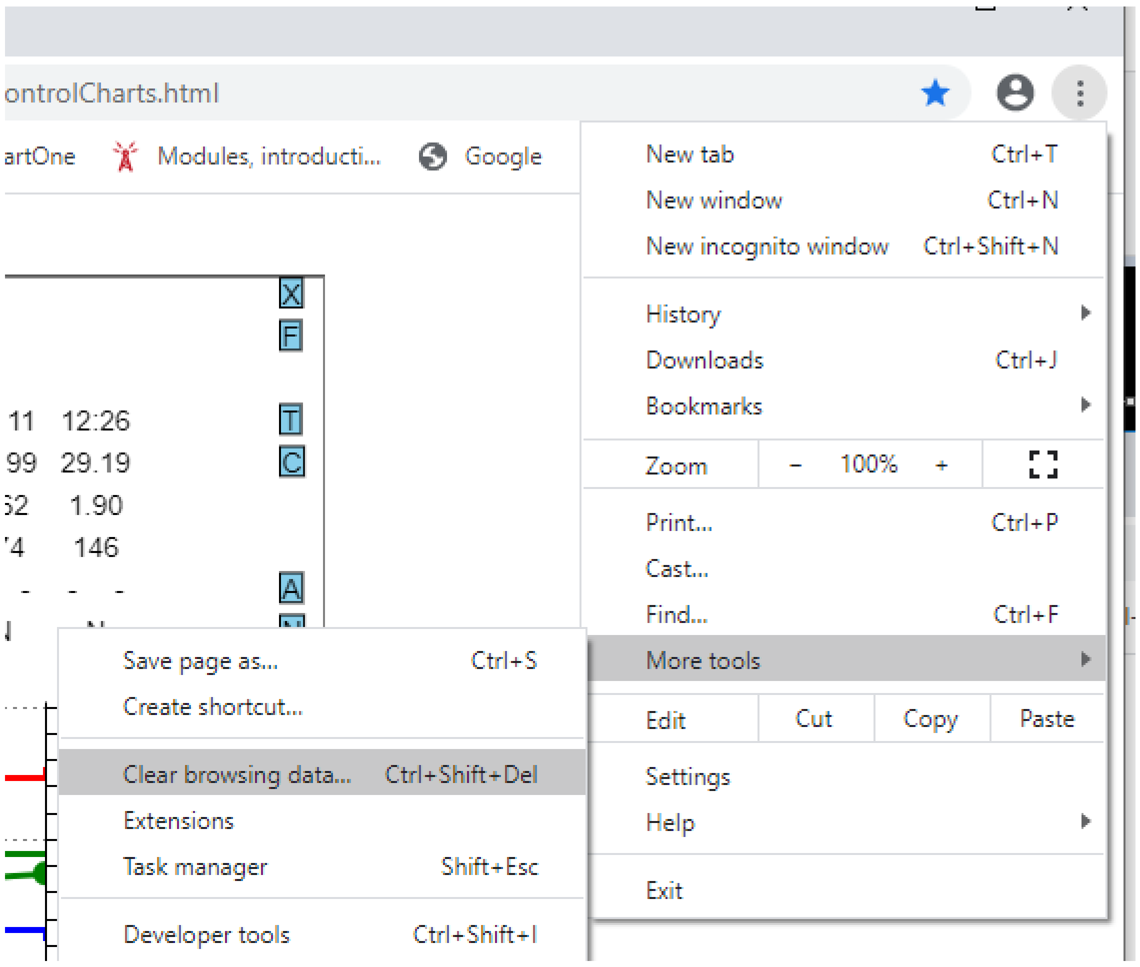
Start the npm "http-server" server using the batch file: startlocalserver.bat
The main folder containing both the JavaScript and TypeScript examples is Quinn-Curtis/JSTS. In that folder is a batch file (**startlocalserver.bat**) which will start the npm "http-server" local server, setting the directory it is located in as the root directory of the test server. All the batch file contains is:

```
http-server ./
```

where http-server starts the npm server, and "./" specifies that the current directory is to be the root folder of the server. In order to start the npm "http-server" that way it is assumed that you have installed the npm "http-server" command line server globally according to the instructions here: https://www.npmjs.com/package/http-server You will need a relatively current version of npm to properly install the http-server package.

Clearing the Browser Cache

You will find that as you make changes to your JavaScript code, then go and test it in a browser, the changes do not seem to go into effect. This is because the browsers all cache JavaScript code and do not reload them automatically. This will cause the browser to execute your old code and not the new, modified code. It is very annoying. You have two options. You can use the browser settings and clear the browser cache each time you want to test new JavaScript code in the browser. Like this for Chrome:

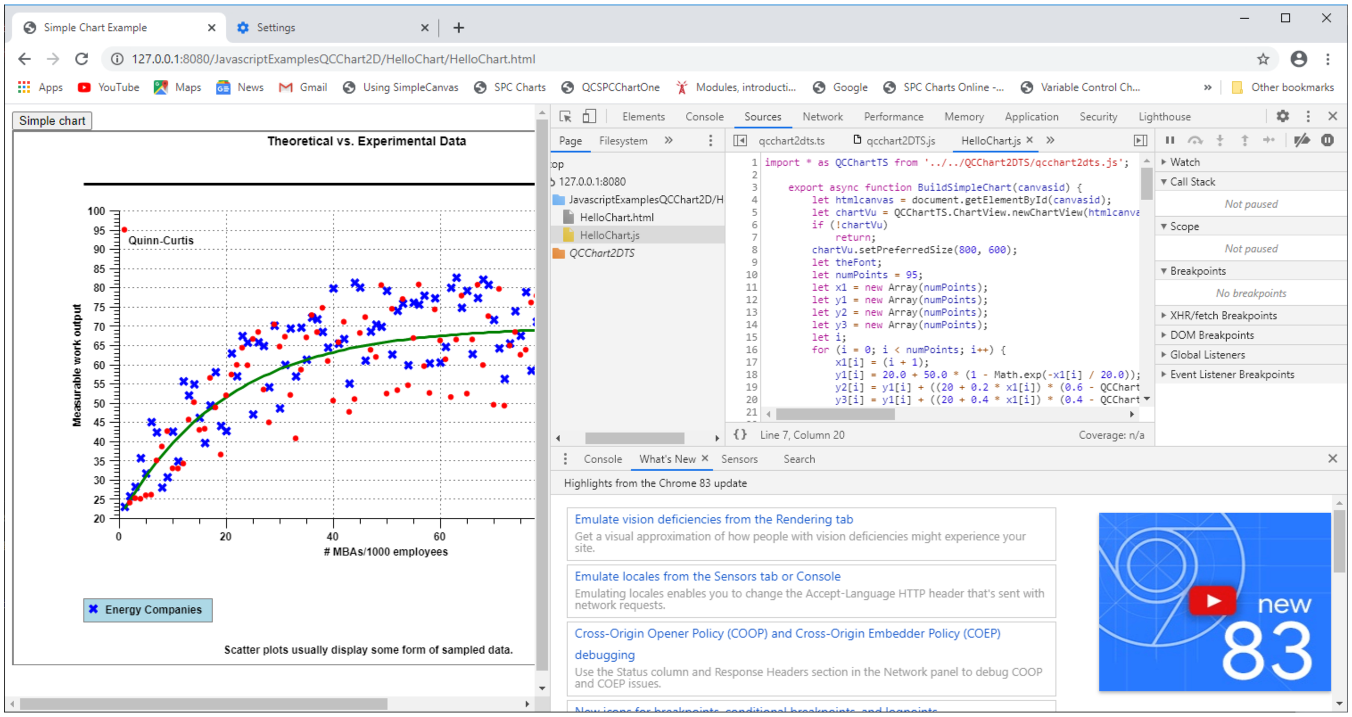


Or, the preferred method, is that you can turn off caching from within the browser Development Mode. Here is a good discussion of that option: https://nicholasbering.ca/tools/2016/10/09/devtools-disable-caching/ . That article discusses the options for Chrome, Firefox and Safari, but Microsoft Edge has a similar option.

Debugging

If for some reason the chart does not appear you will have start debugging.

There are ways to setup Visual Studio Code for integrated debugging. But it looks pretty complicated. Instead, we choose to just use the integrated F12 debugging available in all of the major browsers. Once you point the browser to the HelloChart.html file you should see the Simple Chart button. Press F12 and the browsers integrated debugger will appear on the right. The debugger portion of the window will look something like:



```
import * as QCChartTS from '../../QCChart2DTS/qcchart2dts.js';

    export async function BuildSimpleChart(canvasid) {
        let htmlcanvas = document.getElementById(canvasid);
        let chartVu = QCChartTS.ChartView.newChartView(htmlcanva
        if (!chartVu)
            return;
        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 - QCChart
            y3[i] = y1[i] + ((20 + 0.4 * x1[i]) * (0.4 - QCChart
```

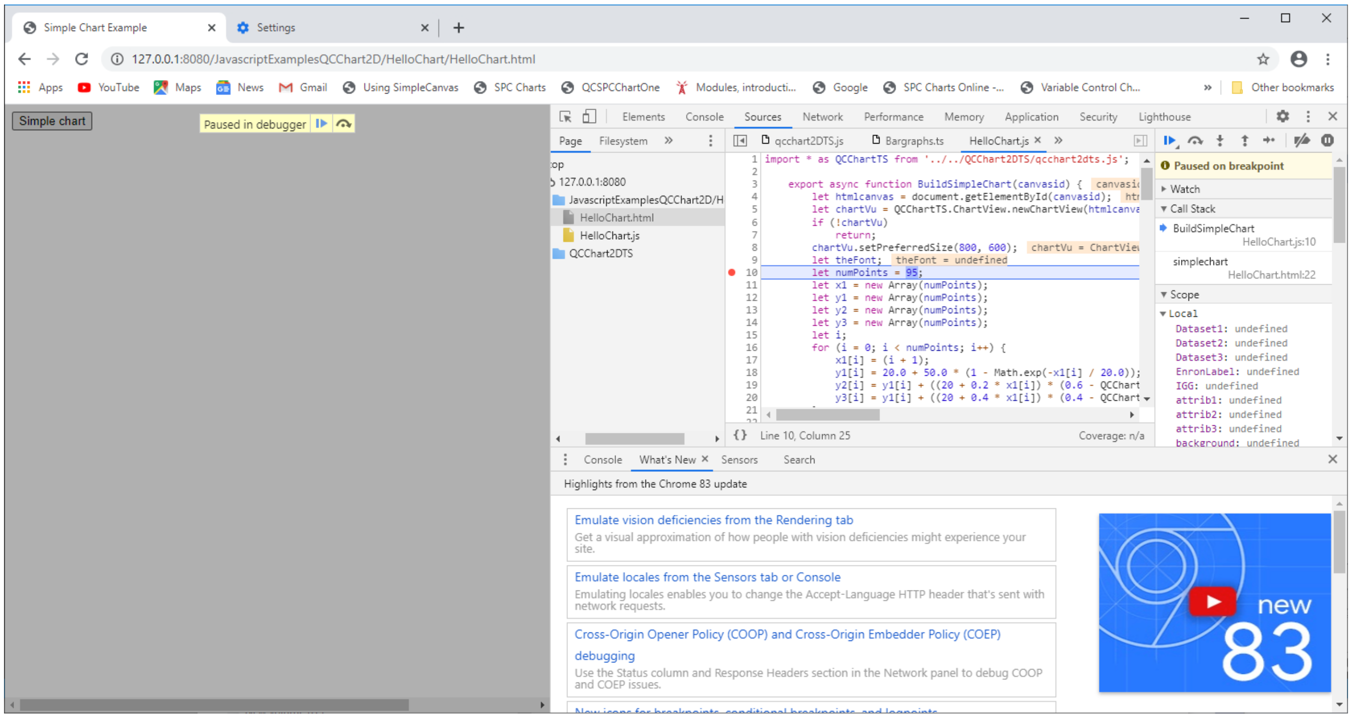
Highlights from the Chrome 83 update

Emulate vision deficiencies from the Rendering tab
Get a visual approximation of how people with vision deficiencies might experience your site.

Emulate locales from the Sensors tab or Console
Emulating locales enables you to change the Accept-Language HTTP header that's sent with network requests.

Cross-Origin Opener Policy (COOP) and Cross-Origin Embedder Policy (COEP) debugging
Use the Status column and Response Headers section in the Network panel to debug COOP and COEP issues.

You should be able to select either the HelloChart.js file, or the HelloChart.html file to view in the main window. All of the things you normally want to do with a debugger (view values, step through, set breakpoints, etc.) you can do from this window. So if you set a breakpoint in the HelloChart.js file, and then click the Simple Chart button in the left hand side of the browser, program execution will stop in the code window when the breakpoint is hit.



```
import * as QCChartTS from '../../QCChart2DTS/qcchart2dts.js';

    export async function BuildSimpleChart(canvasid) {  canvasi
        let htmlcanvas = document.getElementById(canvasid);  htm
        let chartVu = QCChartTS.ChartView.newChartView(htmlcanva
        if (!chartVu)
            return;
        chartVu.setPreferredSize(800, 600);  chartVu = ChartView
        let theFont;  theFont = undefined
        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 - QCChart
            y3[i] = y1[i] + ((20 + 0.4 * x1[i]) * (0.4 - QCChart
```

Line 10, Column 25   Coverage: n/a

Paused on breakpoint

Call Stack
BuildSimpleChart   HelloChart.js:10
simplechart   HelloChart.html:22

Scope
Local
```
Dataset1: undefined
Dataset2: undefined
Dataset3: undefined
EnronLabel: undefined
IGG: undefined
attrib1: undefined
attrib2: undefined
attrib3: undefined
background: undefined
```

Highlights from the Chrome 83 update

Emulate vision deficiencies from the Rendering tab
Get a visual approximation of how people with vision deficiencies might experience your site.

Emulate locales from the Sensors tab or Console
Emulating locales enables you to change the Accept-Language HTTP header that's sent with network requests.

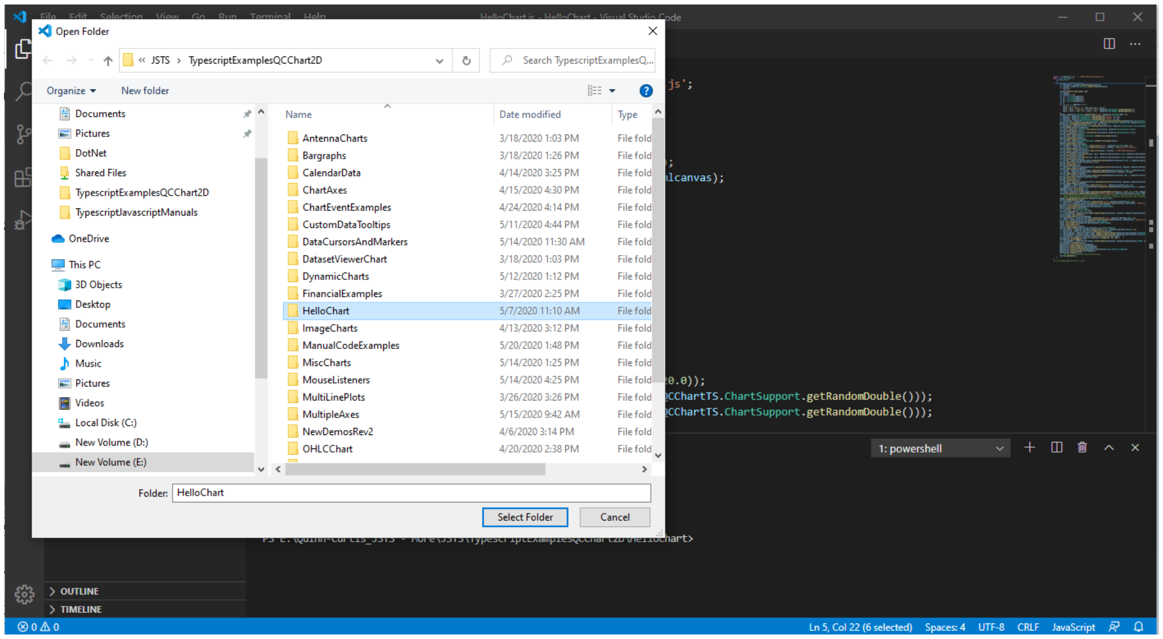
Cross-Origin Opener Policy (COOP) and Cross-Origin Embedder Policy (COEP) debugging
Use the Status column and Response Headers section in the Network panel to debug COOP and COEP issues.

Using TypeScript

The QCChart2DTS library is written in TypeScript. We went to the trouble of adjusting the source code to compile using the strictest mode (**"strict": true** under the “compilerOptions” section of the projects tsconfig.json file) of the TypeScript compiler. The output of the compiler is the qcchart2dts.js file, which is the JavaScript version of the library. That is the file that you reference inside your own TypeScript program. All of the class, property and function names inside the library are the same, regardless of whether you calling it from TypeScript or JavaScript. The qcchart2dts.js file is located in the Quinn-Curtis/JSTS/QCChart2DTS folder. So your TypeScript source files which reference the QCChart2DTS library must import it from that location. We specify the location of the qcchart2dts.js file relative to the source folder of the file we will be working with, using:

```
import * as  QCChartTS from '../../QCChart2DTS/qcchart2dts.js';
```

Start off by creating a folder unique to your intended code, under our JSTS/TypescriptExamplesQCChart2D/ folder. We will use the folder name HelloChart. *Since we include the HelloChart folder as one of our examples, if you plan on following this tutorial exactly, just rename the existing HelloChart folder something else, HelloChart2 for example.* In its simplest form, that folder will contain just two files: your test HTML page (HelloChart.html) and the JavaScript source page (HelloChart.ts). In this example, the HelloChart.html page can just be an bare-bones HTML page, the same as our TypescriptExamplesQCChart2D/HelloChart/HelloChart.html. The only difference is in the embedded filename used to access the HelloChart.js JavaScript file. Even though your HelloChart.ts source file will be written using TypeScript, the HTML page cannot import it in that format. It must be converted to the JavaScript equivalent, and that is what the TypeScript compiler does.



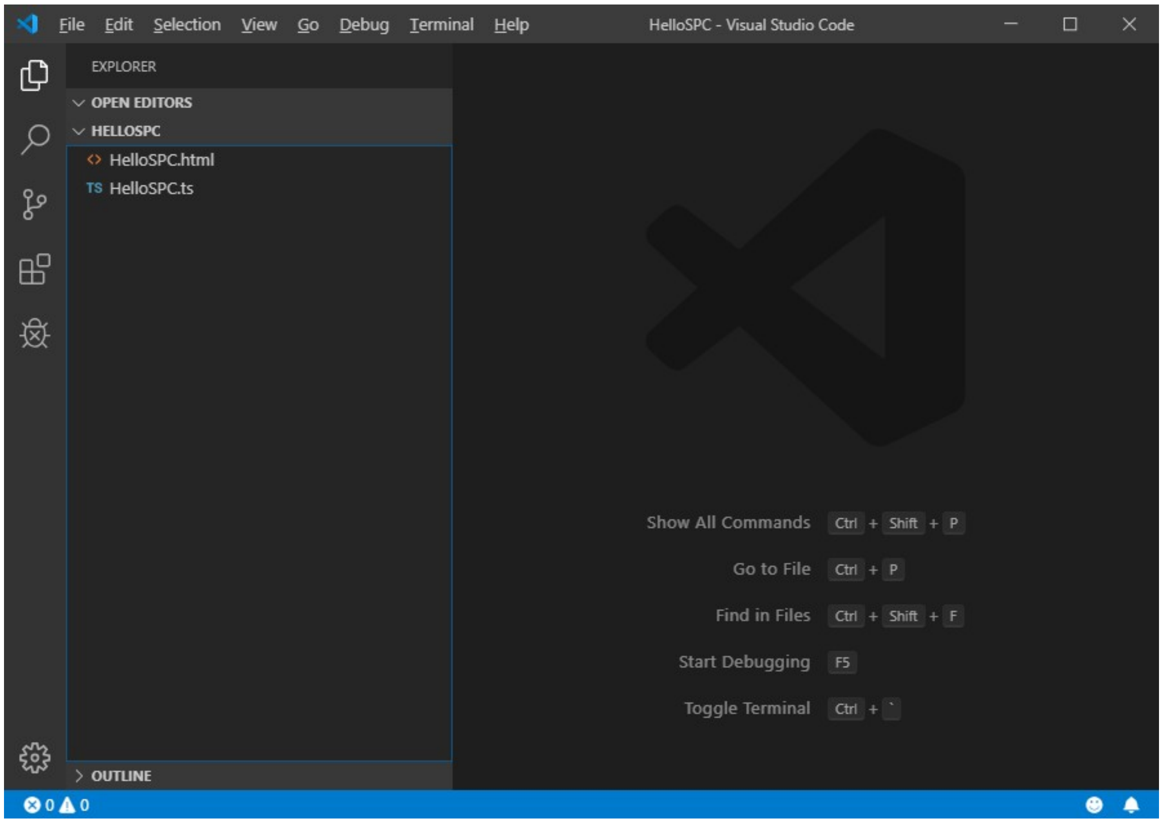
Name	Date modified	Type
AntennaCharts	3/18/2020 1:03 PM	File fold
Bargraphs	3/18/2020 1:26 PM	File fold
CalendarData	4/14/2020 3:25 PM	File fold
ChartAxes	4/15/2020 4:30 PM	File fold
ChartEventExamples	4/24/2020 4:14 PM	File fold
CustomDataTooltips	5/11/2020 4:44 PM	File fold
DataCursorsAndMarkers	5/14/2020 11:30 AM	File fold
DatasetViewerChart	3/18/2020 1:03 PM	File fold
DynamicCharts	5/12/2020 1:12 PM	File fold
FinancialExamples	3/27/2020 2:25 PM	File fold
HelloChart	5/7/2020 11:10 AM	File fold
ImageCharts	4/13/2020 3:12 PM	File fold
ManualCodeExamples	5/20/2020 1:48 PM	File fold
MiscCharts	5/14/2020 1:25 PM	File fold
MouseListeners	5/14/2020 4:25 PM	File fold
MultiLinePlots	3/26/2020 3:26 PM	File fold
MultipleAxes	5/15/2020 9:42 AM	File fold
NewDemosRev2	4/6/2020 3:14 PM	File fold
OHLCChart	4/20/2020 2:38 PM	File fold

Folder: HelloChart

Ln 5, Col 22 (6 selected)   Spaces: 4   UTF-8   CRLF   JavaScript

From your JavaScript Development Environment (Visual Studio Code in our case), open the HelloChart folder using the IDE File | Open Folder option, and select the HelloChart folder you created.

Next, create two new files in the HelloChart directory: HelloChart.html and HelloChart.ts, using the File | New File options of the IDE. In VS Code they are initially named Untitled-1 and Untitled-2, but when you go to File | Save, or File | Save As you will be prompted for their proper names and then you can rename them HelloChart.html and HelloChart.ts . So you should end up with a folder named HelloChart containing two blank file, HelloChart.html and HelloChart.ts.


File Edit Selection View Go Debug Terminal Help
HelloSPC - Visual Studio Code
EXPLORER
OPEN EDITORS
HELLOSPC
HelloSPC.html
HelloSPC.ts
OUTLINE
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Start Debugging F5
Toggle Terminal Ctrl + `

Copy the code below into the HelloChart.html file.

```
<!DOCTYPE html>
<head>
    <meta charset="utf-8" />
    <title>Simple Chart Example</title>
</head>
<body>

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

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


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

     var hellochart = new HelloChart();


          function simplechart() {
            hellochart.BuildSimpleChart("chartCanvas1");
          }


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

        <br>
        <br>

</body>
</html>
```

Note that the import statement looks to the current directory for the HelloChart.js file.

```
import { HelloChart} from './HelloChart.js';
```

The import statement loads the HelloChart.js file and makes the HelloChart class inside available as the class HelloChart. When that is completed, an instance of the HelloChart class found in that file, is instantiated using new HelloChart(). Until you compile the HelloChart project, the HelloChart.js file will not exist. When the project is compiled, the HelloChart.ts file is compiled into the HelloChart.js file and placed in the same folder, HelloChart, from which it is loaded when needed.

When the button (id = simplechart_menuitem) is pressed, it triggers a call to the BuildSimpleChart function imported from the HelloChart.js file. Note that the event handler (simplechart) for the button:

```
          document.getElementById("simplechart_menuitem").onclick = simplechart;
```

is installed inside the JavaScript block of code. Since JavaScript modules are loaded asynchronously, you **cannot** assign the event handler when the button is created in the HTML. That is because the JavaScript code inside the <script type="module"> block has not yet been loaded when the button is defined, and therefore the simplechart function contained within cannot be referenced. If you attempt to do so it will be assigned a null (or undefined) value. But if you do the event handler assignment inside the <script type="module"> code block, after the simplechart function has been defined, it will work.

Save the HelloChart.html file and go to the HelloChart.ts file. You can copy the code below into that file.

```
import * as  QCChartTS from '../../QCChart2DTS/qcchart2dts.js';


export class HelloChart {

    public constructor() {

    }


    public async  BuildSimpleChart(canvasid: string) {

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

        let chartVu: QCChartTS.ChartView = QCChartTS.ChartView.newChartView(htmlcanvas);
        if (!chartVu) return;
        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);
        let xAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.X_AXIS);
        chartVu.addChartObject(xAxis);
        let yAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
        chartVu.addChartObject(yAxis);
        let xAxisLab: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(xAxis);
        xAxisLab.setTextFont(theFont);
        chartVu.addChartObject(xAxisLab);
        let yAxisLab: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(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(yAxis, titleFont,
"Measurable work output");
        chartVu.addChartObject(yaxistitle);
        let xaxistitle: QCChartTS.AxisTitle = QCChartTS.AxisTitle.newAxisTitle(xAxis, titleFont,
"# MBAs/1000 employees");
        chartVu.addChartObject(xaxistitle);
        let xgrid: QCChartTS.Grid = QCChartTS.Grid.newGrid(xAxis, yAxis,
QCChartTS.ChartConstants.X_AXIS, QCChartTS.ChartConstants.GRID_MAJOR);
        chartVu.addChartObject(xgrid);
        let ygrid: QCChartTS.Grid = QCChartTS.Grid.newGrid(xAxis, 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 toolTipFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Arial",
QCChartTS.ChartFont.REGULAR, 12);
        let datatooltip: QCChartTS.DataToolTip = QCChartTS.DataToolTip.newDataToolTip(chartVu);
        let xValueTemplate: QCChartTS.NumericLabel =
QCChartTS.NumericLabel.newNumericLabelFormatDecs(QCChartTS.ChartConstants.DECIMALFORMAT, 0);
        let yValueTemplate: QCChartTS.NumericLabel =
QCChartTS.NumericLabel.newNumericLabelFormatDecs(QCChartTS.ChartConstants.DECIMALFORMAT, 1);
        let textTemplate: QCChartTS.ChartText =
QCChartTS.ChartText.newChartTextFontString(toolTipFont, "");
        textTemplate.setTextBgColor(QCChartTS.ChartColor.fromRgb(255, 255, 204));
        textTemplate.setTextBgMode(true);
        let toolTipSymbol: QCChartTS.ChartSymbol =
QCChartTS.ChartSymbol.newChartSymbol(pTransform1, QCChartTS.ChartConstants.SQUARE,
QCChartTS.ChartAttribute.newChartAttributeColor(QCChartTS.ChartColor.GREEN));
        toolTipSymbol.setSymbolSize(10);
        datatooltip.setTextTemplate(textTemplate);
        datatooltip.setXValueTemplate(xValueTemplate);
        datatooltip.setYValueTemplate(yValueTemplate);
        datatooltip.setDataToolTipFormat(QCChartTS.ChartConstants.DATA_TOOLTIP_XY_ONELINE);
        datatooltip.setToolTipSymbol(toolTipSymbol);
        datatooltip.setEnable(true);
        //   chartVu.setCurrentMouseListener(datatooltip);

        chartVu.updateDraw();

    }


}
```

In the HelloChart.ts file, the body of the program is setup as a TypeScript class, named HelloChart. It has two member functions: the public default constructor, the public BuildSimpleChart member function which is called from the HelloChart.html HTML file. We could also have setup up the program as standalone functions not enclosed by a class structure, similar what is done in the JavaScript example HelloChart, found under JavascriptExamplesQCChart2D/HelloChart.

Note that the QCChart2DTS library is being imported from its location in the file system, relative to the HelloChart.js file.

```
import * as  QCChartTS from '../../QCChart2DTS/qcchart2dts.js';
```

The "import * as QCChartTS" in the import statement assigns all classes, static functions within a class, properties and constants in the software to the namespace QCChartTS, so you must preface any reference to the formal name of any classes or static constants, found within the QCChart2D library by explicitly referencing that namespace.

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

 let chartVu: QCChartTS.ChartView = QCChartTS.ChartView.newChartView(htmlcanvas);
 if (!chartVu) return;
 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);
```

In broad terms, first you create an instance of a ChartView object using a static constructor.:

```
export async function BuildSimpleChart(canvasid) {
 let htmlcanvas: QCChartTS.Canvas = <QCChartTS.Canvas>document.getElementById(canvasid);

 let chartVu: QCChartTS.ChartView = QCChartTS.ChartView.newChartView(htmlcanvas);
 if (!chartVu) return;
```

passing in the required parameters.

Next you get, or create your data and assign it to one or more of our Dataset objects.

```
 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()));
}
```

Next, you define the coordinate system you want to display the data in. You can auto-scale the coordinate system to match the data, or scale the coordinate system manually to your own set of values.

```
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);
```

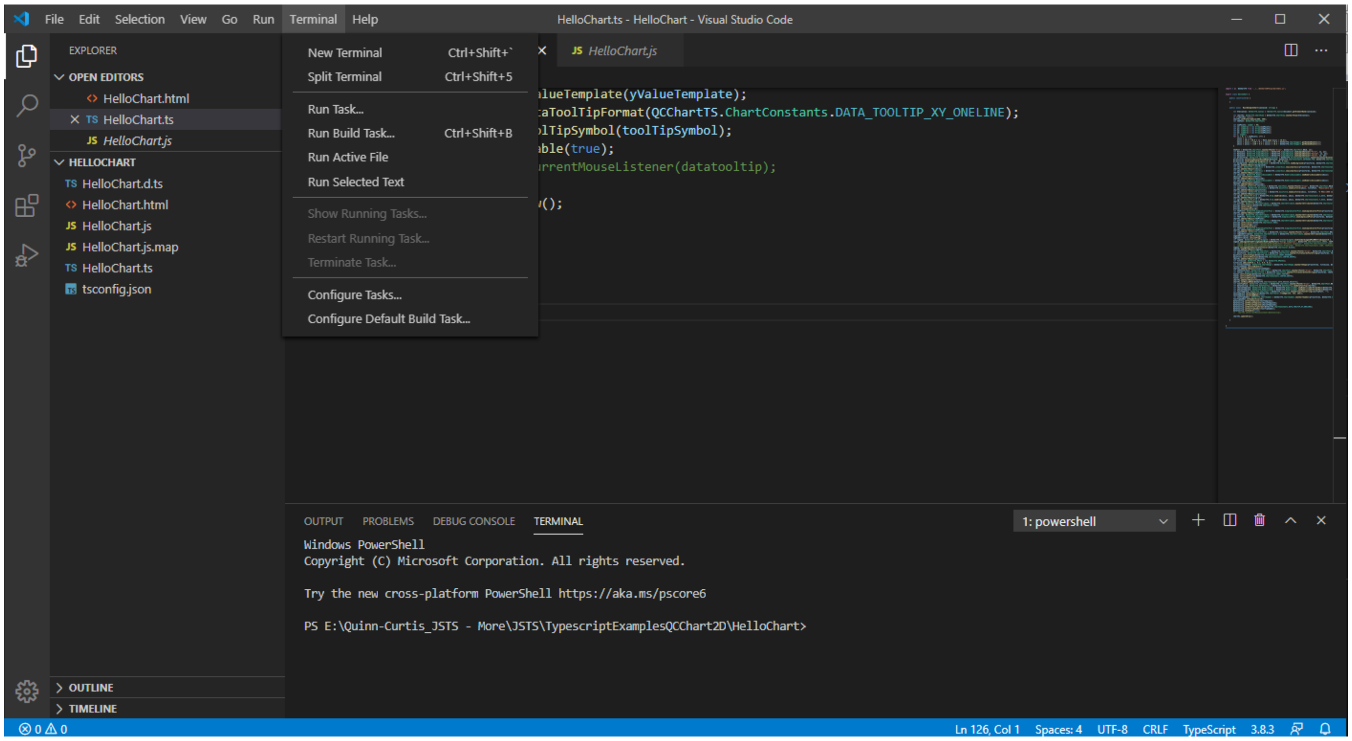
Next you define the graphical objects you want displayed in the graph. This would include background objects, axes, axes labels, titles, footers, annotations, legends etc. Each graphical object can be customized according to its type. Each graphical object is added (chartVu.addChartObj) to a display list maintained by the parent ChartView.

```
let background: QCChartTS.Background = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.PLOT_BACKGROUND, QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(background);
let xAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.X_AXIS);
chartVu.addChartObject(xAxis);
let yAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
chartVu.addChartObject(yAxis);
let xAxisLab: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(xAxis);
xAxisLab.setTextFont(theFont);
chartVu.addChartObject(xAxisLab);
let yAxisLab: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(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(yAxis, titleFont,
"Measurable work output");
```

```
chartVu.addChartObject(yaxistitle);
let xaxistitle: QCChartTS.AxisTitle = QCChartTS.AxisTitle.newAxisTitle(xAxis, titleFont, "# MBAs/
1000 employees");
chartVu.addChartObject(xaxistitle);
let xgrid: QCChartTS.Grid = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.X_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
chartVu.addChartObject(xgrid);
let ygrid: QCChartTS.Grid = QCChartTS.Grid.newGrid(xAxis, 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);
```

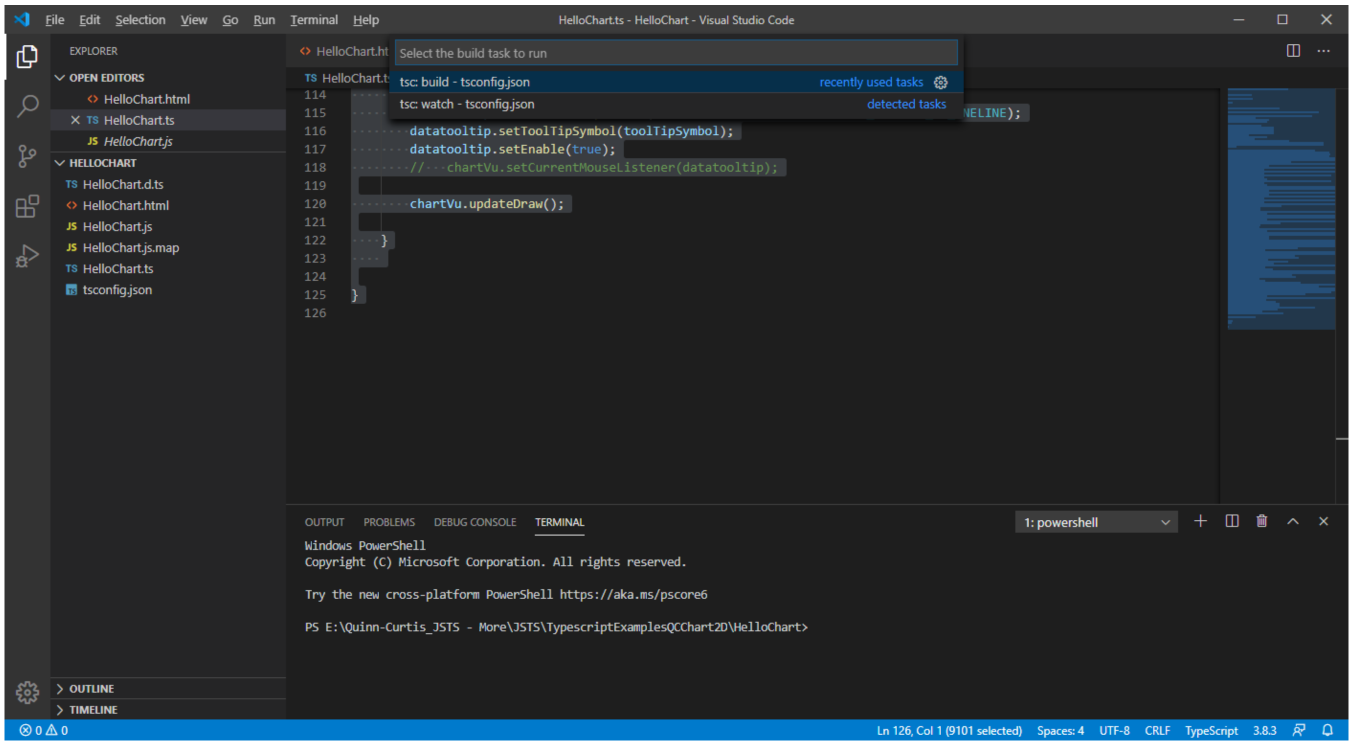
Unlike the JavaScript version of the same program, once the HelloChart.ts TypeScript file is setup correctly you will need to compile it in order to generate the HelloChart.js file. **In order to compile the project you need a tsconfig.json file in the project folder.** Just copy the one from the TypescriptExamplesQCChart2D/SimpleLinePlots example and place it in the TypescriptExamplesQCChart2D/HelloChart folder. See Chapter 1 for more information about the options found in the tsconfig.json file.



```
alueTemplate(yValueTemplate);
:aToolTipFormat(QCChartTS.ChartConstants.DATA_TOOLTIP_XY_ONELINE);
>lTipSymbol(toolTipSymbol);
able(true);
irrentMouseListener(datatooltip);

)();
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\Quinn-Curtis_JSTS - More\JSTS\TypescriptExamplesQCChart2D\HelloChart>
```

Once that is in place, compile the HelloChart.ts TypeScript file by selecting Terminal | Run Build Task.



```
                                                    NELINE);
        datatooltip.setToolTipSymbol(toolTipSymbol);
        datatooltip.setEnable(true);
        //   chartVu.setCurrentMouseListener(datatooltip);

        chartVu.updateDraw();

    }


}

```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\Quinn-Curtis_JSTS - More\JSTS\TypescriptExamplesQCChart2D\HelloChart>
```

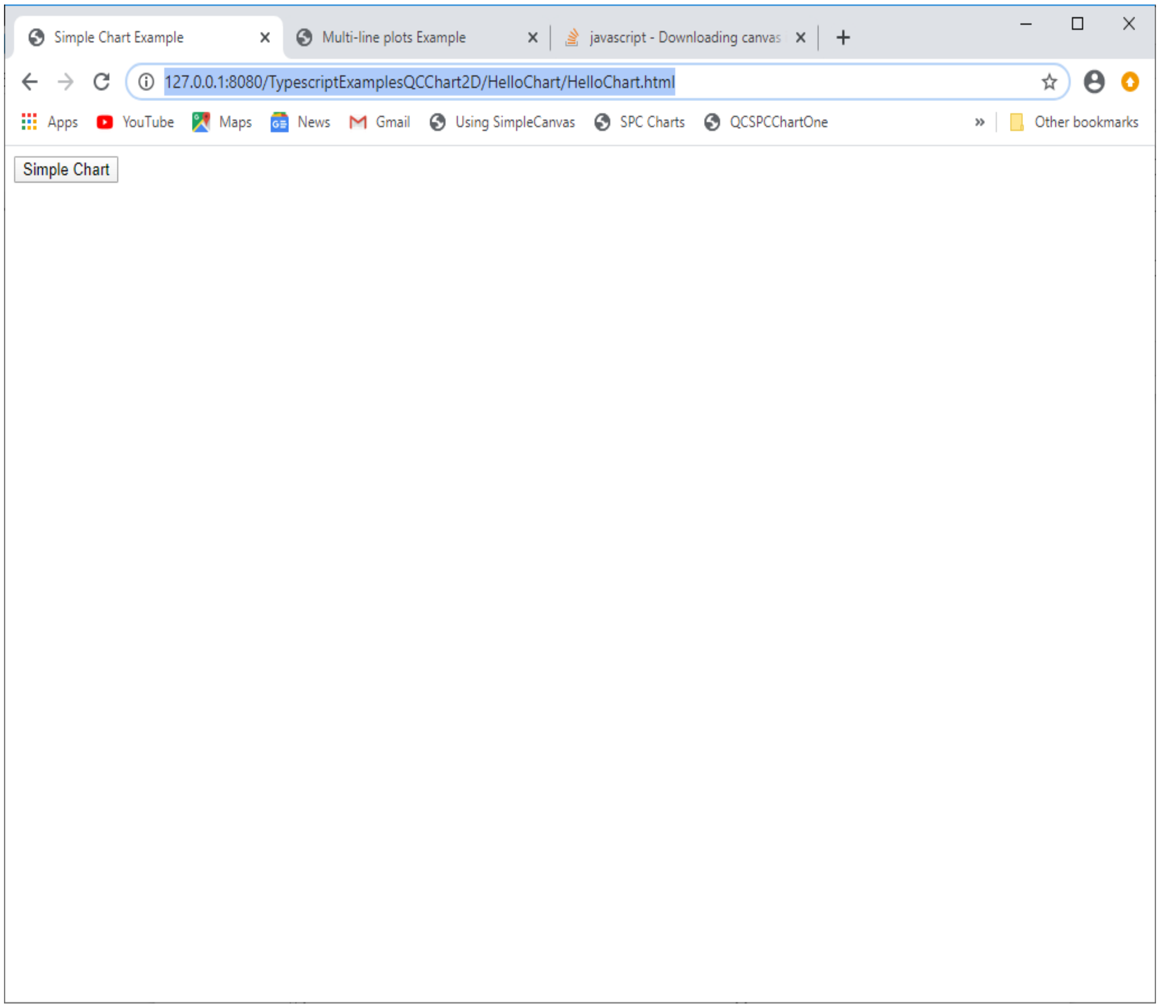
Then you need to select tsc: build – tsconfig.json from the drop down list (its hard to see).

This will use the tsconfig.json file in the current directory to control the TypeScript to JavaScript compilation. It will only take a second. After that you will either have a list of errors to fix, or the JavaScript equivalent of the original TypeScript file, with the filename HelloChart.js and located in the HelloChart folder, next to the other files.

Other files also created during compilation are the HelloChart.d.ts file (contains TypeScript type declarations for the HelloChart.js file), and the HelloChart.js.map (contains a mapping of the HelloChart.js file to the HelloChart.ts file) which is used in debugging the HelloChart.ts source when you are actually executing the associated HelloChart.js JavaScript.

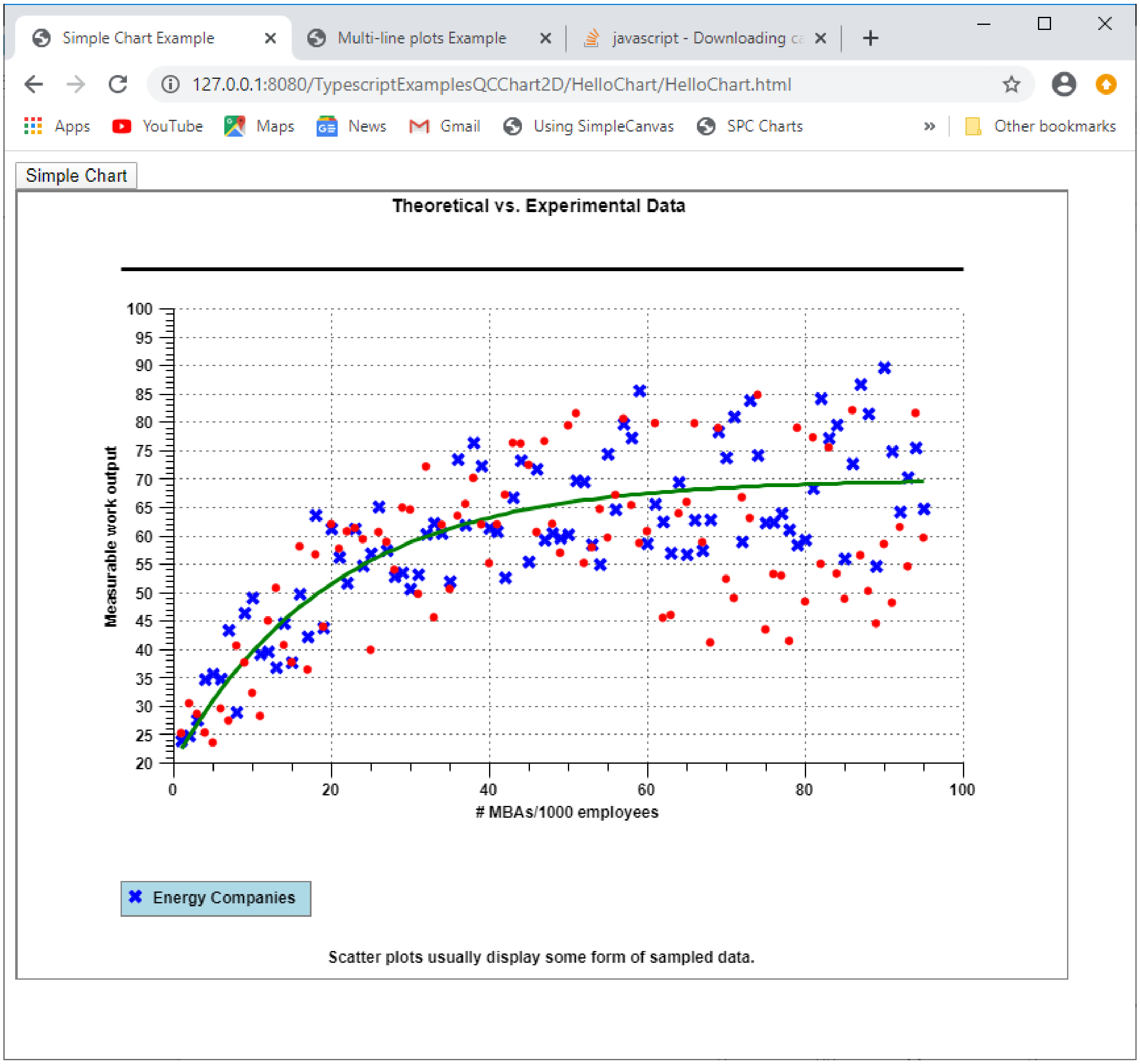
Assuming everything is setup correctly, and you have started some sort of test server which references the Quinn-Curtis/JSTS folder as its root directory, you should be able to run the test program by setting a browser to load the file at the URL:

http://127.0.0.1:8080/TypescriptExamplesQCChart2D/HelloChart/HelloChart.html

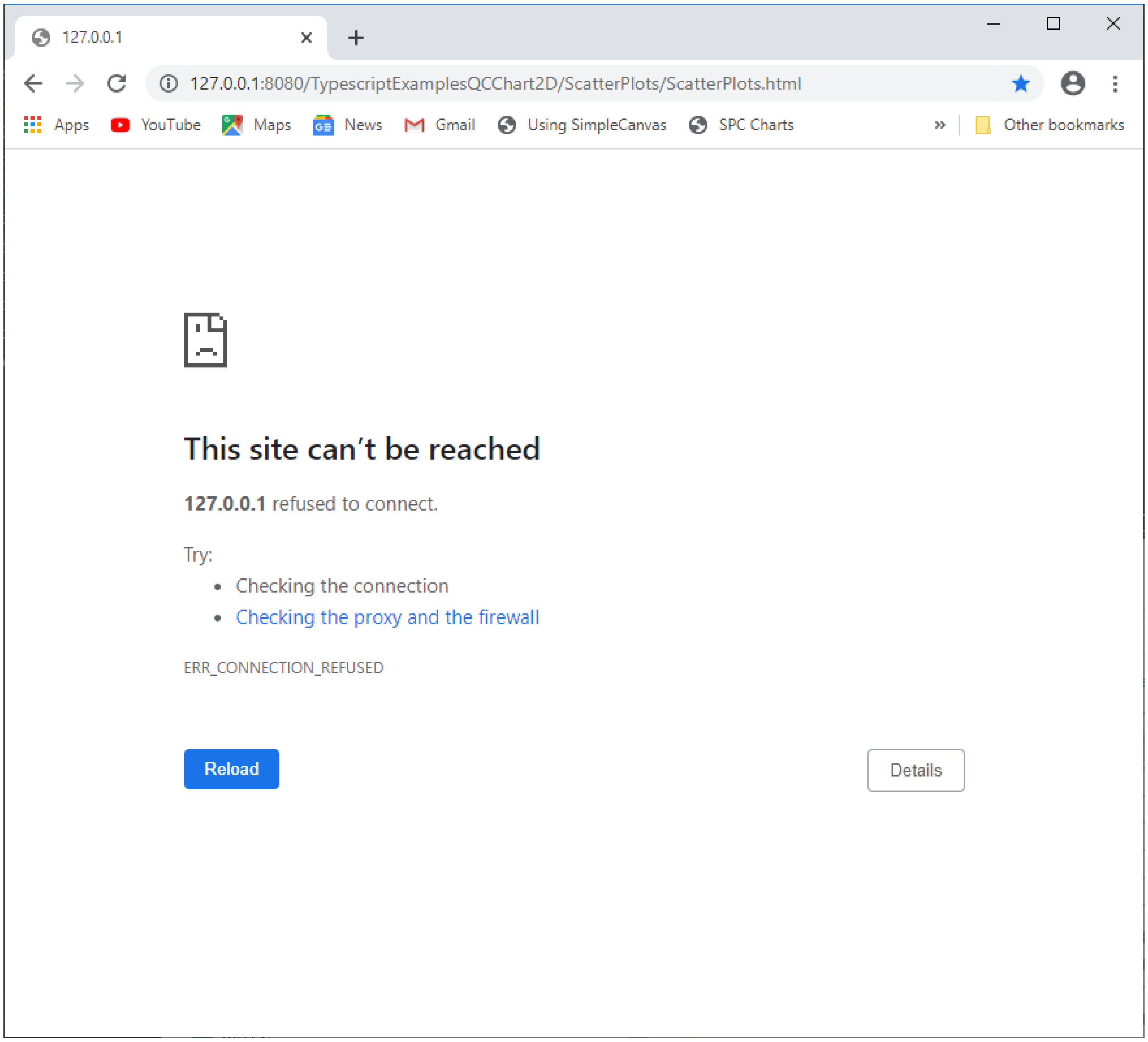


The result will be the following display:

showing just the **Simple Chart** button. Click the button and the chart will appear.


Simple Chart Example
Multi-line plots Example
javascript - Downloading c
127.0.0.1:8080/TypescriptExamplesQCChart2D/HelloChart/HelloChart.html
Apps
YouTube
Maps
News
Gmail
Using SimpleCanvas
SPC Charts
Other bookmarks
Simple Chart
Theoretical vs. Experimental Data
Measurable work output
# MBAs/1000 employees
Energy Companies
Scatter plots usually display some form of sampled data.

If you get a screen that looks like:



This site can't be reached

127.0.0.1 refused to connect.

Try:

- Checking the connection
- Checking the proxy and the firewall

ERR_CONNECTION_REFUSED

you probably did not start the http-server local server

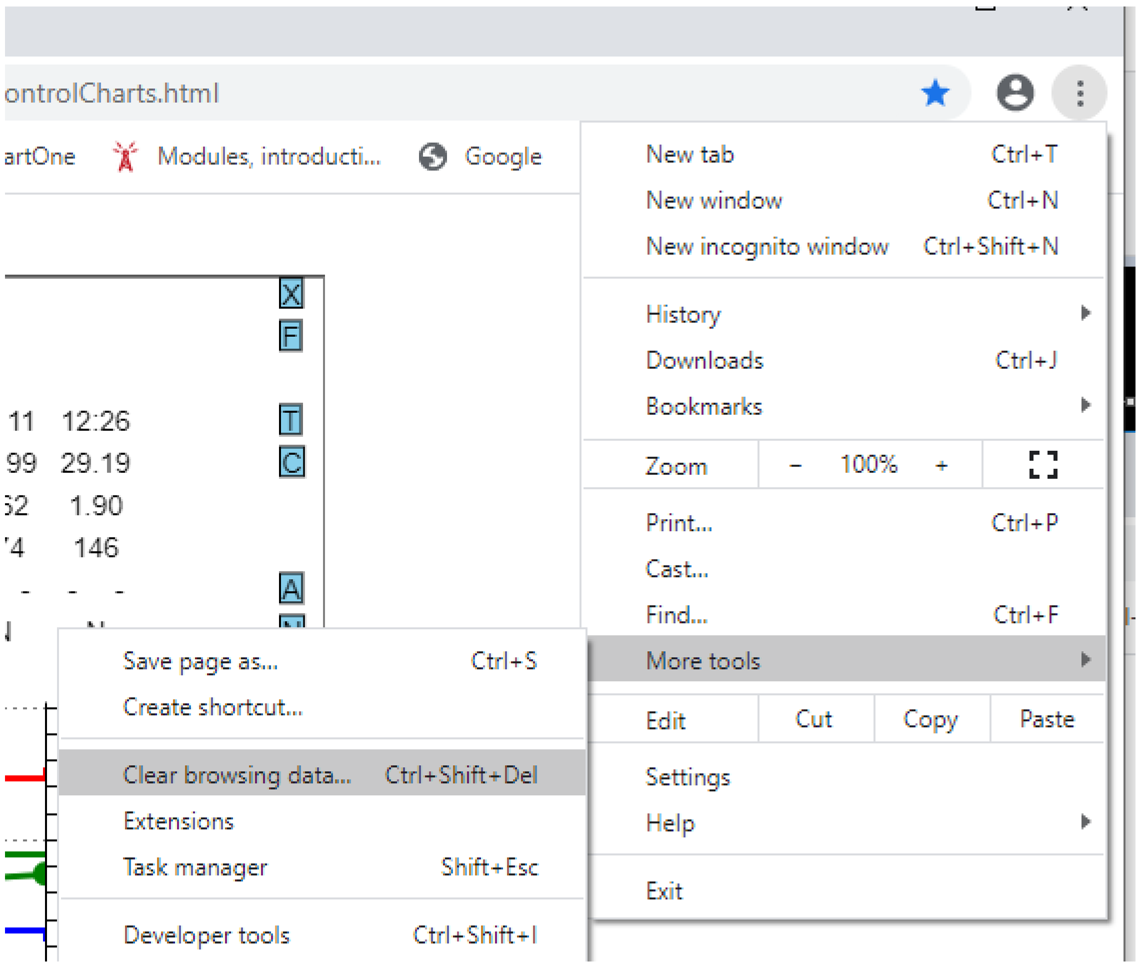
Start the npm "http-server" server using the batch file: startlocalserver.bat
The main folder containing both the JavaScript and TypeScript examples is Quinn-Curtis/JSTS. In that folder is a batch file (**startlocalserver.bat**) which will start the npm "http-server" local server, setting the directory it is located in as the root directory of the test server. All the batch file contains is:

```
http-server ./
```

where http-server starts the npm server, and "./" specifies that the current directory is to be the root folder of the server. In order to start the npm "http-server" that way it is assumed that you have installed the npm "http-server" command line server globally according to the instructions here: https://www.npmjs.com/package/http-server You will need a relatively current version of npm to properly install the http-server package.

Clearing the Browser Cache

You will find that as you make changes to your TypeScript code, compile it into JavaScript, then go and test it in a browser, the changes do not seem to go into effect. This is because the browsers all cache JavaScript code and do not reload them automatically. This will cause the browser to execute your old code and not the new, modified code. It is very annoying. You have two options. You can use the browser settings and clear the browser cache each time you want to test new JavaScript code in the browser. Like this for Chrome:

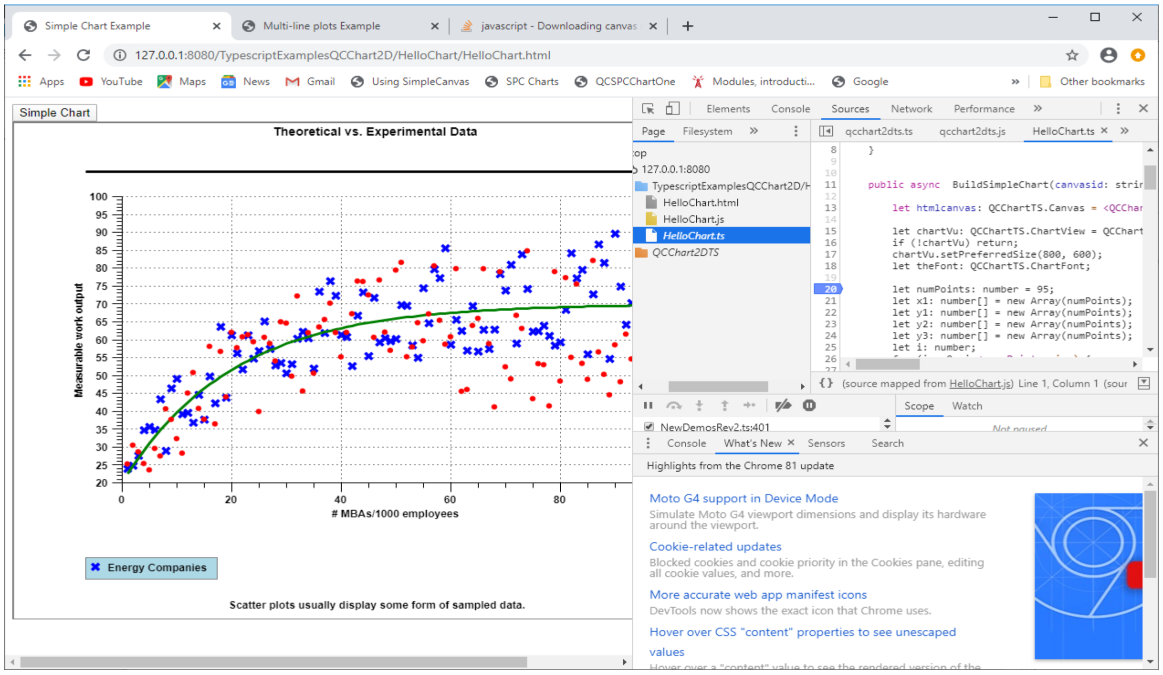


Or, the preferred method, is that you can turn off caching from within the browser Development Mode. Here is a good discussion of that option: https://nicholasbering.ca/tools/2016/10/09/devtools-disable-caching/ . That article discusses the options for Chrome, Firefox and Safari, but Microsoft Edge has a similar option.

Debugging

If for some reason the chart does not appear you will have start debugging.

There are ways to setup Visual Studio Code for integrated debugging. But it looks more complicated than we want to explain. Instead, we choose to just use the integrated F12 debugging available in all of the major browsers. Once you point the browser to the HelloChart.html file you should see the Simple Chart button. Press F12 and the browsers integrated debugger will appear on the right. The debugger portion of the window will look something like:



```
    }


    public async  BuildSimpleChart(canvasid: strin

        let htmlcanvas: QCChartTS.Canvas = <QCChar

        let chartVu: QCChartTS.ChartView = QCChart
        if (!chartVu) return;
        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;
```

Highlights from the Chrome 81 update

Moto G4 support in Device Mode
Simulate Moto G4 viewport dimensions and display its hardware around the viewport.

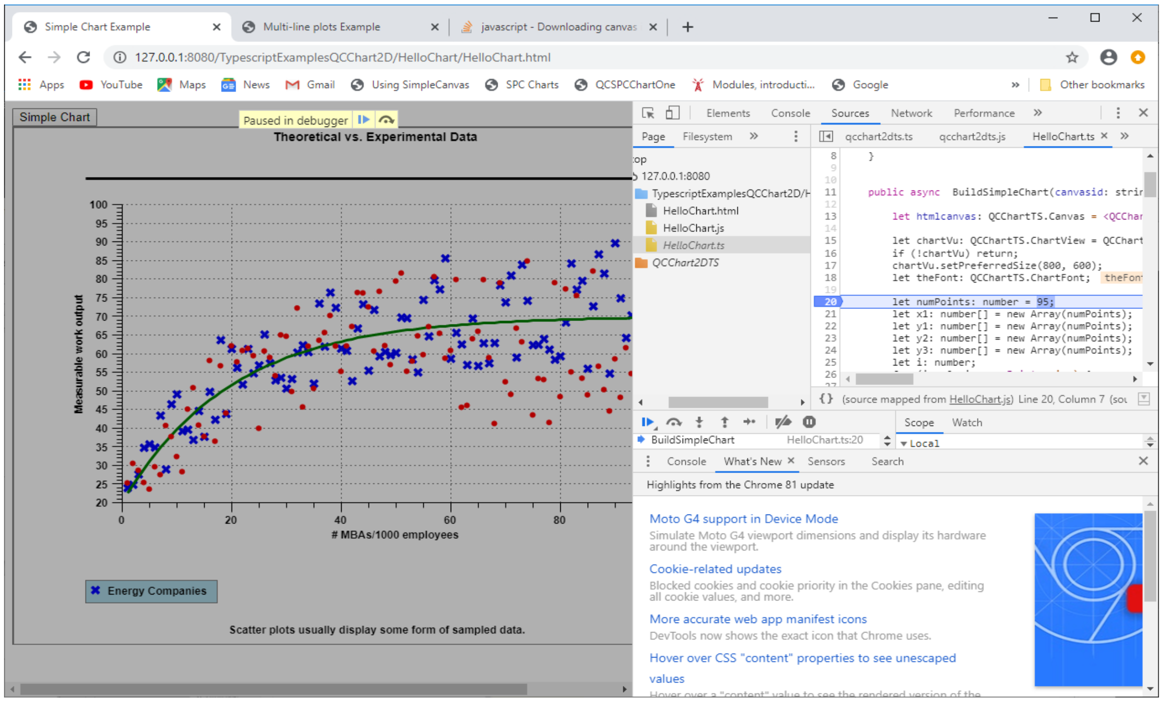
Cookie-related updates
Blocked cookies and cookie priority in the Cookies pane, editing all cookie values, and more.

More accurate web app manifest icons
DevTools now shows the exact icon that Chrome uses.

Hover over CSS "content" properties to see unescaped values

You should be able to select either the HelloChart.ts file, HelloChart.js file, or the HelloChart.html file to view in the main window. It is the presence of the HelloChart.js.map file which lets you debug the TypeScript code, even though you are actually running the JavaScript code.

All of the things you normally want to do with a debugger (view values, step through, set breakpoints,etc.) you can do from this window. So if you set a breakpoint in the HelloChart.ts file, and then click the Simple Chart button in the left hand side of the browser, program execution will stop in the code window when the breakpoint is hit.



```
    }


    public async  BuildSimpleChart(canvasid: strir

        let htmlcanvas: QCChartTS.Canvas = <QCChar

        let chartVu: QCChartTS.ChartView = QCChart
        if (!chartVu) return;
        chartVu.setPreferredSize(800, 600);
        let theFont: QCChartTS.ChartFont;  theFon

        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;
```

(source mapped from HelloChart.js) Line 20, Column 7 (sou

Highlights from the Chrome 81 update

Moto G4 support in Device Mode
Simulate Moto G4 viewport dimensions and display its hardware around the viewport.

Cookie-related updates
Blocked cookies and cookie priority in the Cookies pane, editing all cookie values, and more.

More accurate web app manifest icons
DevTools now shows the exact icon that Chrome uses.

Hover over CSS "content" properties to see unescaped values

Using QCChart2D under Linux (Ubuntu)

We tested the software under a Ubuntu workstation, both in development mode using the Ubuntu version of Visual Studio Code, and as a server, serving up the example program HTML and *.js files. In general, everything seems to work exactly the same.

Important Note: Compared to a Windows server, one thing to be aware of is that the file system of Ubuntu (Linux in general) is case sensitive. So be consistent in the case you use for naming and referencing your files: HTML *.js (JavaScript), and *.ts (TypeScript). Note that main library for QCChart2DTS is spelled in lower case: qcchart2dts.js . So wherever it is referenced it must be lower case. If you are using a Windows server, where the file system is not case sensitive, it doesn't matter what the case of the *.js and *.ts files are.

You can install Visual Studio Code using the Ubuntu Software app found in the main Ubuntu Toolbar. Just search on Visual Studio Code and follow the prompts.

We use the npm http-server as the test sever for our examples, though you can use any test server you want. In order to download and install the npm http-server, you first need to install npm. So go to terminal mode (ctrl-alt-t), and enter:

```
sudo apt install npm
```

Assuming npm installs correctly, install the npm http-server module globally using:

```
sudo npm install http-server -g
```

The installs all need to be handled by sudo (super user do). The previously two steps only need to be done once. The next step will probably need to be done each time you use the workstation as a development computer.

Once http-server is installed, you can start it by going to terminal mode (ctrl-alt-t), if you aren't already there. Navigate to the quinn-curtis/JSTS folder using the cd command. Once there, enter:

```
http-server
```

at the command prompt. The defaults it uses (address = 127.0.0.1 (usually the *localhost* address), and port = 8080) are what our example programs are setup for, so you do not need to change those, unless you are experienced and have a need to change those values. It is very important you start the http-server program from within the quinn-curtis/JSTS folder, because this is the only way the relative file locations we use in the examples work themselves out.

Normally when you are developing you go to the terminal window, start http-server and leave it running while you are working on your code. When done, you close the terminal window, closing the http-server. You would do this each time you work on your code. You can make a Ubuntu script, similar to our startlocalserver.bat file to automate starting of the http-server if you want.

You don't have to use http-server as the test server. You may have another test server, or a real server, you already use. There are ones based on python, php, node, and Ruby among others. Here is a good link describing some of your options: https://askubuntu.com/questions/1102594/how-do-i-set-up-the-simplest-http-local-server

If you are going to use the Visual Studio Code IDE as your development environment, and you plan to program using TypeScript, you will need to install TypeScript on your computer. The example below installs it globally.

```
sudo npm install -g typescript
```

And you don't have to use Visual Studio Code as your development environment. The folder setup of the examples should be usable with any JavaScript or TypeScript editor, where you make the root of your project the same as the example program folder: quinn-curtis/JSTS/TypescriptExamplesQCChart2D/HelloChart for the HelloChart example.

Using QCChart2D in Asp.Net web applications

We were able to use Visual Studio 2019 (VS 2017 is very similar) to create standard Asp.Net applications and add and display QCChart2D charts in each. There are many different variants of Asp.Net and we chose four representative samples for modern web development.

Special Note: We found the portability (transfer from one computer or folder to another) of Asp.Net projects to be limited. Also, even after they are cleaned, traditional Asp.Net projects are huge because of dozens of libraries found in the default packages folder for the project. Because of this, we have removed the two traditional Asp.Net projects (WebFormApplication1 and AspWebAppSinglePageWebApplication1) from the trial and commercial download. But you can download them here: http://quinn-curtis.com/downloadsoftware/TradAspNetChartJSTSProj.zip . Copy the two projects into the Quinn-Curtis\JSTS\AspNetChartVS2019 folder . They will have their own copies of the *qcchart2dts.js* library (found in each project root, under the *QCChart2DTS* folder), which will be the trial version we use in trial downloads of the software. If you have the commercial version of the software, you should copy the qcchart2dts.js file you have in your Quinn-Curtis\JSTS\ QCChart2DTS folder, into the two project folders you just downloaded, so that you are working with commercial version in those examples.

Also, we found that on the initial loading of a project, phantom errors about missing stuff can display in the Errors window. These phantom errors do not keep the project from compiling and running in the test server. If you Clean the project the errors do *not* go away. But if you Clean the solution, exit Visual Studio, restart Visual Studio, and reload the project, the errors will go away. This seems to be associated with moving the project from one computer or folder location to another, and does not happen when you create your own project.

The Asp.Net projects are found in the Quinn-Curtis\JSTS\AspNetChartVS2019 folder.

Project name	**Description**
WebFormApplication1	A basic Asp.Net Form-based web application
AspWebAppSinglePageWebApplication1	A basic Asp.Net Form-based single page application
AspNetCoreWebApplication1	A basic Asp.Net Core 3.0 application
ASPNetCoreMVCWebApplication1	A basic Asp.Net MVC Core 3.0 application

These were created directly from the standard new project templates which ship with VS 2019, so don't blame us if they seem overly complicated. What we did was insert a couple of folders in each project (QCChart2DTS and HelloChart) which hold the qcchart2dts.js library, and the HelloChart example. We then added some JavaScript inside the HTML of one of the various HTML page templates found in the project. And that displays a simple line and scatter chart. In all of the examples, the QCChart2DTS and HelloChart folders are exactly the same. The HelloChart.js example program is slightly different than that found in the JavaScript and TypeScript folders, because they take into account a slightly different directory structure of the Asp.Net web applications. /Also some of the object variables (thePlot1, thePlot2, thePlot3, xAxis, yAxis and chartVu) are made global to the HelloChart.js file so that the InitChartData function can update the chart with new data, without recreating the chart from scratch. Here are some notes on what to do.

Where to place the QCChart2DTS and HelloChart folders

Traditional Asp.Net

In the Asp.Net Fom-based examples (WebFormApplication1 and AspWebAppSinglePageWebApplication1), the root folder of the website is the root folder of the project. So when you run the application, the test server starts the web application in the root folder of the example program. This means that the program cannot go outside of that folder for files, or else it will be a browser security violation. To accommodate that we must place a local copy of our QCChart2DTS library underneath the example program folder, so that the qcchart2dts.js file is accessible to the JavaScript which displays the chart. So you will find the QCChart2DTS and HelloChart folders duplicated under all of the Asp.Net example program. In order to point the example program HelloChart.js to the qcchart2dts.js library, it uses the import statement:

```
import * as  QCChartTS from './../QCChart2DTS/qcchart2dts.js';
```

Rather than place the QCChart2DTS folder in the project root, you could also place it in the *Content* or *Scripts* folder. In that case you would modify the import statement to something like:

```
import * as  QCChartTS from './../Scripts/QCChart2DTS/qcchart2dts.js';
```

Asp.Net Core and Asp.Net MVC

In the Asp.Net Core example programs (AspNetCoreWebApplication1, ASPNetCoreMVCWebApplication1), the root folder of the resulting website is the *wwwroot* folder under the example program folder. Now, the magic of MVC is able to generate web content on the fly and inject it into the wwwroot folder when URLs are used to access that folder. But our software doesn't do that. So, for the time being, you need to place our QCChart2DTS and HelloChart folders underneath the wwwroot folder. We placed the HelloChart folder directly under the wwroot folder, and the QCChart2DTS folder under the *wwwroot/lib* folder. Because if you only place them underneath the project folder, they cannot be accessed, because it would be a security violation to access them outside of the website root folder which is *wwwroot*. In order to point the example program HelloChart.js to the qcchart2dts.js library, it uses the import statement:

```
import * as  QCChartTS from './../lib/QCChart2DTS/qcchart2dts.js';
```

at the top of the HelloChart.js. This is the relative path location, from the HelloChart folder, to the qcchart2dts.js file, in the QCChart2DTS folder. Note the insertion of the /lib folder in the import statement, which makes it different from the import statement in traditional Asp.Net example.

Add a script block which calls to the HelloChart.js file in the appropriate HTML of the project. The script block, some processing buttons, and the HTML5 Canvas container the chart is placed in, consists of the following code:

```
<div class="container">
    <canvas id="chartCanvas1" width="800" height="600"></canvas>
</div>

<script type="module">
    import { BuildSimpleChart, InitChartData } from './HelloChart/HelloChart.js';
    BuildSimpleChart("chartCanvas1");

    function updateData() {

        if (document) {
            var basevalue = Number.parseFloat(document.getElementById("StartValueTextBox").value);
            var count = Number.parseFloat(document.getElementById("CountTextBox").value);

            if ((count > 1) && (count <= 1000000))
             InitChartData(basevalue, count);
        }
        return false;
    }

      document.getElementById("InitDataButton").onclick = updateData;

</script>

<div class="col-md-2">
    <p>
        Start Value: <input id="StartValueTextBox" type="text" value="20" />
    </p>
    <p>
        Count: <input id="CountTextBox" type="text" value="95" />
    </p>
    <p>
        <input id="InitDataButton" type="button" value="Initialize Data" />
    </p>

</div>
```

Project name	**Modified HTML block**
WebFormApplication1	Added to the default.aspx file
AspWebAppSinglePageWebApplication1	Added to the Views/Home/_Home.cshtml file
AspNetCoreWebApplication1	Added to the Pages/Index.cshtml file
ASPNetCoreMVCWebApplication1	Added to the Views/Home/Index.cshtml file

Note: Since HTML script blocks marked as modules are loaded asynchronous with all other elements of the page, you cannot assign an event (onclick) reference to the button, when the button is created in HTML. Instead you must do it inside the script, so that you are sure that the reference to the JavaScript function exists. That is why there is the line

```
      document.getElementById("InitDataButton").onclick = updateData;
```

at the end of the script, assigning the InitChartData.onclick event handler to the updateData function inside the script block.

25. 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 (qcspccharts.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 **setChartObjEnable** 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 <div> 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, described 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.

INDEX