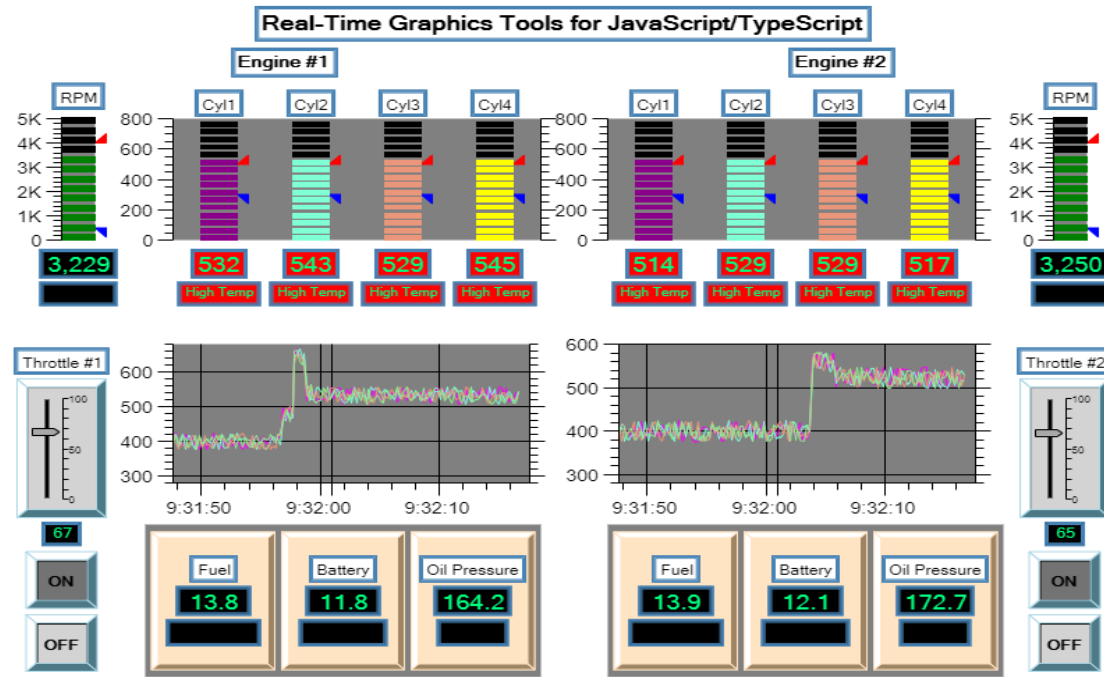


QCRTGraph Real-Time Graphics Tools for JavaScript/TypeScript



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 06/18/2021 Rev. 3.01

Real-Time Graphics Tools for *JavaScript/TypeScript* Documentation and Software Copyright Quinn-Curtis, Inc. 2021

Quinn-Curtis, Inc. Tools for JavaScript/TypeScript END-USER LICENSE AGREEMENT

IMPORTANT-READ CAREFULLY: This Software End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Quinn-Curtis, Inc. for the Quinn-Curtis, Inc. SOFTWARE identified above, which includes all Quinn-Curtis, Inc. *JavaScript/TypeScript* software (on any media) and related documentation (on any media). By installing, copying, or otherwise using the SOFTWARE, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, do not install or use the SOFTWARE. If the SOFTWARE was mailed to you, return the media envelope, UNOPENED, along with the rest of the package to the location where you obtained it within 30 days from purchase.

1. The SOFTWARE is licensed, not sold.

2. GRANT OF LICENSE.

(A) **Developer License.** After you have purchased the license for SOFTWARE, and have received the file containing the licensed copy, you are licensed to copy the SOFTWARE only into the memory of the number of computers corresponding to the number of licenses purchased. The primary user of the computer on which each licensed copy of the SOFTWARE is installed may make a second copy for his or her exclusive use on a portable computer. Under no other circumstances may the SOFTWARE be operated at the same time on more than the number of computers for which you have paid a separate license fee. You may not duplicate the SOFTWARE in whole or in part, except that you may make one copy of the SOFTWARE for backup or archival purposes. You may terminate this license at any time by destroying the original and all copies of the SOFTWARE in whatever form.

(B) **30-Day Trial License.** You may download and use the SOFTWARE without charge on an evaluation basis for thirty (30) days from the day that you DOWNLOAD the trial version of the SOFTWARE. The termination date of the trial SOFTWARE is embedded in the downloaded SOFTWARE and cannot be changed. You must pay the license fee for a Developer License of the SOFTWARE to continue to use the SOFTWARE after the thirty (30) days. If you continue to use the SOFTWARE after the thirty (30) days without paying the license fee you will be using the SOFTWARE on an unlicensed basis.

Redistribution of 30-Day Trial Copy. Bear in mind that the 30-Day Trial version of the SOFTWARE becomes invalid 30-days after downloaded from our web site, or one of our sponsor's web sites. If you wish to redistribute the 30-day trial version of the SOFTWARE you should arrange to have it redistributed directly from our web site. If you are using SOFTWARE on an evaluation basis you may make copies of the evaluation SOFTWARE as you wish; give exact copies of the original evaluation SOFTWARE to anyone; and distribute the evaluation SOFTWARE in its unmodified form via electronic means (Internet, BBS's, Shareware distribution libraries, CD-ROMs, etc.). You may not charge any fee for the copy or use of the evaluation SOFTWARE itself. You must not represent in any way that you are selling the SOFTWARE itself. You must distribute a copy of this EULA with any copy of the SOFTWARE and anyone to whom you distribute the SOFTWARE is subject to this EULA.

(C) **Redistributable License.** The standard Developer License permits the programmer to deploy and/or distribute applications that use the Quinn-Curtis SOFTWARE, royalty free. We cannot allow developers to use this SOFTWARE to create a graphics toolkit (a library or any type of graphics component that will be used in combination with a program development environment) for resale to other developers.

If you utilize the SOFTWARE in an application program, or in a web site deployment, should we ask, you must supply Quinn-Curtis, Inc. with the name of the application program and/or the URL where the SOFTWARE is installed and being used.

3. **RESTRICTIONS.** You may not reverse engineer, de-compile, or disassemble the SOFTWARE, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation. You may not rent, lease, or lend the SOFTWARE. You may not use the SOFTWARE to perform any illegal purpose.

4. **SUPPORT SERVICES.** Quinn-Curtis, Inc. may provide you with support services related to the SOFTWARE. Use of Support Services is governed by the Quinn-Curtis, Inc. policies and programs described in the user manual, in

online documentation, and/or other Quinn-Curtis, Inc.-provided materials, as they may be modified from time to time. Any supplemental SOFTWARE code provided to you as part of the Support Services shall be considered part of the SOFTWARE and subject to the terms and conditions of this EULA. With respect to technical information you provide to Quinn-Curtis, Inc. as part of the Support Services, Quinn-Curtis, Inc. may use such information for its business purposes, including for product support and development. Quinn-Curtis, Inc. will not utilize such technical information in a form that personally identifies you.

5. TERMINATION. Without prejudice to any other rights, Quinn-Curtis, Inc. may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the SOFTWARE.

6. COPYRIGHT. The SOFTWARE is protected by United States copyright law and international treaty provisions. You acknowledge that no title to the intellectual property in the SOFTWARE is transferred to you. You further acknowledge that title and full ownership rights to the SOFTWARE will remain the exclusive property of Quinn-Curtis, Inc. and you will not acquire any rights to the SOFTWARE except as expressly set forth in this license. You agree that any copies of the SOFTWARE will contain the same proprietary notices which appear on and in the SOFTWARE.

7. EXPORT RESTRICTIONS. You agree that you will not export or re-export the SOFTWARE to any country, person, entity, or end user subject to U.S.A. export restrictions. Restricted countries currently include, but are not necessarily limited to Cuba, Iran, Iraq, Libya, North Korea, Sudan, and Syria. You warrant and represent that neither the U.S.A. Bureau of Export Administration nor any other federal agency has suspended, revoked or denied your export privileges.

8. NO WARRANTIES. Quinn-Curtis, Inc. expressly disclaims any warranty for the SOFTWARE. THE SOFTWARE AND ANY RELATED DOCUMENTATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OR MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE REMAINS WITH YOU.

9. LIMITATION OF LIABILITY. IN NO EVENT SHALL QUINN-CURTIS, INC. OR ITS SUPPLIERS BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES OF ANY KIND ARISING OUT OF THE DELIVERY, PERFORMANCE, OR USE OF THE SUCH DAMAGES. IN ANY EVENT, QUINN-CURTIS'S LIABILITY FOR ANY CLAIM, WHETHER IN CONTRACT, TORT, OR ANY OTHER THEORY OF LIABILITY WILL NOT EXCEED THE GREATER OF U.S. \$1.00 OR LICENSE FEE PAID BY YOU.

10. U.S. GOVERNMENT RESTRICTED RIGHTS. The SOFTWARE is provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer SOFTWARE clause of DFARS 252.227-7013 or subparagraphs (c)(i) and (2) of the Commercial Computer SOFTWARE- Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is: Quinn-Curtis, Inc., 18 Hearthstone Dr., Medfield MA 02052 USA.

11. MISCELLANEOUS. If you acquired the SOFTWARE in the United States, this EULA is governed by the laws of the state of Massachusetts. If you acquired the SOFTWARE outside of the United States, then local laws may apply.

Should you have any questions concerning this EULA, or if you desire to contact Quinn-Curtis, Inc. for any reason, please contact Quinn-Curtis, Inc. by mail at: Quinn-Curtis, Inc., 18 Hearthstone Dr., Medfield MA 02052 USA, or by telephone at: (508)359-6639, or by electronic mail at: support@Quinn-Curtis.com.

Table of Contents

1. Introduction.....	1
Real-Time Graphics Tools for <i>JavaScript/TypeScript</i> Background.....	1
Transpiling vs Compiling.....	2
HTML5.....	2
JavaScript.....	3
TypeScript as a Productivity Tool.....	3
Visual Studio Code.....	4
Using QCRTGraph under Linux (Ubuntu).....	4
Programming QCRTGraph directly from JavaScript.....	6
Programing QCRTGraph using TypeScript.....	15
Programing QCRTGraph within Wordpress.....	23
Programing QCRTGraph using the Angular framework.....	23
Directory Structure of QCRTGraph for JavaScript/TypeScript.....	26
Versions of QCRTGraphTS.....	27
Chapter Summary.....	28
2. Class Architecture of the Real-Time Graphics Tools for <i>JavaScript/TypeScript</i> Class Library.....	30
Major Design Considerations.....	30
Real-Time Graphics Tools for <i>JavaScript/TypeScript</i> Class Summary.....	32
Namespace	35
QCRTGraph Classes.....	36
Process Variable and Alarms Classes.....	36
Panel Meter Classes.....	37
Single Value Indicators.....	39
Multiple Value Indicators.....	45
Alarm Indicator Classes.....	49
Meter Axis Classes.....	50
Form Control Classes.....	52
Scroll Frame Class.....	54
Miscellaneous Classes.....	57
3. QCChart2D for <i>JavaScript/TypeScript</i> Class Summary.....	61
QCChart2D for <i>JavaScript/TypeScript</i> Class Summary.....	61
Chart Window Classes.....	62
Data Classes.....	62
Scale Classes.....	63
Coordinate Transform Classes.....	63
Auto-Scaling Classes.....	65
Chart Object Classes.....	66
Mouse Interaction Classes.....	91
File and Printer Rendering Classes.....	93
Miscellaneous Utility Classes.....	93
4. Process Variable and Alarm Classes.....	97
Real-Time Process Variable.....	97

Real-Time Alarms.....	101
Real-Time Alarms Event Handling.....	104
5. Panel Meter Classes.....	108
LED/LCD Style Font.....	109
Panel Meters.....	110
Numeric Panel Meter.....	115
Alarm Panel Meter.....	119
String Panel Meter.....	122
Time/Date Panel Meter.....	124
Form Control Panel Meter.....	129
6. Single Channel Bar Indicator.....	133
Bar Indicator.....	134
7. Multiple Channel Bar Indicator.....	146
Multiple Channel Bar Indicator.....	146
8. Meters Coordinates, Meter Axes and Meter Axis Labels.....	158
Meter Coordinates.....	158
Meter Axis.....	162
Numeric Meter Axis Labels.....	167
String Meter Axis Labels.....	171
9. Meter Indicators: Needle, Arc and Symbol.....	175
Meter Indicators.....	176
Arc Meter Indicator.....	177
Needle Meter Indicator.....	186
Symbol Meter Indicators.....	191
10. Dials and Clocks.....	197
Converting Dial and Clock Data using RTComboProcessVar.....	197
11. Single and Multiple Channel Annunciators.....	207
Single Channel Annunciator.....	208
Multi-Channel Annunciators.....	210
12. The Scroll Frame and Single Channel Scrolling Plots.....	217
Scroll Frame.....	218
Single Channel Scrolling Graphs.....	224
13. Multi-Channel Scrolling Plots.....	233
Multi-Channel Scrolling Graphs.....	234
14. Buttons, Track Bars and Other Form Control Classes.....	241
Control Buttons.....	243
Control TrackBars.....	249
Form Control Panel Meter.....	255
Form Control Grid.....	258
15. PID Control.....	263
Implementation.....	266
PID Control.....	266
16. Zooming Real-Time Data.....	272
Simple Zooming of a single channel scroll frame.....	273

Super Zooming of multiple physical coordinate systems.....	276
Limiting the Zoom Range.....	279
17. Miscellaneous Shape Drawing.....	280
3D Borders and Background Frames.....	280
Rounded Rectangles.....	284
General Shapes.....	286
18. Process Variable Viewer.....	289
Copying the Canvas Image.....	296
20. Using Real-Time Graphics Tools for <i>JavaScript/TypeScript</i> to Create Web Applications.....	299
Visual Studio Code.....	301
Using JavaScript Only.....	301
Using TypeScript.....	315
Using QCRTGraph under Linux (Ubuntu).....	332
Using QCRTGraph in Asp.Net web applications.....	333

Real-Time Graphics Tools for *JavaScript/TypeScript*

1. Introduction

QCRTGraph for JavaScript/TypeScript (JSTS-RTG-DEVR)

Real-Time Graphics Tools for *JavaScript/TypeScript* Background

A large subcategory of the charting software market is concerned with the continuous or on-demand update of real-time data in a scrolling chart, gauge (bar graph), meter, annunciator or text format. Software that creates graphs of this type should make the creation and update of real-time graphs as simple and as fast as possible. The original **QCChart2D** charting product was designed to allow for the fast creation and update of custom charts using a large number of auto-scale, auto-axes, and auto-labeling routines. A good application for the **QCChart2D** software is the on-demand creation and display of historical stock market data, where the data source, time frame and scale are defined by user inputs. This is the type of charting application that you will find on Yahoo, MSN and every brokerage firm web site. A related application would involve the second by second update of real-time stock market data as it streams from a real-time data source. The software that is used for the display of historical data is seldom used to display real-time data, because its data structures are not designed for incremental updates, and its rendering routines are not fast enough to convert the data to a chart within the allowable display update interval. The **Real-Time Graphics Tools for JavaScript/TypeScript** integrates the **QCChart2D** charting software with real-time data structures and specialized rendering routines. It is designed for on-the-fly rendering of the real-time data using new specialized classes for scrolling graphs, gauges (bar graphs), meters, annunciators and text. Plot objects created using the **QCChart2D** classes can be freely mixed with the new **Real-Time Graphics Tools** classes. Advanced user interface features such as zooming and tool-tips can be used with real-time scrolling charts.

Like the **QCChart2D** software, the **Real-Time Graphics Tools for JavaScript/TypeScript** uses the graphics features found in the HTML5 canvas graphics API. These include:

- Arbitrary line thickness and line styles for all lines.
- Gradients, and color transparency for solid objects.
- Generalized geometry support used to create arbitrary shapes
- Compatible with browser printer options
- Font support for a large number of fonts, using a variety of font styles, size and rotation attributes.
- Imaging support for a large number of image formats
- Advanced matrix support for handling 2D transformations.

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 level 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 20 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 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 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 a 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 QCRTGraph 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,

4 Introduction

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 you 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 QCRTGraph 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 Linux 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 qcrthgraphs is spelled in lower case: qcrthgraphs.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
```

6 Introduction

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/TypescriptExamplesQCRTGraph/HelloRTScroll for the HelloRTScroll example.

Programming QCRTGraph directly from JavaScript

The goal of this software is to make programming the QCRTGraph library from JavaScript just as easy, powerful, and flexible, as it from C# and Java (using other versions of QCRTGraph). 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 HelloRTScroll example found in the Quinn-Curtis/JSTS/JavascriptExamplesQCRTGraph/ folder. All of the JavaScript programming examples are found in the Quinn-Curtis/JSTS/JavascriptExamplesQCRTGraph/ examples folder.

```
import * as QCRTGraphTS from '../..QCRTGraphTS/qcrtgraphts.js';

var timerId = 0;
var timercount = 0;
var timercountlimit = 500;
var chartVu = null;
var currentTemperatureValue1 = 110;
var currentTemperature1 = null;
var currentTemperatureValue2 = 100;
var currentTemperature2 = null;
var scrollFrame = new QCRTGraphTS.RTScrollFrame();
var templowalarm1 = null;
var temphighalarm1 = null;
var templowalarm2 = null;
var temphighalarm2 = null;
var font12 = QCRTGraphTS.ChartFont.newChartFont3("Arial", QCRTGraphTS.ChartFont.REGULAR, 12);
var font14 = QCRTGraphTS.ChartFont.newChartFont3("Arial", QCRTGraphTS.ChartFont.REGULAR, 14);
var font14Numeric = QCRTGraphTS.ChartFont.newChartFont3("Arial",
QCRTGraphTS.ChartFont.REGULAR, 14);

    async function TimerTickProc() {
if (!currentTemperature1 || !currentTemperature2)
    return;
if (chartVu) {
    // Random data
    currentTemperatureValue1 += 5 * (0.5 - QCRTGraphTS.ChartSupport.getRandomDouble());
    currentTemperatureValue2 += 8 * (0.5 - QCRTGraphTS.ChartSupport.getRandomDouble());
    // This method uses the default time stamp, which is the current time-of-day
    currentTemperature1.setCurrentValueNumber(currentTemperatureValue1);
    currentTemperature2.setCurrentValueNumber(currentTemperatureValue2);
    chartVu.updatedDraw();
}
timercount++;
if (timercount > timercountlimit)
    StopTimer();
}
    async function StartTimer(milliseconds, count) {
if (timerId != 0)
    StopTimer();
timercountlimit = count;
timerId = setInterval(() => TimerTickProc(), milliseconds);
}
    async function StopTimer() {
clearInterval(timerId);
timerId = 0;
}
    export async function BuildRTScroll(canvasid) {
let htmlcanvas = document.getElementById(canvasid);
```

```

chartVu = QCRTGraphTS.ChartView.newChartView(htmlcanvas);
if (!chartVu) return;
chartVu.setPreferredSize(800, 600);
let x1 = 0.1;
let y1 = 0.1;
let x2 = 0.9;
let y2 = 0.9;
let axisFont = font12;
let startTime = new Date();
let endTime = new Date();
QCRTGraphTS.ChartCalendar.add(endTime, QCRTGraphTS.ChartConstants.SECOND, 60);
InitializeProcessVar();
if (!currentTemperature1 || !currentTemperature2)
    return;
let pTransform1 = QCRTGraphTS.TimeCoordinates.newTimeCoordinatesTime(startTime,
currentTemperature1.DefaultMinimumDisplayValue, endTime,
currentTemperature1.DefaultMaximumDisplayValue);
pTransform1.setGraphBorderDiagonal(x1, y1, x2, y2);
let graphbackground = QCRTGraphTS.Background.newBackground(pTransform1,
QCRTGraphTS.ChartConstants.GRAPH_BACKGROUND, QCRTGraphTS.ChartColor.WHITE);
chartVu.addChartObject(graphbackground);
let plotbackground = QCRTGraphTS.Background.newBackground(pTransform1,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.BLACK);
chartVu.addChartObject(plotbackground);
let xaxis = QCRTGraphTS.TimeAxis.newTimeAxisType(pTransform1,
QCRTGraphTS.ChartConstants.X_AXIS);
xaxis.LineColor = QCRTGraphTS.ChartColor.BLACK;
chartVu.addChartObject(xaxis);
let yaxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
QCRTGraphTS.ChartConstants.Y_AXIS);
yaxis.LineColor = QCRTGraphTS.ChartColor.BLACK;
chartVu.addChartObject(yaxis);
let yAxisGrid = QCRTGraphTS.Grid.newGrid(xaxis, yaxis, QCRTGraphTS.ChartConstants.Y_AXIS,
QCRTGraphTS.ChartConstants.GRID_MAJOR);
yAxisGrid.setColor(QCRTGraphTS.ChartColor.WHITE);
yAxisGrid.setLineWidth(1);
yAxisGrid.setLineStyle(QCRTGraphTS.ChartConstants.LS_DOT_1_4);
chartVu.addChartObject(yAxisGrid);
let xAxisGrid = QCRTGraphTS.Grid.newGrid(xaxis, yaxis, QCRTGraphTS.ChartConstants.X_AXIS,
QCRTGraphTS.ChartConstants.GRID_MAJOR);
xAxisGrid.setColor(QCRTGraphTS.ChartColor.WHITE);
xAxisGrid.setLineWidth(1);
xAxisGrid.setLineStyle(QCRTGraphTS.ChartConstants.LS_DOT_1_4);
chartVu.addChartObject(xAxisGrid);
let yaxis2 = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
QCRTGraphTS.ChartConstants.Y_AXIS);
yaxis2.LineColor = QCRTGraphTS.ChartColor.BLACK;
yaxis2.AxisTickDir = QCRTGraphTS.ChartConstants.AXIS_MAX;
yaxis2.AxisIntercept = xaxis.AxisMax;
chartVu.addChartObject(yaxis2);
let xAxisLab = QCRTGraphTS.TimeAxisLabels.newTimeAxisLabels(xaxis);
xAxisLab.TextFont = axisFont;
chartVu.addChartObject(xAxisLab);
let yAxisLab = QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(yaxis);
yAxisLab.TextFont = axisFont;
chartVu.addChartObject(yAxisLab);
scrollFrame =
QCRTGraphTS.RTScrollFrame.newRTScrollFrameChartViewProcessVarCoordsScrollMode(chartVu,
currentTemperature1, pTransform1,
QCRTGraphTS.ChartConstants.RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL);
scrollFrame.addProcessVar(currentTemperature2);
scrollFrame.ScrollScaleModeY = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_Y_MINMAX;
// Allow 100 samples to accumulate before autoscaling y-axis. This prevents rapid
// changes of the y-scale for the first few samples
scrollFrame.MinSamplesForAutoScale = 100;
scrollFrame.ScrollRescaleMargin = 0.01;
chartVu.addChartObject(scrollFrame);
let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.YELLOW, 3,
QCRTGraphTS.ChartConstants.LS_SOLID);
let lineplot1 = QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(pTransform1,
attrib1);

```

8 Introduction

```
    lineplot1.setFastClipMode(QCARTGraphTS.ChartConstants.FASTCLIP_X);
    let solarPanellinePlot1 =
QCARTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform1, lineplot1,
currentTemperature1);
    chartVu.addChartObject(solarPanellinePlot1);
    let attrib2 = QCARTGraphTS.ChartAttribute.newChartAttribute3(QCARTGraphTS.ChartColor.GREEN, 3,
QCARTGraphTS.ChartConstants.LS_SOLID);
    let lineplot2 = QCARTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(pTransform1,
attrib2);
    lineplot2.setFastClipMode(QCARTGraphTS.ChartConstants.FASTCLIP_X);
    let solarPanellinePlot2 =
QCARTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform1, lineplot2,
currentTemperature2);
    chartVu.addChartObject(solarPanellinePlot2);
    let alarmlines = QCARTGraphTS.RTAlarmIndicator.newRTAlarmIndicatorAxisRTPlot(yaxis,
solarPanellinePlot1);
    alarmlines.AlarmIndicatorMode = QCARTGraphTS.ChartConstants.RT_ALARM_LIMIT_LINE_INDICATOR;
    // Auto-scale can move alarm lines outside of plot area, clip the lines in this case.
    alarmlines.ChartObjClipping = QCARTGraphTS.ChartConstants.PLOT_AREA_CLIPPING;
    chartVu.addChartObject(alarmlines);
    let titlefont = font14;
    let charttitle = QCARTGraphTS.ChartTitle.newChartTitle(pTransform1, titlefont, "Scroll
Application #1", QCARTGraphTS.ChartConstants.CHART_HEADER,
QCARTGraphTS.ChartConstants.CENTER_PLOT);
    chartVu.addChartObject(charttitle);
    chartVu.updateDraw();
    StartTimer(100, 10000);
  }
  async function InitializeProcessVar() {
    temlowalarm1 =
QCARTGraphTS.RTAlarm.newRTAlarmAlarmType(QCARTGraphTS.ChartConstants.RT_ALARM_LOWER_THAN, 80);
    temlowalarm1.AlarmMessage = "Low Alarm";
    temlowalarm1.AlarmSymbolColor = QCARTGraphTS.ChartColor.BLUE;
    temlowalarm1.AlarmTextColor = QCARTGraphTS.ChartColor.BLUE;
    temphighalarm1 =
QCARTGraphTS.RTAlarm.newRTAlarmAlarmType(QCARTGraphTS.ChartConstants.RT_ALARM_GREATER_THAN, 120);
    temphighalarm1.AlarmMessage = "High Alarm";
    temphighalarm1.AlarmSymbolColor = QCARTGraphTS.ChartColor.RED;
    temphighalarm1.AlarmTextColor = QCARTGraphTS.ChartColor.RED;
    currentTemperature1 = QCARTGraphTS.RTProcessVar.newRTProcessVar("Temp 1",
QCARTGraphTS.ChartAttribute.newChartAttribute4(QCARTGraphTS.ChartColor.GREEN, 1,
QCARTGraphTS.ChartConstants.LS_SOLID, QCARTGraphTS.ChartColor.GREEN));
    currentTemperature1.MinimumValue = -20;
    currentTemperature1.MaximumValue = 200;
    currentTemperature1.DefaultMinimumDisplayValue = 0;
    currentTemperature1.DefaultMaximumDisplayValue = 150;
    currentTemperature1.setCurrentValueNumber(currentTemperatureValue1);
    currentTemperature1.addAlarm(temlowalarm1);
    currentTemperature1.addAlarm(temphighalarm1);
    // Important, enables historical data collection for scroll graphs
    currentTemperature1.DatasetEnableUpdate = true;
    temlowalarm2 =
QCARTGraphTS.RTAlarm.newRTAlarmAlarmType(QCARTGraphTS.ChartConstants.RT_ALARM_LOWER_THAN, 80);
    temlowalarm2.AlarmMessage = "Low Alarm";
    temlowalarm2.AlarmSymbolColor = QCARTGraphTS.ChartColor.BLUE;
    temlowalarm2.AlarmTextColor = QCARTGraphTS.ChartColor.BLUE;
    temphighalarm2 =
QCARTGraphTS.RTAlarm.newRTAlarmAlarmType(QCARTGraphTS.ChartConstants.RT_ALARM_GREATER_THAN, 120);
    temphighalarm2.AlarmMessage = "High Alarm";
    temphighalarm2.AlarmSymbolColor = QCARTGraphTS.ChartColor.RED;
    temphighalarm2.AlarmTextColor = QCARTGraphTS.ChartColor.RED;
    currentTemperature2 = QCARTGraphTS.RTProcessVar.newRTProcessVar("Temp 2",
QCARTGraphTS.ChartAttribute.newChartAttribute4(QCARTGraphTS.ChartColor.GREEN, 1,
QCARTGraphTS.ChartConstants.LS_SOLID, QCARTGraphTS.ChartColor.GREEN));
    currentTemperature2.MinimumValue = -20;
    currentTemperature2.MaximumValue = 200;
    currentTemperature2.DefaultMinimumDisplayValue = 0;
    currentTemperature2.DefaultMaximumDisplayValue = 150;
    currentTemperature2.setCurrentValueNumber(currentTemperatureValue2);
    currentTemperature2.addAlarm(temlowalarm2);
    currentTemperature2.addAlarm(temphighalarm2);
```

```
// Important, enables historical data collection for scroll graphs
currentTemperature2.DatasetEnableUpdate = true;
}
```

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

```
import * as QCRTGraphTS from '../../QCRTGraphTS/qcrgraphts.js';
```

The “../../QCRTGraphTS” folder specification means to back up two directory levels and go to the QCRTGraphTS folder, where the qcrgraphts.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/JavascriptExamplesQCRTGraph>HelloRTScroll>HelloRTScroll.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/JavascriptExamplesQCRTGraph>HelloRTScroll>HelloRTScroll.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 QCRTGraphTS** statement, all JavaScript objects found in the library are located in the namespace QCRTGraphTS, which prevents them from conflicting with any other libraries you may be using which have similar named objects. The file qcrgraphts.js contains the entire

10 Introduction

QCRTGraph library. It also contains the complete QCChart2D library because that is what QCRTGraph is built on.

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

```
chartVu = QCRTGraphTS.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 container.

Our original QCRTGraph software made extensive use of overloaded constructors and function names. But JavaScript/TypeScript 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 =  
QCRTGraphTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCRTGraphTS.ChartConstants.LINEAR  
_SCALE, QCRTGraphTS.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 declarations **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> ..

In general, the QCRTGraph library plots data contained in RTProcessVar objects. So the first thing you need to do is create one of the objects for every variable you want to plot. In the HelloRTScroll example, to RTProcessVar objects are initialized, currentTemperature1 and currentTemperature2.

```
async function InitializeProcessVar() {  
  
templowalarm1 = QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_LOWER  
THAN, 80);  
templowalarm1.AlarmMessage = "Low Alarm";  
templowalarm1.AlarmSymbolColor = QCRTGraphTS.ChartColor.BLUE;  
templowalarm1.AlarmTextColor = QCRTGraphTS.ChartColor.BLUE;
```



```

temphighalarm1 = QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_GREATER_THAN, 120);
    temphighalarm1.AlarmMessage = "High Alarm";
    temphighalarm1.AlarmSymbolColor = QCRTGraphTS.ChartColor.RED;
    temphighalarm1.AlarmTextColor = QCRTGraphTS.ChartColor.RED;

currentTemperature1 = QCRTGraphTS.RTProcessVar.newRTProcessVar("Temp 1", QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.GREEN, 1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN));
    currentTemperature1.MinimumValue = -20;
    currentTemperature1.MaximumValue = 200;
    currentTemperature1.DefaultMinimumDisplayValue = 0;
    currentTemperature1.DefaultMaximumDisplayValue = 150;
    currentTemperature1.setCurrentValueNumber(currentTemperatureValue1);
    currentTemperature1.addAlarm(templowalarm1);
    currentTemperature1.addAlarm(temphighalarm1);
    // Important, enables historical data collection for scroll graphs
    currentTemperature1.DatasetEnableUpdate = true;

templowalarm2 = QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_LOWER_THAN, 80);
    templowalarm2.AlarmMessage = "Low Alarm";
    templowalarm2.AlarmSymbolColor = QCRTGraphTS.ChartColor.BLUE;
    templowalarm2.AlarmTextColor = QCRTGraphTS.ChartColor.BLUE;

temphighalarm2 = QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_GREATER_THAN, 120);
    temphighalarm2.AlarmMessage = "High Alarm";
    temphighalarm2.AlarmSymbolColor = QCRTGraphTS.ChartColor.RED;
    temphighalarm2.AlarmTextColor = QCRTGraphTS.ChartColor.RED;

currentTemperature2 = QCRTGraphTS.RTProcessVar.newRTProcessVar("Temp 2", QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.GREEN, 1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN));
    currentTemperature2.MinimumValue = -20;
    currentTemperature2.MaximumValue = 200;
    currentTemperature2.DefaultMinimumDisplayValue = 0;
    currentTemperature2.DefaultMaximumDisplayValue = 150;
    currentTemperature2.setCurrentValueNumber(currentTemperatureValue2);
    currentTemperature2.addAlarm(templowalarm2);
    currentTemperature2.addAlarm(temphighalarm2);
    // Important, enables historical data collection for scroll graphs
    currentTemperature2.DatasetEnableUpdate = true;
}

```

Note that all constants used in the setup, and all QCRTGraph types in general, need to be prefixed by the QCRTGraphTS 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 QCRTGraph 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 QCRTGraphTS 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. The namespace you designate for the QCRTGraphTS library also applies to all of the QCChart2D routines and constants within, because the entire QCChart2D library is included within the QCRTGraphTS.js library file you are importing.

Next, the chart which will display the real-time scrolling graphs is created. First, display aspects of the graph are defined by creating chart and plot objects, and adding them to the parent **ChartView**

12 Introduction

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 (**TimeAxis** and **LinearAxis**), x- and y-axes labels (**TimeAxisLabels** and **NumericAxisLabels**), and axis grids (**Grid**).

[JavaScript]

```
let pTransform1 = QCRTGraphTS.TimeCoordinates.newTimeCoordinatesTime(startTime, currentTemperature1.DefaultMinimumDisplayValue, endTime, currentTemperature1.DefaultMaximumDisplayValue);
pTransform1.setGraphBorderDiagonal(x1, y1, x2, y2);
let graphbackground = QCRTGraphTS.Background.newBackground(pTransform1, QCRTGraphTS.ChartConstants.GRAPH_BACKGROUND, QCRTGraphTS.ChartColor.WHITE);
chartVu.addChartObject(graphbackground);
let plotbackground = QCRTGraphTS.Background.newBackground(pTransform1, QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.BLACK);
chartVu.addChartObject(plotbackground);
let xaxis = QCRTGraphTS.TimeAxis.newTimeAxisType(pTransform1, QCRTGraphTS.ChartConstants.X_AXIS);
xaxis.LineColor = QCRTGraphTS.ChartColor.BLACK;
chartVu.addChartObject(xaxis);
let yaxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1, QCRTGraphTS.ChartConstants.Y_AXIS);
yaxis.LineColor = QCRTGraphTS.ChartColor.BLACK;
chartVu.addChartObject(yaxis);
let yAxisGrid = QCRTGraphTS.Grid.newGrid(xaxis, yaxis, QCRTGraphTS.ChartConstants.Y_AXIS, QCRTGraphTS.ChartConstants.GRID_MAJOR);
yAxisGrid.setColor(QCRTGraphTS.ChartColor.WHITE);
yAxisGrid.setLineWidth(1);
yAxisGrid.setLineStyle(QCRTGraphTS.ChartConstants.LS_DOT_1_4);
chartVu.addChartObject(yAxisGrid);
let xAxisGrid = QCRTGraphTS.Grid.newGrid(xaxis, yaxis, QCRTGraphTS.ChartConstants.X_AXIS, QCRTGraphTS.ChartConstants.GRID_MAJOR);
xAxisGrid.setColor(QCRTGraphTS.ChartColor.WHITE);
xAxisGrid.setLineWidth(1);
xAxisGrid.setLineStyle(QCRTGraphTS.ChartConstants.LS_DOT_1_4);
chartVu.addChartObject(xAxisGrid);
let yaxis2 = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1, QCRTGraphTS.ChartConstants.Y_AXIS);
yaxis2.LineColor = QCRTGraphTS.ChartColor.BLACK;
yaxis2.AxisTickDir = QCRTGraphTS.ChartConstants.AXIS_MAX;
yaxis2.AxisIntercept = xaxis.AxisMax;
chartVu.addChartObject(yaxis2);
let xAxisLab = QCRTGraphTS.TimeAxisLabels.newTimeAxisLabels(xaxis);
xAxisLab.TextFont = axisFont;
chartVu.addChartObject(xAxisLab);
let yAxisLab = QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(yaxis);
yAxisLab.TextFont = axisFont;
chartVu.addChartObject(yAxisLab);
```

Next, the scrolling part of the graphs is created, starting with the **RTScrollFrame**, which controls the -y-ling of the graphs as data values are added to the **RTProcessVar** objects attached to the graph.

```
scrollFrame = QCRTGraphTS.RTScrollFrame.newRTScrollFrameChartViewProcessVarCoordsScrollMode(chartVu, currentTemperature1, pTransform1, QCRTGraphTS.ChartConstants.RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL);
scrollFrame.addProcessVar(currentTemperature2);
scrollFrame.ScrollScaleModeY = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_Y_MINMAX;
// Allow 100 samples to accumulate before autoscaling y-axis. This prevents rapid
// changes of the y-scale for the first few samples
scrollFrame.MinSamplesForAutoScale = 100;
scrollFrame.ScrollRescaleMargin = 0.01;
chartVu.addChartObject(scrollFrame);
```

```

let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.YELLOW, 3, QCRTGraphTS.ChartConstants.LS_SOLID);
let lineplot1 = QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(pTransform1, attrib1);
lineplot1.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
let solarPanelLinePlot1 = QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform1, lineplot1, currentTemperature1);
chartVu.addChartObject(solarPanelLinePlot1);
let attrib2 = QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.GREEN, 3, QCRTGraphTS.ChartConstants.LS_SOLID);
let lineplot2 = QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(pTransform1, attrib2);
lineplot2.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
let solarPanelLinePlot2 = QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform1, lineplot2, currentTemperature2);
chartVu.addChartObject(solarPanelLinePlot2);
let alarmlines = QCRTGraphTS.RTAlarmIndicator.newRTAlarmIndicatorAxisRTPlot(yaxis, solarPanelLinePlot1);
alarmlines.AlarmIndicatorMode = QCRTGraphTS.ChartConstants.RT_ALARM_LIMIT_LINE_INDICATOR;
// Auto-scale can move alarm lines outside of plot area, clip the lines in this case.
alarmlines.ChartObjClipping = QCRTGraphTS.ChartConstants.PLOT_AREA_CLIPPING;
chartVu.addChartObject(alarmlines);

```

And lastly, a local timer is used to update chart once a second. First, the two `RTProcessVar` objects, `currentTemperature1` and `currentTemperature2` are updated using simulated data. And second, the chart is refreshed to show the new data by calling the `updateDraw` method of the parent `ChartView` objects `chartVu` in the example below.

[JavaScript]

```

async function TimerTickProc() {
  if (!currentTemperature1 || !currentTemperature2)
    return;
  if (chartVu) {
    // Random data
    currentTemperatureValue1 += 5 * (0.5 - QCRTGraphTS.ChartSupport.getRandomDouble());
    currentTemperatureValue2 += 8 * (0.5 - QCRTGraphTS.ChartSupport.getRandomDouble());
    // This method uses the default time stamp, which is the current time-of-day
    currentTemperature1.setCurrentValueNumber(currentTemperatureValue1);
    currentTemperature2.setCurrentValueNumber(currentTemperatureValue2);
    chartVu.updateDraw();
  }
  timercount++;
  if (timercount > timercountlimit)
    StopTimer();
}

```

In it you will find that simulated data is created and then passed into one of our `RTProcessVar` classes. The real-time plotting routines reference the `RTProcessVar` classes and the real-time plots get updated when the chart is refreshed using the `chartVu.updateDraw` method.

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

14 Introduction

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. 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 RT Scroll Example</title>
</head>
<body>

  <div id="buttondiv">
    <button type="button" id="rtScroll_menuitem">RT Scroll</button>
  </div>

  <div id="canvasdiv">
    <canvas id="rtScrollCanvas1" width="800" height="600"></canvas>
  </div>
  <script type="module">

import { BuildRTScroll} from './HelloRTScroll.js';

    function rtScroll() {
      BuildRTScroll("rtScrollCanvas1");
    }

    // Since loading of script module is asynchronous, need to assign onclick event handler
s after module loaded.
    document.getElementById("rtScroll_menuitem").onclick = rtScroll;

  </script>

  <br>
  <br>
</body>
</html>
```

In this example the button click for the RT Scroll button invokes the JavaScript *rtScroll* function call inside the script section. That function in turn calls the **BuildRTScroll** in the imported `HelloRTScroll.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="rtScrollCanvas1" width="800" height="600"></canvas>
</div>
```

You can give the canvas any id that you want, but this needs to be passed into the JavaScript work file (`HelloRTScroll.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: QCRTGraphTS.Canvas = <QCRTGraphTS.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 BuildRTScroll function from the HelloRTScroll.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 HelloRTScroll function in the script block has not yet been loaded. Instead is it assigned explicitly in the javascript code, after the HelloRTScroll.js module has been loaded.

```
document.getElementById("rtScroll_menuitem").onclick = rtScroll;
```

You can read more about the ES6 module loading specification on the web:

<https://www.sitepoint.com/using-es-modules/> .

Programing QCRTGraph 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 September. 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/TypescriptExamplesQCRTGraph folder. Each example folder contains an HTML file to load into a browser to test the example, a *.ts file which loads the qcrtgraph.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 (HelloRTScroll) 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,
```

16 Introduction

```
    "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 qcrgraphs.js library was developed using this option.

moduleResolution: node - Specifies module resolution strategy: 'node' (Node) or 'classic' (TypeScript pre 1.6) .

exclude: [- 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.

node_modules
]

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 (HelloRTScroll) is defined with a BuildRTScroll member function.

```
import * as QCRTGraphTS from '../..//QCRTGraphTS/qcrtgraphts.js';

export class HelloRTScroll {

    timerId: number = 0;
    timercount: number = 0;
    timercountlimit: number = 500;
    chartVu: QCRTGraphTS.ChartView | null = null;

    currentTemperatureValue1: number = 110;
    currentTemperature1: QCRTGraphTS.RTProcessVar | null = null;;
    currentTemperatureValue2: number = 100;
    currentTemperature2: QCRTGraphTS.RTProcessVar | null = null;;

    scrollFrame: QCRTGraphTS.RTScrollFrame = new QCRTGraphTS.RTScrollFrame();

    templowalarm1: QCRTGraphTS.RTAlarm | null = null;
    temphighalarm1: QCRTGraphTS.RTAlarm | null = null;;
    templowalarm2: QCRTGraphTS.RTAlarm | null = null;;
    temphighalarm2: QCRTGraphTS.RTAlarm | null = null;;

    font12: QCRTGraphTS.ChartFont = QCRTGraphTS.ChartFont.newChartFont3("Arial",
    QCRTGraphTS.ChartFont.REGULAR, 12);

    font14: QCRTGraphTS.ChartFont = QCRTGraphTS.ChartFont.newChartFont3("Arial",
    QCRTGraphTS.ChartFont.REGULAR, 14);

    font14Numeric: QCRTGraphTS.ChartFont = QCRTGraphTS.ChartFont.newChartFont3("Arial",
    QCRTGraphTS.ChartFont.REGULAR, 14);

    public constructor() {

    }

    TimerTickProc() {
        if (!this.currentTemperature1 || !this.currentTemperature2)
            return;
        if (this.chartVu) {
            // Random data
            this.currentTemperatureValue1 += 5 * (0.5 - QCRTGraphTS.ChartSupport.getRandomDouble());
        }
    }
}
```

18 Introduction

```
this.currentTemperatureValue2 += 8 * (0.5 - QCRTGraphTS.ChartSupport.getRandomDouble());

// This method uses the default time stamp, which is the current time-of-day
this.currentTemperature1.setCurrentValueNumber(this.currentTemperatureValue1);
this.currentTemperature2.setCurrentValueNumber(this.currentTemperatureValue2);
this.chartVu.updateDraw();
}

this.timercount++;
if (this.timercount > this.timercountlimit)
    this.StopTimer();
}

public async StartTimer(milliseconds: number, count: number) {
    if (this.timerId != 0)
        this.StopTimer();
    this.timercountlimit = count;
    this.timerId = setInterval(() =>
        this.TimerTickProc(),
        milliseconds);
}

public async StopTimer() {
    clearInterval(this.timerId);
    this.timerId = 0;
}

public async BuildRTScroll(canvasid: string) {

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

    this.chartVu = QCRTGraphTS.ChartView.newChartView(htmlcanvas);
    if (!this.chartVu) return;

    this.chartVu.setPreferredSize(800, 600);

    let x1: number = 0.1;
    let y1: number = 0.1;
    let x2: number = 0.9;
    let y2: number = 0.9;
    let axisFont: QCRTGraphTS.ChartFont = this.font12;

    let startTime: Date = new Date();
    let endTime: Date = new Date();
    QCRTGraphTS.ChartCalendar.add(endTime, QCRTGraphTS.ChartConstants.SECOND, 60);

    this.InitializeProcessVar();

    if (!this.currentTemperature1 || !this.currentTemperature2)
        return;

    let pTransform1: QCRTGraphTS.TimeCoordinates =
        QCRTGraphTS.TimeCoordinates.newTimeCoordinatesTime(startTime,
            this.currentTemperature1.DefaultMinimumDisplayValue, endTime,
            this.currentTemperature1.DefaultMaximumDisplayValue);
    pTransform1.setGraphBorderDiagonal(x1, y1, x2, y2);
    let graphbackground: QCRTGraphTS.Background =
        QCRTGraphTS.Background.newBackground(pTransform1, QCRTGraphTS.ChartConstants.GRAPH_BACKGROUND,
        QCRTGraphTS.ChartColor.WHITE);
    this.chartVu.addChartObject(graphbackground);
    let plotbackground: QCRTGraphTS.Background =
        QCRTGraphTS.Background.newBackground(pTransform1, QCRTGraphTS.ChartConstants.PLOT_BACKGROUND,
        QCRTGraphTS.ChartColor.BLACK);
    this.chartVu.addChartObject(plotbackground);
    let xaxis: QCRTGraphTS.TimeAxis = QCRTGraphTS.TimeAxis.newTimeAxisType(pTransform1,
        QCRTGraphTS.ChartConstants.X_AXIS);
    xaxis.LineColor = QCRTGraphTS.ChartColor.BLACK;
    this.chartVu.addChartObject(xaxis);
```



```

        let yaxis: QCRTGraphTS.LinearAxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
QCRTGraphTS.ChartConstants.Y_AXIS);
        yaxis.LineColor = QCRTGraphTS.ChartColor.BLACK;
        this.chartVu.addChartObject(yaxis);
        let yAxisGrid: QCRTGraphTS.Grid = QCRTGraphTS.Grid.newGrid(xaxis, yaxis,
QCRTGraphTS.ChartConstants.Y_AXIS, QCRTGraphTS.ChartConstants.GRID_MAJOR);
        yAxisGrid.setColor(QCRTGraphTS.ChartColor.WHITE);
        yAxisGrid.setLineWidth(1);
        yAxisGrid.setLineStyle(QCRTGraphTS.ChartConstants.LS_DOT_1_4);
        this.chartVu.addChartObject(yAxisGrid);
        let xAxisGrid: QCRTGraphTS.Grid = QCRTGraphTS.Grid.newGrid(xaxis, yaxis,
QCRTGraphTS.ChartConstants.X_AXIS, QCRTGraphTS.ChartConstants.GRID_MAJOR);
        xAxisGrid.setColor(QCRTGraphTS.ChartColor.WHITE);
        xAxisGrid.setLineWidth(1);
        xAxisGrid.setLineStyle(QCRTGraphTS.ChartConstants.LS_DOT_1_4);
        this.chartVu.addChartObject(xAxisGrid);
        let yaxis2: QCRTGraphTS.LinearAxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
QCRTGraphTS.ChartConstants.Y_AXIS);
        yaxis2.LineColor = QCRTGraphTS.ChartColor.BLACK;
        yaxis2.AxisTickDir = QCRTGraphTS.ChartConstants.AXIS_MAX;
        yaxis2.AxisIntercept = xaxis.AxisMax;
        this.chartVu.addChartObject(yaxis2);
        let xAxisLab: QCRTGraphTS.TimeAxisLabels =
QCRTGraphTS.TimeAxisLabels.newTimeAxisLabels(xaxis);
        xAxisLab.TextFont = axisFont;
        this.chartVu.addChartObject(xAxisLab);
        let yAxisLab: QCRTGraphTS.NumericAxisLabels =
QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(yaxis);
        yAxisLab.TextFont = axisFont;
        this.chartVu.addChartObject(yAxisLab);
        this.scrollFrame =
QCRTGraphTS.RTScrollFrame.newRTScrollFrameChartViewProcessVarCoordsScrollMode(this.chartVu,
this.currentTemperature1, pTransform1,
QCRTGraphTS.ChartConstants.RT_FIXEDEXTEXT_MOVINGSTART_AUTOSCROLL);
        this.scrollFrame.addProcessVar(this.currentTemperature2);
        this.scrollFrame.ScrollScaleModeY = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_Y_MINMAX;
        // Allow 100 samples to accumulate before autoscaling y-axis. This prevents rapid
        // changes of the y-scale for the first few samples
        this.scrollFrame.MinSamplesForAutoScale = 100;
        this.scrollFrame.ScrollRescaleMargin = 0.01;
        this.chartVu.addChartObject(this.scrollFrame);
        let attrib1: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.YELLOW, 3,
QCRTGraphTS.ChartConstants.LS_SOLID);
        let lineplot1: QCRTGraphTS.SimpleLinePlot =
QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(pTransform1, attrib1);
        lineplot1.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
        let solarPanelLinePlot1: QCRTGraphTS.RTSimpleSingleValuePlot =
QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform1, lineplot1,
this.currentTemperature1);
        this.chartVu.addChartObject(solarPanelLinePlot1);
        let attrib2: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.GREEN, 3,
QCRTGraphTS.ChartConstants.LS_SOLID);
        let lineplot2: QCRTGraphTS.SimpleLinePlot =
QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(pTransform1, attrib2);
        lineplot2.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
        let solarPanelLinePlot2: QCRTGraphTS.RTSimpleSingleValuePlot =
QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform1, lineplot2,
this.currentTemperature2);
        this.chartVu.addChartObject(solarPanelLinePlot2);
        let alarmlines: QCRTGraphTS.RTAlarmIndicator =
QCRTGraphTS.RTAlarmIndicator.newRTAlarmIndicatorAxisRTPlot(yaxis, solarPanelLinePlot1);
        alarmlines.AlarmIndicatorMode = QCRTGraphTS.ChartConstants.RT_ALARM_LIMIT_LINE_INDICATOR;
        // Auto-scale can move alarm lines outside of plot area, clip the lines in this case.
        alarmlines.ChartObjClipping = QCRTGraphTS.ChartConstants.PLOT_AREA_CLIPPING;
        this.chartVu.addChartObject(alarmlines);
        let titlefont: QCRTGraphTS.ChartFont = this.font14;
        let charttitle: QCRTGraphTS.ChartTitle = QCRTGraphTS.ChartTitle.newChartTitle(pTransform1,
titlefont, "Scroll Application #1", QCRTGraphTS.ChartConstants.CHART_HEADER,
QCRTGraphTS.ChartConstants.CENTER_PLOT);

```

20 Introduction

```
this.chartVu.addChartObject(charttitle);

this.chartVu.updateDraw();

this.StartTimer(100, 10000);

}

public InitializeProcessVar() {
    this.templowalarm1 =
    QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_LOWERTHAN, 80);
    this.templowalarm1.AlarmMessage = "Low Alarm";
    this.templowalarm1.AlarmSymbolColor = QCRTGraphTS.ChartColor.BLUE;
    this.templowalarm1.AlarmTextColor = QCRTGraphTS.ChartColor.BLUE;
    this.temphighalarm1 =
    QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_GREATERTHAN, 120);
    this.temphighalarm1.AlarmMessage = "High Alarm";
    this.temphighalarm1.AlarmSymbolColor = QCRTGraphTS.ChartColor.RED;
    this.temphighalarm1.AlarmTextColor = QCRTGraphTS.ChartColor.RED;
    this.currentTemperature1 = QCRTGraphTS.RTPProcessVar.newRTPProcessVar("Temp 1",
    QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.GREEN, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN));
    this.currentTemperature1.MinimumValue = -20;
    this.currentTemperature1.MaximumValue = 200;
    this.currentTemperature1.DefaultMinimumDisplayValue = 0;
    this.currentTemperature1.DefaultMaximumDisplayValue = 150;
    this.currentTemperature1.setCurrentValueNumber(this.currentTemperatureValue1);
    this.currentTemperature1.addAlarm(this.templowalarm1);
    this.currentTemperature1.addAlarm(this.temphighalarm1);
    // Important, enables historical data collection for scroll graphs
    this.currentTemperature1.DatasetEnableUpdate = true;
    this.templowalarm2 =
    QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_LOWERTHAN, 80);
    this.templowalarm2.AlarmMessage = "Low Alarm";
    this.templowalarm2.AlarmSymbolColor = QCRTGraphTS.ChartColor.BLUE;
    this.templowalarm2.AlarmTextColor = QCRTGraphTS.ChartColor.BLUE;
    this.temphighalarm2 =
    QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_GREATERTHAN, 120);
    this.temphighalarm2.AlarmMessage = "High Alarm";
    this.temphighalarm2.AlarmSymbolColor = QCRTGraphTS.ChartColor.RED;
    this.temphighalarm2.AlarmTextColor = QCRTGraphTS.ChartColor.RED;
    this.currentTemperature2 = QCRTGraphTS.RTPProcessVar.newRTPProcessVar("Temp 2",
    QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.GREEN, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN));
    this.currentTemperature2.MinimumValue = -20;
    this.currentTemperature2.MaximumValue = 200;
    this.currentTemperature2.DefaultMinimumDisplayValue = 0;
    this.currentTemperature2.DefaultMaximumDisplayValue = 150;
    this.currentTemperature2.setCurrentValueNumber(this.currentTemperatureValue2);
    this.currentTemperature2.addAlarm(this.templowalarm2);
    this.currentTemperature2.addAlarm(this.temphighalarm2);
    // Important, enables historical data collection for scroll graphs
    this.currentTemperature2.DatasetEnableUpdate = true;

}
}
```

The “../qcrgraphts” folder specification means to back up two directory levels and go to the qcrgraphts folder, where the qcrgraphts.js library file is located. Note that is is the qcrgraphts.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/TypescriptExamplesQCRTGraph/HelloRTScroll/HelloRTScroll.html>

You can usually replace 127.0.0.1 with *localhost*.

<http://localhost:8080/TypescriptExamplesQCRTGraph/HelloRTScroll/HelloRTScroll.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 QCRTGraphTS statement, all JavaScript objects found in the library are located in the namespace QCRTGraphTS, which prevents them from conflicting with any other libraries you may be using which have similar named objects. The file qcertgraphs.js contains the entire QCRTGraph library. It also contains the entire QCChart2D charting library, upon which QCRTGraph is built.

So most of the work is done in the BuildRTScroll member function of the HelloRTScroll class. But it needs to be invoked from the HTML file you want to place the chart in. Look at the HelloRTScroll/HelloRTScroll.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 HelloRTScroll.js file found in the JavascriptExamplesQCRTGraph/HelloRTScroll folder.

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

22 Introduction

```
<div id="buttondiv">
  <button type="button" id="rtScroll_menuitem">RT Scroll</button>
</div>

<div id="canvasdiv">
  <canvas id="rtScrollCanvas1" width="800" height="600"></canvas>
</div>
  <script type="module">

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

var rtplot = new HelloRTScroll();

    function rtScroll() {
      rtplot.BuildRTScroll("rtScrollCanvas1");
    }

    // Since loading of script module is asynchronous, need to assign onclick event handler
s after module loaded.
    document.getElementById("rtScroll_menuitem").onclick = rtScroll;

  </script>

  <br>
  <br>

</body>
</html>
```

In this example the button click for the RT Scroll button invokes the JavaScript *HelloRTScroll* function call inside the script section. The *HelloRTScroll* class is imported from the *HelloRTScroll.js* file, and an instance of the class is created as the local object *rtscroll*. The RT Scroll chart button event handler function in turn calls the *rtscroll.BuildRTScroll* in the imported *HelloRTScroll.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 *HelloRTScroll* class from the *HelloRTScroll.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 *HelloRTScroll* function in the script block has not yet been loaded. Instead is it assigned explicitly in the javascript code, after the *HelloRTScroll.js* module has been loaded.

```
document.getElementById("rtbarplot_menuitem").onclick = HelloRTScroll;
```

You can read more about the ES6 module loading specification on the web:

<https://www.sitepoint.com/using-es-modules/> .

Programing QCRTGraph within Wordpress

Using QCRTGraph 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 (**BuildRTScroll** 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 { BuildRTScroll } from http://yourwebsite/HelloRTScroll.js;

  BuildRTScroll("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/HelloRTScroll.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> .

Programing QCRTGraph using the Angular framework

Angular is an open-source web application framework promulgated by Google. It is used in conjunction with the TypeScript programming language, and HTML to define page templates. It is well documented online and you can start learning it here: [AngularJS: Developer Guide: Introduction](#) .

Since this software is written entirely in TypeScript (and transpiled into JavaScript) it is particularly useful for adding charts to Angular web applications. There are only a couple of small techniques you need to know.

An Angular web application is going to have at least one HTML file that is part of the project. The HTML file is considered a template and contains Angular specific elements that define the overall look of the web page. It is updated dynamically by the programming and database elements of the framework. QCRTGraph must be displayed in a HTML *canvas* element. So you need to add a canvas element to your Angular HTML template file, positioned where you want the chart to appear. You can add as many canvas elements as you want and put unique chart in each. Just keep the variable name (#chartCanvas1 in the example below) of the canvas elements unique. Something like this:

24 Introduction

```
<br>

    <div id="buttondiv">
      <button type="button" (click)="buttonClickFunction()" >Display Chart</button>
    </div>

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

In this one of the introductory Angular examples you can download, this code was added to the body of the `app.component.html` file at the desired location. The button element is not really needed, we just added it to generate an event which in turn loads the chart. The canvas element is giving the variable name (which is sort of like an HTML ID, but accessible directly from the Angular TypeScript behind code) `#chartCanvas1` with a height and width of 800x600.

The TypeScript behind code for this template is the file `app.component.ts`. It will end up looking something like this. In this example we placed on the `QCRTGraph` calls in a separate TypeScript file (`HelloRTScroll.ts`). That file defines the `HelloRTScroll` class. The `HelloRTScroll` class is imported into the `app.component.ts` file using the statement:

```
import * as HelloRTScroll from './HelloRTScroll';
```

The complete `app.component.ts` appears below.

```
import { Component, ViewChild, ElementRef, AfterViewInit } from '@angular/core';

import * as HelloRTScroll from './HelloRTScroll';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent implements AfterViewInit {
  /** Template reference to the canvas element */
  @ViewChild('chartCanvas1', { static: true }) chartCanvas1!: ElementRef<HTMLCanvasElement>;
  title = 'Hello World!';

  private rtchart: HelloRTScroll.HelloRTScroll | null = null;

  constructor() {

  }

  buttonClickFunction() {
    // this.BuildXBarRChart();
    this.rtchart = new HelloRTScroll.HelloRTScroll();
    let htmlcanvas: HTMLCanvasElement | null = <HTMLCanvasElement>this.chartCanvas1.nativeElement;

    this.rtchart.BuildRTScroll(htmlcanvas);
  }

  ngAfterViewInit() {

  }
}
```

Include this statement in the class member variable section, referring back to the variable name *chartCanvas1* in the canvas element you placed in the `app1.component.html` file.

```
@ViewChild('chartCanvas1', { static: true }) chartCanvas1!: ElementRef<HTMLCanvasElement>;
```

When you actually build the chart (making calls into our library) you will need to convert this variable name into the underlying HTML canvas element (type `HTMLCanvasElement`) using code similar to this:

```
this.rtchart = new HelloRTScroll.HelloRTScroll();  
let htmlcanvas: HTMLCanvasElement | null = <HTMLCanvasElement>this.chartCanvas1.nativeElement;  
  
this.rtchart.BuildRTScroll(htmlcanvas);
```

The resulting *htmlcanvas* TypeScript variable can then be passed into the one of the many `new ChartView...` (or other) constructors. In this case we pass it to the `BuildRTScroll` method of the `HelloRTScroll` class we use to display the graph. In the `BuildRTScroll` method it uses the *htmlcanvas* element to create the `ChartView` class the graph is placed in.

The `HelloRTScroll.ts` file contains the TypeScript class `HelloRTScroll` used to display the scrolling graph. At the top is an import of the `QCRTGraphTS` library.

```
import * as QCRTGraphTS from '../..../QCRTGraphTS/qcrtgraphts.js';
```

The bulk of the remaining code is similar to all of our other TypeScript examples. The only exception is the type definition of the `timerId` variable as a member function of the class. For some reason TypeScript is very particular about how the return value from a call to the standard function `setInterval` is defined, and we found that this is how it needs to be typed.

```
timerId: null | ReturnType<typeof setTimeout> = null
```

The `BuildRTScroll` method does the work of creating the graph. It uses the passed in *htmlcanvas* element to create a `ChartView`, and the rest of the chart elements are added to that view.

```
public BuildRTScroll( htmlcanvas: HTMLCanvasElement) {  
  
    this.chartVu = QCRTGraphTS.ChartView.newChartView(htmlcanvas);  
    if (!this.chartVu) return;  
  
    this.chartVu.setPreferredSize(800, 600);  
.  
.
```

Assuming you have a standard global Angular development environment installed, you can run it by going to the command line, going into the angular example folder where the project is, starting a development server in the root of the project folder by running `"ng serve"`. Then point your web browser to ["http://localhost:4200/"](http://localhost:4200/) and the test page should appear. As with most development environments, make sure you can run your application before you start trying to add the charts to it.

You will find the Angular specific files discussed in this section (app.component.html, app.component.ts, and HelloRTScroll.ts) in the TypeScriptExamplesQCRTGraph/AngularExampleCode/src/app folder. The overall Angular project (a very simple one) ends up so huge we did not want to burden the download with entire project.

Directory Structure of QCRTGraph for JavaScript/TypeScript

The QCRTGraph for JavaScript/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

QCRTGraphJSTSDoc.pdf – This document, the User guide

QCRTGraphTS\ - Contains the QCRTGraphJSTS library

docs/

modules\

qcrgraphts.html - help index file

AspNetRTGraphVS2019\ - Holds ASP.Net examples specific directory

JavascriptExamplesQCRTGraph\ - Holds the JavaScript programming examples

Annunciators

AutoInstrumentPanel

Bargraphs

Dynamometer

FetalMonitor

HelloRTBarPlot

HelloRTMeterPlot

HelloRTScroll

HomeAutomation

HybridCar

Meters

MiniScope

PIDControlTuner

Polygraph

ProcessMonitoring

ProcessVarDataTables

RTStockDisplay

RTXYPlot

ScrollingGraphs

SweepGraphExample1

TrackbarsAndButtons

Treadmill

WeatherStation

TypescriptExamplesQCRTGraph\ - TypeScript Example programs

Same files as JavaScript above, only for TypeScript

Versions of QCRTGraphTS

There are two versions of the QCRTGraph for JavaScript/TypeScript class library: the 30-day trial versions, and the developer version. Each version has different characteristics that are summarized below:

30-Day Trial Version

The trial version of QCRTGraph for JavaScript/TypeScript is downloaded as a zip file named Trial_QCRTGraphJSTS.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.

Developer Version

The developer version of QCRTGraph for JavaScript/TypeScript is downloaded as a zip file, name something similar to JSTSRTGDEV1R3x0x353x1.zip. The developer version does not time out and you can use it to create application programs 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 **Real-Time Graphics Tools for JavaScript/TypeScript** package designed to run on any hardware that has a JavaScript/TypeScript runtime installed on it.

Chapter 2 presents the overall class architecture of the **Real-Time Graphics Tools for JavaScript/TypeScript** and summarizes all of the classes found in the software.

Chapter 3 summarizes the important **QCRTGraph** classes that you must be familiar with in order to use the **Real-Time Graphics Tools for JavaScript/TypeScript** software.

Chapter 4 describes the process variable and alarm classes that hold **Real-Time Graphics Tools for JavaScript/TypeScript** data.

Chapter 5 describes the panel meter classes: numeric, alarm, string and time/date panel meters.

Chapter 6 describes the single channel bar indicator classes, including segmented, custom, and pointer bar subtypes.

Chapter 7 describes the multi-channel bar indicator classes, including segmented, custom, and pointer bar subtypes.

Chapters 8 describe the meter setup classes: meter coordinates, meter axes, and meter axis labels.

Chapter 9 describes the meter indicator classes including classes for meter needles, arc, segmented arc, and symbol indicators.

Chapter 10 how the meter indicator classes are used to create dials and clocks..

Chapter 11 describes the annunciator classes.

Chapter 12 describes the scroll frame classes (**RTScrollFrame** and **RTVerticalScrollFrame**) and the implementation of scrolling plots based on the **QCRTGraph SimpleLinePlot**, **SimpleBarPlot**, **SimpleScatterPlot** and **SimpleLineMarkerPlot** classes using the: **RTSimpleSingleValuePlot** class.

Chapter 13 describes the **RTGroupMultiValuePlot** class and the implementation of scrolling plots based on the **GroupPlot**.

Chapter 14 describes custom control classes: buttons, and track bars.

Chapter 15 describes the PID control class.

Chapter 16 describes tricks and techniques for zooming of real-time data.

Chapter 17 describes miscellaneous classes for drawing shapes and creating rectangular and circular backdrops for graphs and controls.

Chapter 18 describes the **RTProcessVarViewer** class – a data table used to display historical data collected by the **RTProcessVar** class.

Chapter 19 describes the printing of the real-time charts created using **QCRTGraph**.

Chapter 20 is a tutorial that describes how to use **Real-Time Graphics Tools for *JavaScript/TypeScript*** to create web applications using Visual Studio Code, using JavaScript and TypeScript..

Chapter 21 is a collection of Frequently Asked Questions about **Real-Time Graphics Tools for *JavaScript/TypeScript***.

2. Class Architecture of the Real-Time Graphics Tools for *JavaScript/TypeScript* Class Library

Major Design Considerations

This chapter presents an overview of the **Real-Time Graphics Tools for *JavaScript/TypeScript*** class architecture. It discusses the major design considerations of the architecture:

Based on the **QCChart2D** charting architecture, it has the same design considerations listed in that software. These are:

- 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.
- Design consideration specific to **Real-Time Graphics Tools for *JavaScript/TypeScript*** are:
- Updates of data classes are asynchronous with rendering of graphics to the screen.
- Real-Time plot objects are derived from **QCChart2D** plot objects resulting in standardized methods for setting plot object properties.
- Any standard plot type from the **QCChart2D** software package, both simple and group plot types, can be implemented as scrolling graphs.
- There are no limits on the number of process variable channels, no limits on the number of alarm limits associated with a process variable, no limits on the number of real-time plots in a graph.
- The update of real-time objects will not interfere or overwrite other objects and will follow the z-order precedence established when the graph was created.

The chapter also summarizes the classes in the **Real-Time Graphics Tools for *JavaScript/TypeScript*** library.

There are five primary features of the overall architecture of the **Real-Time Graphics Tools for JavaScript/TypeScript** classes. These features address major shortcomings in existing charting software for use with both JavaScript/TypeScript and other computer languages.

- **Real-Time Graphics Tools for JavaScript/TypeScript** uses the standard JavaScript/TypeScript HTML5/Canvas architecture. Real-Time graphs are placed in a **ChartView** window that is a container for the chart objects. Position one or more **ChartView** objects in JavaScript/TypeScript container windows using a HTML5 canvas object. Mix static **QCChart2D** and real-time charts with other components in the same container.
- The **Real-Time Graphics Tools for JavaScript/TypeScript** software uses a new real-time update and rendering paradigm which represents a shift in the way Quinn-Curtis has always done real-time updates in past. In the past, graphs were always updated incrementally as new data arrived. This is no longer the case. Instead rendering is no longer incremental. When a graph is rendered, the entire graph is redrawn using the most current data. A special process variable class (**RTProcessVar**) is used to store new data as it is acquired. In the case of graphs that require a historical display of information, such as scrolling graphs, the process variable class also manages a **ChartDataset** object that holds historical information. Updating the process variable with new data values does NOT trigger a screen update. Because the screen update is not event driven from the update of the data, the process variable can be updated hundreds, or even thousands of times faster than the screen. The graph should be rendered to the screen, using a timer or some other event, at a frame rate of 10 updates/second or slower. The rendered graphs will always reflect the most current data, and in the case of scrolling graphs or other graphs that display time persistent data, will always display all data within the current scale limits. As processor speeds improve and JavaScript/TypeScript become faster, the screen updates should be able to approach the 30-60 frames/seconds of a CRT monitor. It will never need to be higher than that because the eye cannot track changes in the screen faster than that anyway.
- Since all real-time plot objects are derived from the **ChartPlot** class, the methods and properties of that class are available to set commonly used attributes such as the real-time plot object scale, line and fill colors.
- Many new real-time classes have been added to the software, implementing display objects that render process variable data in a variety of graph and text formats. These include single and multiple bar indicator classes, meter axis and meter indicator classes, panel meter classes, and annunciator classes. Rather than create a whole new set of classes that reproduce all of the **SimplePlot** and **GroupPlot** classes of the **QCChart2D** library, two special classes (**RTSimpleSingleValuePlot** and **RTGroupMultiValuePlot**) are used to interface the **static** plot objects to the process variable data classes. That way any **SimplePlot** or **GroupPlot** object can be converted into a real-time scrolling graph without adding any code to the **Real-Time Graphics Tools for JavaScript/TypeScript** library.

Real-Time Graphics Tools for *JavaScript/TypeScript* Class Summary

The **Real-Time Graphics Tools for *JavaScript/TypeScript*** library is a super set of the **QCChart2D** library. The classes of the **QCChart2D** library are an integral part of the software. A summary of the **QCChart2D** classes appears below.

QCChart2D 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.

For each of these categories see the associated description in the **QCChart2D** manual. The **Real-Time Graphics Tools for JavaScript/TypeScript** classes are in addition to the ones above. They are summarized below.

Real-Time Graphics Tools for JavaScript/TypeScript Class Summary

Process Variable and Alarms

Real-time data is stored in **RTProcessVar** classes. The **RTProcessVar** class is designed to represent a single process variable, complete with limit values, an unlimited number of high and low alarms, historical data storage, and descriptive strings for use in displays.

Single Value Indicators

A single value indicator is a real-time display object that is attached to a single **RTProcessVar** object. This includes single channel bar indicators (which includes solid, segmented, custom and pointer bar indicators), meter indicators (which includes meter needles, meter arcs and meter symbol indicators), single channel annunciator indicators, panel meter indicators and scrolling graph plots based on a **SimplePlot** chart object.

Multiple Value Indicators

A multiple value indicator is a real-time display object that is attached to a group of **RTProcessVar** objects. This includes multiple channel bar indicators (which includes solid, segmented, custom and pointer bar indicators), multiple channel annunciator indicators, panel meter indicators organized in a grid, and scrolling graph plots based on a **GroupPlot** chart object.

Alarm Indicators

Alarm indicators are used to display alarm lines, symbols and fill areas for the **RTProcessVar** objects associated with the single value and multiple value indicator classes.

Panel Meter Classes

The **RTPanelMeter** derived classes are special cases of the single value indicator classes that are used throughout the software to display real-time data in a text format. Panel meters are available for numeric values, string values, time/date values and alarm values.

Meter Axis Classes

Meter indicators needed new classes to support the drawing of meter axes, meter axis labels and meter alarm objects.

Control Classes

Since HTML5 does not include canvas-base scroll bars and buttons, we have created some simple stand-ins for these essential objects. The **RTControlButton** class implements on/off colors and on/off text for momentary, toggle and radio button style buttons. The **RTControlTrackBar** class implements a track bar control that can be used to input integer or floating point data values into your program. **RTControlButton** and **RTControlTrackBar** objects can be group together in a grid, organizing the control objects functionally and visually.

Scroll Frame	A scroll frame manages constant rescaling of coordinate systems of plot objects (RTSimpleSingleValuePlot and RTGroupMultiValuePlot objects) that are displayed in a scrolling graph. The RTScrollFrame class manages a horizontal scroll frame, while the RTVerticalScrollFrame manages a vertical scroll frame.
Auto Indicator Classes	A group of classes encapsulate the real-time indicators (bars, meters, dials, clocks, panel meter, and scroll graphs) as self-contained ChartView derived classes, so that they can be placed individually on forms.
Miscellaneous Classes	Support classes are used to display special symbols used for alarm limits in the software, special round and rectangular shapes that can be used as backdrops for groupings of chart objects and PID control.

Real-Time Graphics Tools for JavaScript/TypeScript Classes

The **QCRTGraph** classes are a super set of the **QCChart2D** charting software. No attempt should be made to utilize the **QCRTGraph** classes without a good understanding of the **QCChart2D** classes. See the **Chart2DJSTSTManual3** PDF file for detailed information about the **QCChart2D** classes. The diagram below depicts the class hierarchy of the **Real-Time Graphics Tools for JavaScript/TypeScript** library without the additional **QCChart2D** classes

Namespace .

- RTAlarmEventArgs
- RTGroupDatasetTruncateArgs
- RTDatasetTruncateArgs

ChartObj

- RTAlarm
- RTAlarmIndicator
- RTMultiAlarmIndicator
- RTProcessVar
 - RTComboProcessVar
- RTPIDControl
- RTRoundedRectangle2D
- RTSymbol
- RTTextFrame
- RTGenShape
- RT3DFrame

ChartPlot

- RTPlot
 - RTSingleValueIndicator
 - RTPanelMeter
 - RTNumericPanelMeter
 - RTAlarmPanelMeter
 - RTStringPanelMeter
 - RTTimePanelMeter
 - RTElapsedTimePanelMeter
 - RTFormControlPanelMeter
 - RTAnnunciator
 - RTBarIndicator
 - RTMeterIndicator
 - RTMeterArcIndicator
 - RTMeterNeedleIndicator
 - RTMeterSymbolIndicator
 - RTSimpleSingleValuePlot
 - RTMultiValueIndicator
 - RTMultiValueAnnunciator
 - RTMultiBarIndicator
 - RTGroupMultiValuePlot
 - RTFormControlGrid
 - RTScrollFrame
 - RTVerticalScrollFrame

PolarCoordinates

```

    RTMeterCoordinates
LinearAxis
    RTMeterAxis
NumericAxisLabels
    RTMeterAxisLabels
StringAxisLabels
    RTMeterStringAxisLabels

ButtonArea
    RTControlButton
ScrollbarArea
    RTControlTrackBar
ChartObj
    RTFormControl

```

QCRTGraph Classes

Process Variable and Alarms Classes

RTProcessVar
RTAlarm
RTAlarmEventArgs

Real-time data is stored in **RTProcessVar** classes. The **RTProcessVar** class is designed to represent a single process variable, complete with limit values, an unlimited number of high and low alarms, historical data storage, and descriptive strings for use in displays.

RTProcessVar Real-time data is stored in **RTProcessVar** classes. The **RTProcessVar** class is designed to represent a single process variable, complete with limit values, an unlimited number of high and low alarms, historical data storage, and descriptive strings for use in displays.

RTAlarm The **RTAlarm** class stores alarm information for the **RTProcessVar** class. The **RTAlarm** class specifies the type of the alarm, the alarm color, alarm text messages and alarm hysteresis value. The **RTProcessVar** classes can hold an unlimited number of **RTAlarm** objects in a Array.

RTAlarmEventArgs

The **RTProcessVar** class can throw an alarm event based on either the current alarm state, or an alarm transition from one alarm state to another. The **RTAlarmEventArgs** class is used to pass alarm data to the event handler.

Panel Meter Classes

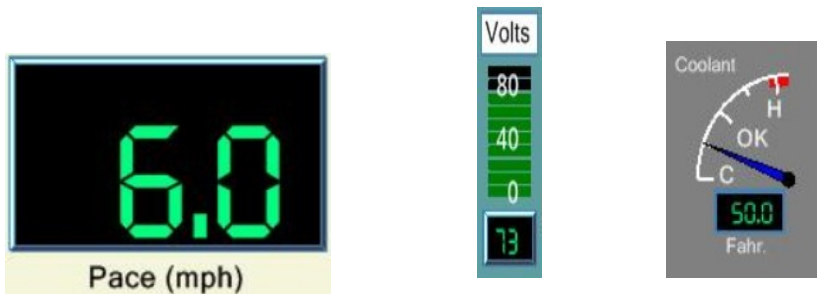
ChartPlot**RTPlot****RTSingleValueIndicator****RTPanelMeter****RTNumericPanelMeter****RTAlarmPanelMeter****RTStringPanelMeter****RTTimePanelMeter****RTElapsedTimePanelMeter****RTFormControlPanelMeter**

The **RTPanelMeter** derived classes are special cases of the single value indicator classes that are used throughout the software to display real-time data in a text format. Panel meters are available for numeric values, string values, time/date values and alarm values. All of the panel meter classes have a great many options for controlling the text font, color, size, border and background of the panel meter rectangle.

RTPanelMeter objects are used in two ways. First, they can be standalone, and once attached to an **RTProcessVar** object they can be added to a **ChartView** as any other **GraphObj** derived class. Second, they can be attached to most of the single channel and multiple channel indicators, such as **RTBarIndicator**, **RTMultiBarIndicator**, **RTMeterIndicator** and **RTAnnunciator** objects, where they provide text output in addition to the indicators graphical output.

RTPanelMeter

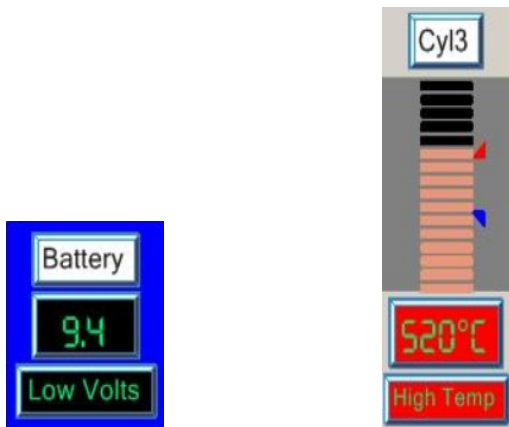
The abstract base class for the panel meter types.



Numeric panel meters can be the primary display method for real-time data, or they can be used as adjuncts to other real-time indicators such as bar indicators and meters.

RTNumericPanelMeter

Displays the floating point numeric value of an **RTProcessVar** object. It contains a template based on the **NumericLabel** class that is used to specify the font and numeric format information associated with the panel meter.



*The lowest panel meter in these examples is the **RTAlarmPanelMeter** object. Alarm properties include custom text for all alarm levels. When an alarm occurs, the foreground color of alarm text and the background color of the alarm text rectangle can be programmed to change state.*

RTAlarmPanelMeter

Displays an alarm text message. It contains a template based on the **StringLabel** class that is used to specify the font and string format information associated with the panel meter. It bases the alarm text message on the alarm information in the associated **RTProcessVar** object.



Panel meter strings can be used to display an objects tag name, units string, and description.

RTStringPanelMeter Displays a string, either an arbitrary string, or a string based on string data in the associated **RTProcessVar** object. It is usually used to display a channels tag string and units string, but it can also be used to display longer descriptive strings. It contains a template based on the **StringLabel** class that is used to specify the font and string format information associated with the panel meter.



*The **RTTimePanelMeter** can display a time/date value in any format supported by the **TimeLabel** format constants. You can also create custom format not directly supported.*

RTTimePanelMeter Displays the time/date value of the time stamp of the associated **RTProcessVar** object. It contains a template based on the **TimeLabel** class that is used to specify the font and time/date format information associated with the panel meter.

RTElapsedTimePanelMeter

Displays the elapsed time (the **ChartTimeSpan** value of the time stamp in milliseconds) of the associated **RTProcessVar** object. It contains a template based on the **ElapsedTimeLabel** class that is used to specify the font and time/date format information associated with the panel meter.

RTFormControlPanelMeter

Encapsulates an **RTFormControl** object (buttons and track bars primarily, though others will also work) in a panel meter format.

Single Value Indicators

ChartPlot

RTPlot

RTSingleValueIndicator

RTAnnunciator
RTBarIndicator
RTMeterIndicator
 RTMeterArcIndicator
 RTMeterNeedleIndicator
 RTMeterSymbolIndicator
RTPanelMeter
RTSimpleSingleValuePlot

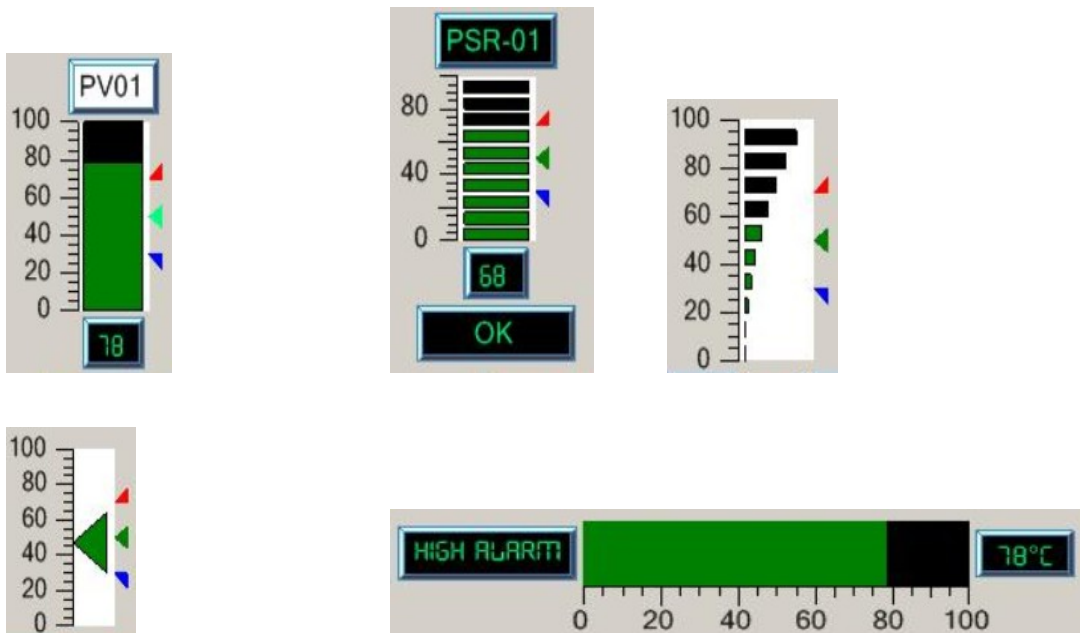
Display objects derived from the **RTSingleValueIndicator** class are attached to a single **RTProcessVar** object. This includes single channel bar indicators (which includes solid, segmented, custom and pointer bar indicators), meter indicators (which includes meter needles, meter arcs and meter symbol indicators), single channel annunciator indicators, panel meter indicators and scrolling graph plots based on a **SimplePlot** chart object. These objects can be positioned in a chart using one of the many chart coordinate systems available for positioning, including physical coordinates (PHYS_POS), device coordinates (DEV_POS), plot normalized coordinates (NORM_PLOT_POS) and graph normalized coordinates (NORM_GRAPH_POS).



An annunciator can contain any combination string, numeric and alarm panel meters. The background color of the annunciator can change in response to an alarm event.

RTAnnunciator

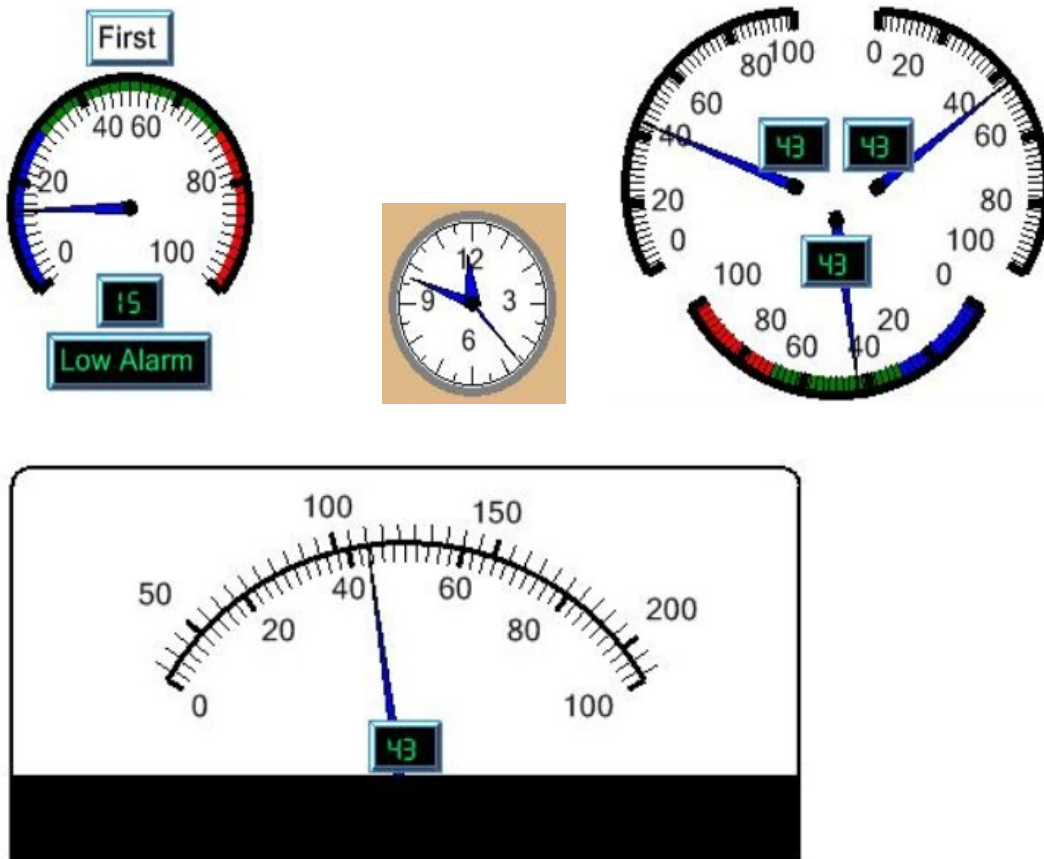
An **RTAnnunciator** is used to display the current values and alarm states of a single channel real-time data. It consists of a rectangular cell that can contain the tag name, units, current value, and alarm status message. Any of these items may be excluded. If a channel is in alarm, the background of the corresponding cell changes its color, giving a strong visual indication that an alarm has occurred.



Clockwise from top: Solid bar indicator, segmented bar indicator, custom segmented bar indicator, horizontal solid bar indicator, and pointer indicator. Only the green bars (and pointer) represent the bar indicators. Other items also shown include axes, axis labels, panel meters, and alarm indicators.

RTBarIndicator

An **RTBarIndicator** is used to display the current value of an **RTProcessVar** using a bar changing its size. One end of each bar is always fixed at the specified base value. Bars can change their size either in vertical or horizontal direction. Sub types within the **RTBarIndicator** class support segmented bars, custom segmented bars with variable width segments, and pointer bar indicators.

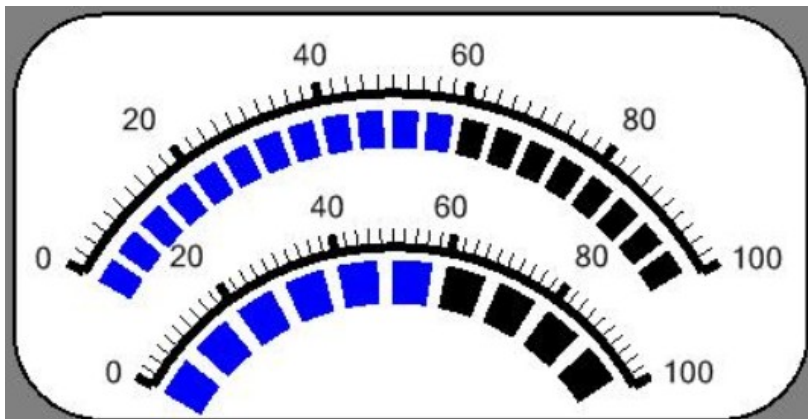
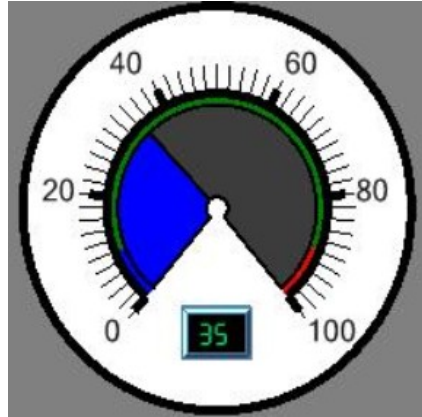


There are an infinite number of meter designs possible using a variety of meter arc ranges, meter scales, meter axes and meter indicator types

RTMeterIndicator

The **RTMeterIndicator** class is the abstract base class for all meter indicators. Familiar examples of analog meters are voltmeters, car speedometers, pressure gauges, compasses and analog clock faces. Three meter types are supported: arc, symbol, and needle meters. An unlimited number of meter indicators can be added to a given meter object. **RTPanelMeter** objects can be attached to an **RTMeterIndicator** object

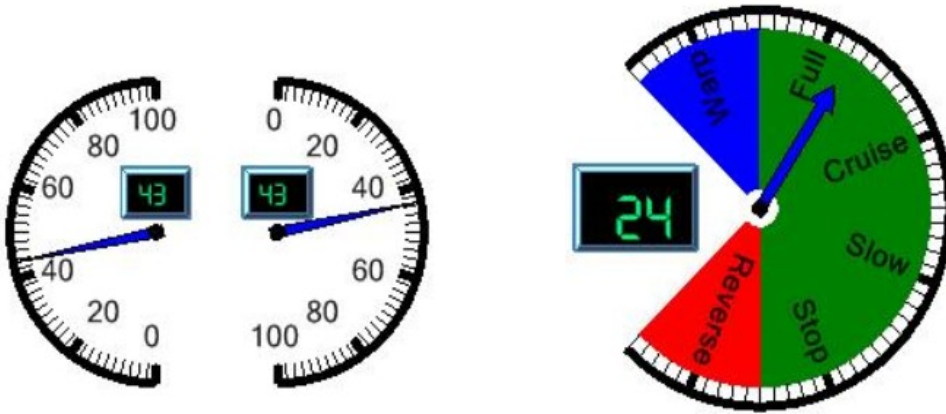
for the display of **RTProcessVar** numeric, alarm and string data in addition to the indicator graphical display. Meter scaling, meter axes, meter axis labels and alarm objects and handle by other classes.



Only the blue meter arc is the arc indicator. The other elements of the meter include meter axes, meter axis labels and panel meters for the numeric, tag and alarm displays.

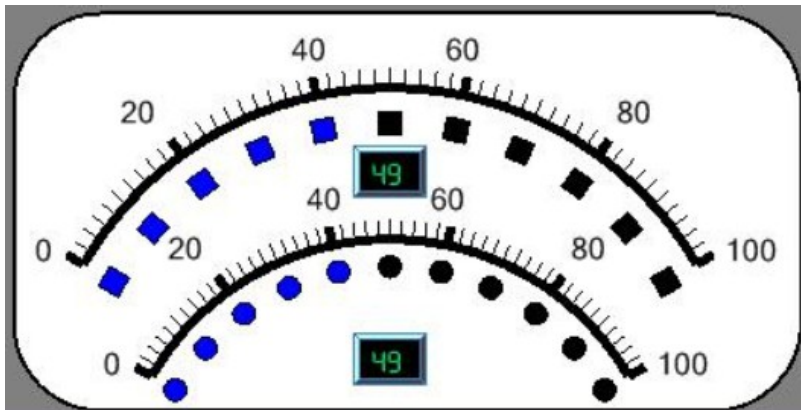
RTMeterArcIndicator

This **RTMeterArcIndicator** class displays the current **RTProcessVar** value as an arc. Segmented meter arcs are one of the **RTMeterArcIndicator** subtypes.



Only the blue meter needles are the meter needle indicators. The other elements of the meter include meter axes, meter axis labels and panel meters for the numeric, tag and alarm displays.

RTMeterNeedleIndicator This **RTMeterNeedleIndicator** class displays the current **RTProcessVar** value as a needle. Subtypes of the **RTMeterNeedleIndicator** are simple needles, pie wedge shaped needles and arrow needles.

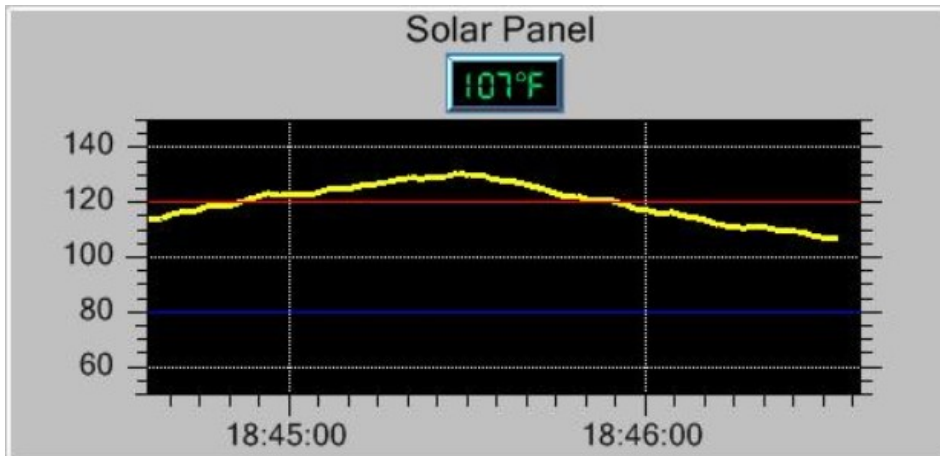


Meter symbols can be any of 10 different shapes, the symbols can have any size, and the spacing between the symbols can have any value.

RTMeterSymbolIndicator This **RTMeterSymbolIndicator** class displays the current **RTProcessVar** value as a symbol moving around in the meter arc. Symbols include all of the scatter plot symbols: SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, BAR3D, CIRCLE.

RTPanelMeter

The abstract base class for the panel meter types. Panel meters based objects can be added to **RTSingleValueIndicator** and **RTMultiValueIndicator** objects to enhance the graphics display with numeric, alarm and string information. The **RTNumericPanelMeter**, **RTAlarmPanelMeter**, **RTStringPanelMeter** and **RTTimePanelMeter** classes are described in the preceding section.



*Any number of **RTSimpleSingleValuePlot** objects can be added to a scrolling graph.*

RTSimpleSingleValuePlot The **RTSimpleSingleValuePlot** plot class uses a template based on the **SimplePlot** class to create a real-time plot that displays **RTProcessVar** current and historical real-time data in a scrolling line, scrolling bar, or scrolling scatter plot format.

Multiple Value Indicators

ChartPlot

RTPlot

RTMultiValueIndicator

RTMultiValueAnnunciator

RTMultiBarIndicator

RTGroupMultiValuePlot

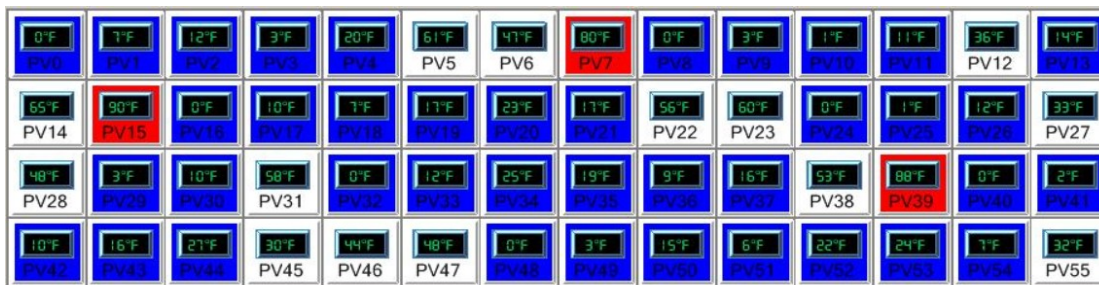
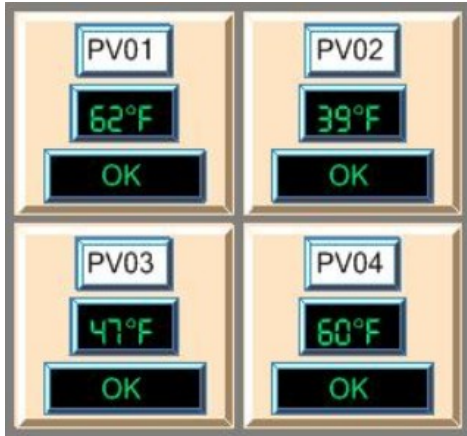
RTFormControlGrid

RTScrollFrame

RTVerticalScrollFrame

Display objects derived from the **RTMultiValueIndicator** class are attached to a collection of **RTProcessVar** objects. This includes multi-channel bar indicators (which includes solid, segmented, custom and pointer bar indicators), multi-channel annunciator indicators, and scrolling graph plots based

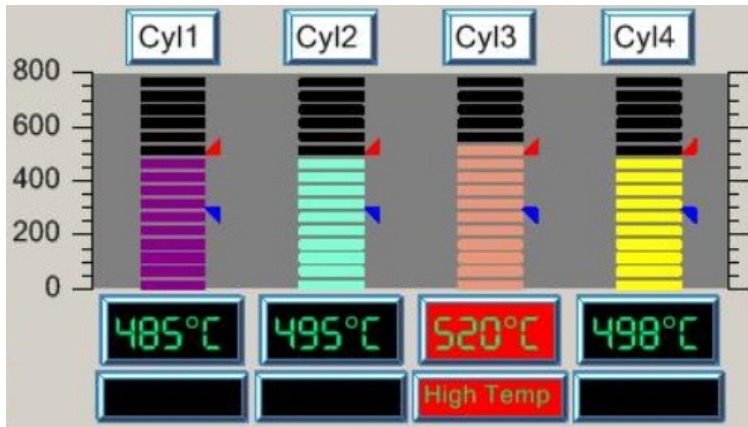
on a **GroupPlot** chart object. These objects can be positioned in a chart using one of the many chart coordinate systems available for positioning, including physical coordinates (PHYS_POS), device coordinates (DEV_POS), plot normalized coordinates (NORM_PLOT_POS) and graph normalized coordinates (NORM_GRAPH_POS).



*The only limit to the number of annunciator cells you can have in an **RTMultiValueAnnunciator** graph is the size of the display and the readability of the text.*

RTMultiValueAnnunciator

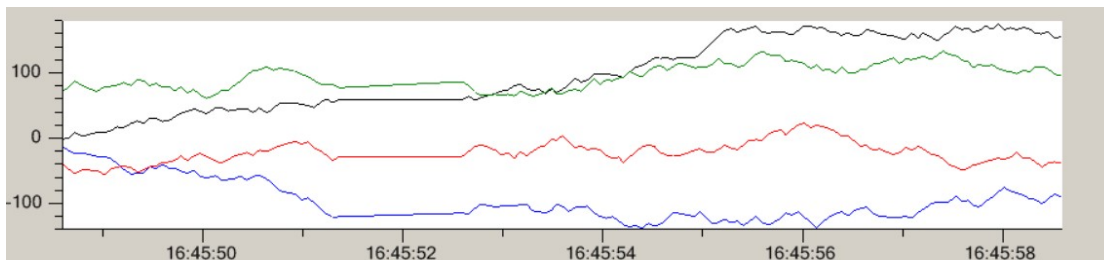
An **RTMultiValueAnnunciator** is used to display the current values and alarm states of a collection of **RTProcessVar** objects. It consists of a rectangular grid with individual channels represented by the rows and columns in of the grid. Each grid cell can contain the tag name, units, current value, and alarm status message for a single **RTProcessVar** object. Any of these items may be excluded. If a channel is in alarm, the background of the corresponding cell changes its color, giving a strong visual indication that an alarm has occurred.



Each bar in the **RTMultiBarIndicator** can have individual colors and alarm limits.

RTMultiBarIndicator

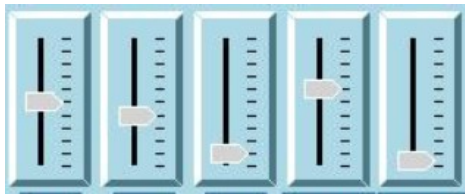
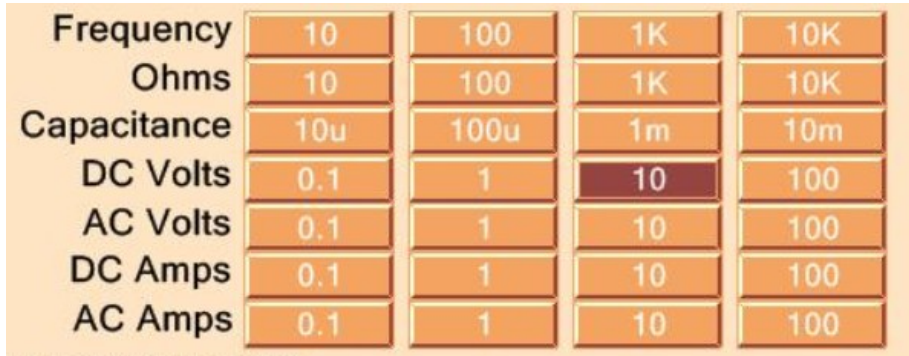
An **RTMultiBarIndicator** is used to display the current value of a collection of **RTProcessVar** objects using a group of bars changing size. The bars are always fixed at the specified base value. Bars can change their size either in vertical or horizontal direction. Sub types within the **RTMultiBarIndicator** class support segmented bars, custom segmented bars with variable width segments, and pointer bar indicators.



The **RTGroupMultiValuePlot** class turns **GroupPlot** objects, like the **MultiLinePlot** object above, into scrolling plots.

RTGroupMultiValuePlot

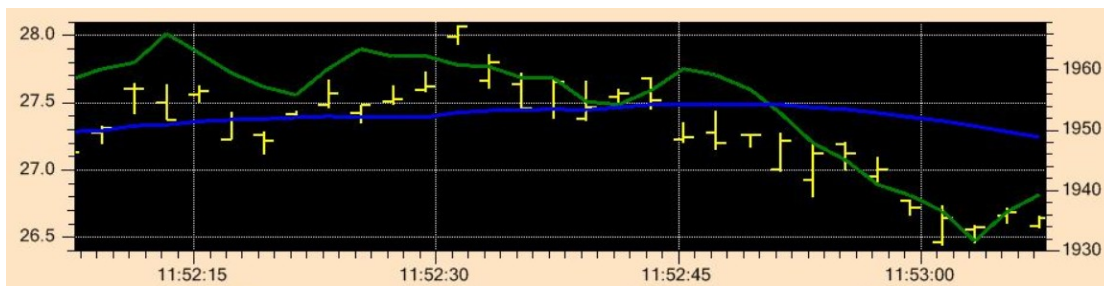
The **RTGroupMultiValuePlot** plot class uses a template based on the **GroupPlot** class to create a real-time plot that displays a collection of **RTProcessVar** objects as a group plot in a scrolling graph.



The **RTFormControlGrid** class organizes buttons and trackbars objects functionally and visually.

RTFormControlGrid

The **RTFormControlGrid** plot class organizes a group of **RTControlTracker** and **RTControlButton** objects in a grid format.



This **RTScrollFrame** combines an **RTGroupMultiValuePlot** (the open-high-low-close plot) with two **RTSimpleSingleValuePlot** plots.

RTScrollFrame

The **RTScrollFrame** plot manages horizontal scrolling and auto-scaling for the **RTSimpleSingleValuePlot** and **RTGroupMultiValuePlot** objects. The **RTScrollFrame** class is discussed in more detail a couple of sections down.

RTVerticalScrollFrame

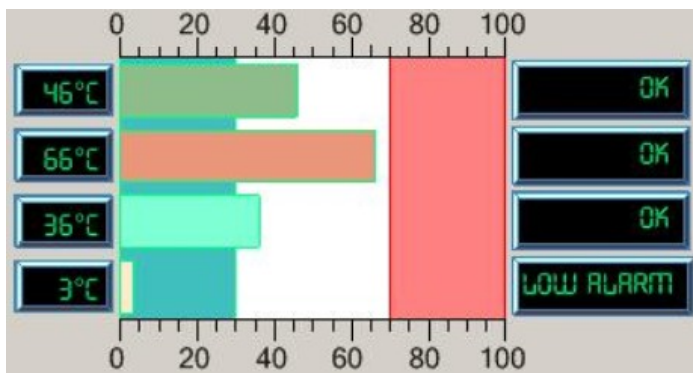
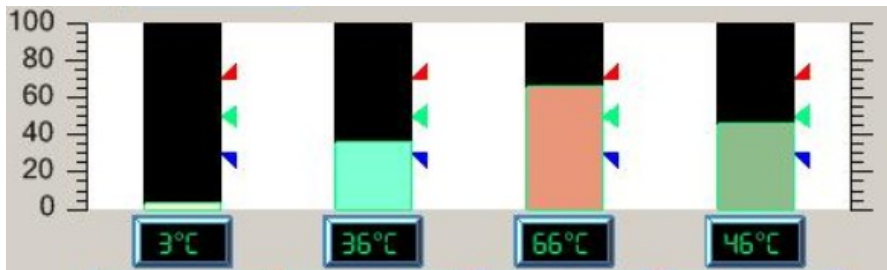
The **RTVerticalScrollFrame** plot manages vertical scrolling and auto-scaling for the **RTSimpleSingleValuePlot** and **RTGroupMultiValuePlot** objects. The **RTScrollFrame** class is discussed in more detail a couple of sections down.

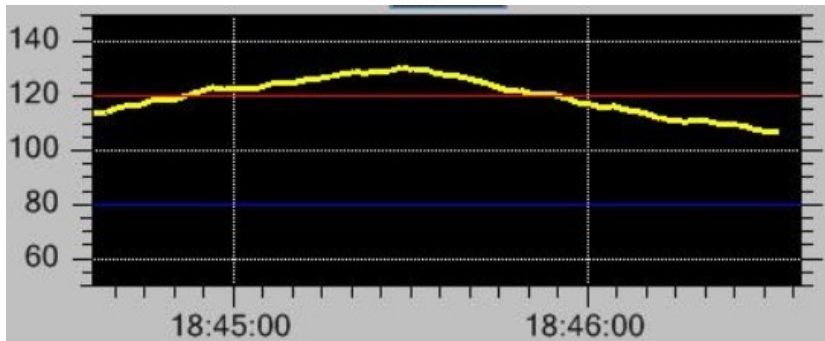
Alarm Indicator Classes

RTAlarmIndicator

RTMultiAlarmIndicator

The alarm indicator classes are used to indicate alarms limits in displays that use a Cartesian (XY) coordinate system. The alarm indicators can have one of three forms: pointer style symbols, horizontal or vertical lines, or horizontal or vertical filled areas. These alarm indicator classes are not used in meter displays. Alarm limits for meter displays are handled by the **RTMeterAxis** class.





The alarm indicators can have one of three forms: pointer style symbols, horizontal or vertical lines, or horizontal or vertical filled areas.

RTAlarmIndicator This class is used to provide alarm limit indicators for **RTSingleValueIndicator** objects.

RTMultiAlarmIndicator This class is used to provide alarm limit indicators for **RTMultiValueIndicator** objects. Each indicator in a multi-indicator object can have unique alarm settings.

Meter Axis Classes

PolarCoordinates

RTMeterCoordinates

LinearAxis

RTMeterAxis

NumericAxisLabels

RTMeterAxisLabels

StringAxisLabels

RTMeterStringAxisLabels

RTMeterCoordinates A meter coordinate system has more properties than a simple Cartesian coordinate system, or even a polar coordinate system. Because of the variation in meter styles, a meter coordinate system sets the start and end angle of the meter arc within the 360 degree polar coordinate system. It also maps a physical coordinate system, representing the meter scale, on

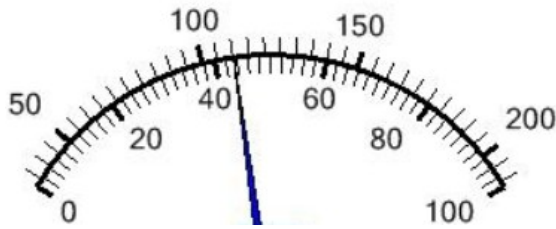
top of the meter arc. And the origin of the meter coordinate system can be offset in both x- and y-directions with respect to the containing plot area.



A meter axis can have any number of alarm arcs. A meter can have multiple axes representing multiple scales

RTMeterAxis

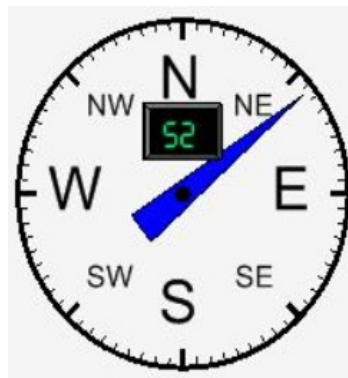
A meter axis extends for the extent of the meter arc and is centered on the origin. Major and minor tick marks are placed at evenly spaced intervals perpendicular to the meter arc. The meter axis also draws meter alarm arcs using the alarm information in the associated **RTProcessVar** object.



A useful feature of multiple meter scales is the ability to display both Fahrenheit and Centigrade scales at the same time.

RTMeterAxisLabels

This class labels the major tick marks of the **RTMeterAxis** class. The class supports many predefined and user-definable formats, including numeric, exponent, percentage, business and currency formats.



Meter tick mark strings can be horizontal, parallel and perpendicular to the tick mark.

RTMeterStringAxisLabels This class labels the major tick marks of the **RTMeterAxis** class using user-defined strings

Form Control Classes

ButtonArea

RTControlButton

ScrollbarArea

RTControlTrackBar

RTPanelMeter

RTFormControlPanelMeter

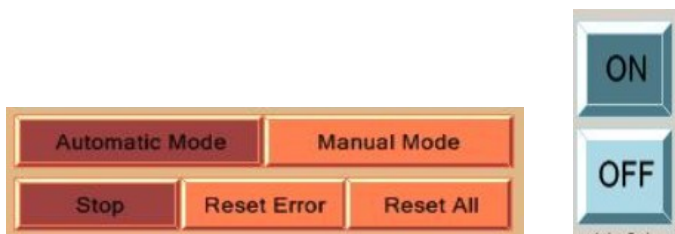
RTMultiValueIndicator

RTFormControlGrid

Real-time displays often require user interface features such as buttons and track 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.

We created Canvas versions of simple scrollbar and button controls. These have the class names **ScrollbarArea** and **ButtonArea**, and are described in the QCChart2D documentation. They are compatible both mouse and touch based systems. The QCRTGraph software contains two additional controls (**RTControlTrackBar**, and **RTControlButton**), subclassed from ScrollbarArea and ButtonArea. The **RTControlTrackBar** control adds a trackbar style pointer thumb, axis and tickmarks to the scrollbar.. The **RTControlButton** is pretty much the same as the ButtonArea control. Typical examples of each type of control are seen below.

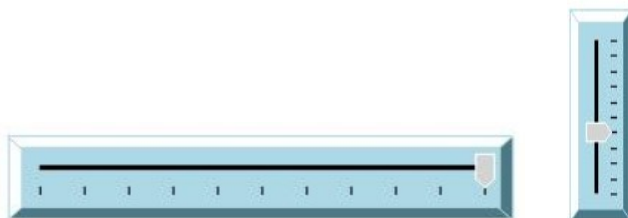
The RTControlTrackBar and RTControlButton classes can be used in conjunction with the **RTFormControlPanelMeter** and **RTFormControlGrid** classes.



The RTControlButton type supports momentary, toggle and radio button styles.

RTControlButton

Derived from the JavaScript/TypeScript Form Control **Button** class, it adds superior On/Off button text and color control and supports momentary, toggle and radio button styles.

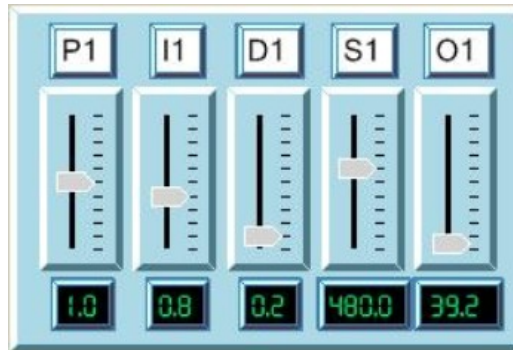


Horizontal and vertical track bars can be scaled for physical real world coordinates.

RTControlTrackBar Derived from the JavaScript/TypeScript Form Control **TrackBar** class, it adds floating point scaling for the track bar endpoints, increments, current value and tick mark frequency.

RTFormControlPanelMeter

This panel meter class contains encapsulates a RTControlButton or RTControlTrackBar object in a panel meter class, so that controls can be added to indicator objects.



***RTFormControlGrid** objects are arranged in a row x column format. Additional panel meter objects (numeric and string panel meters in the track bar example above) can be attached to the primary control grid object.*

RTFormControlGrid The **RTFormControlGrid** organizes a collection of **control** objects functionally and visually in a grid format.

Scroll Frame Class

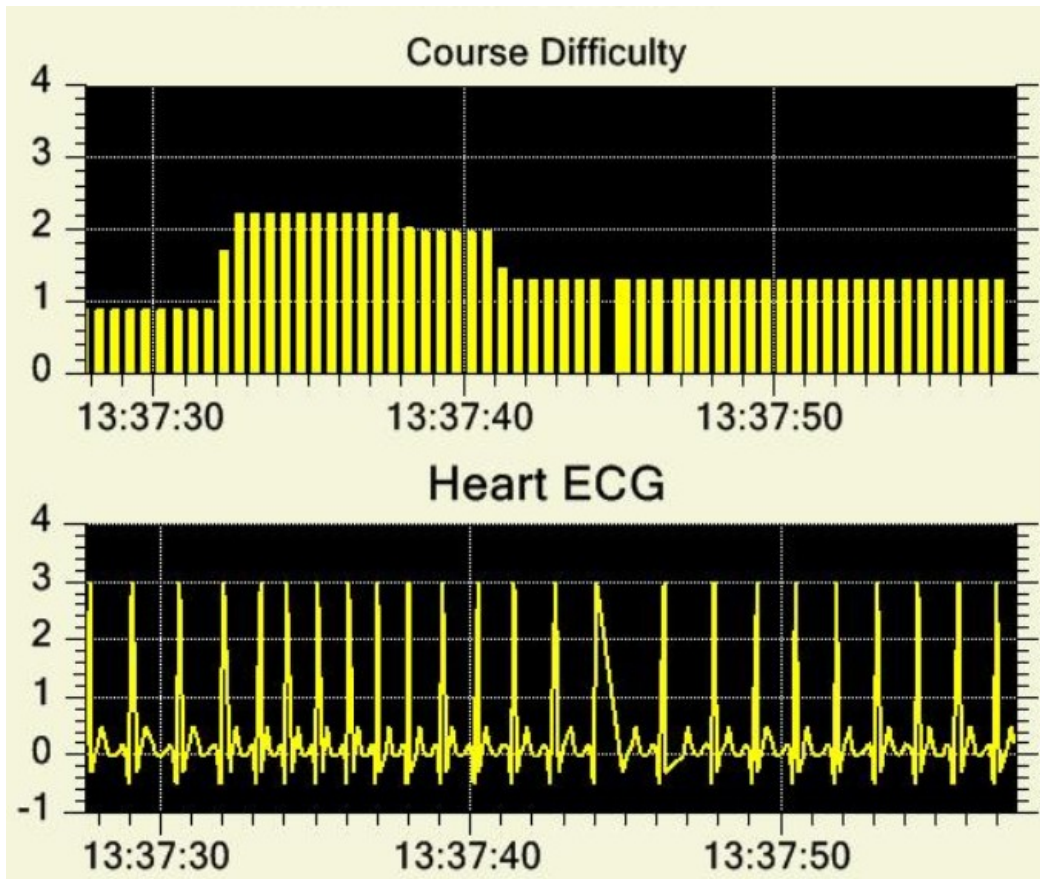
ChartPlot

RTPlot

RTMultiValueIndicator

RTScrollFrame

RTVerticalScrollFrame

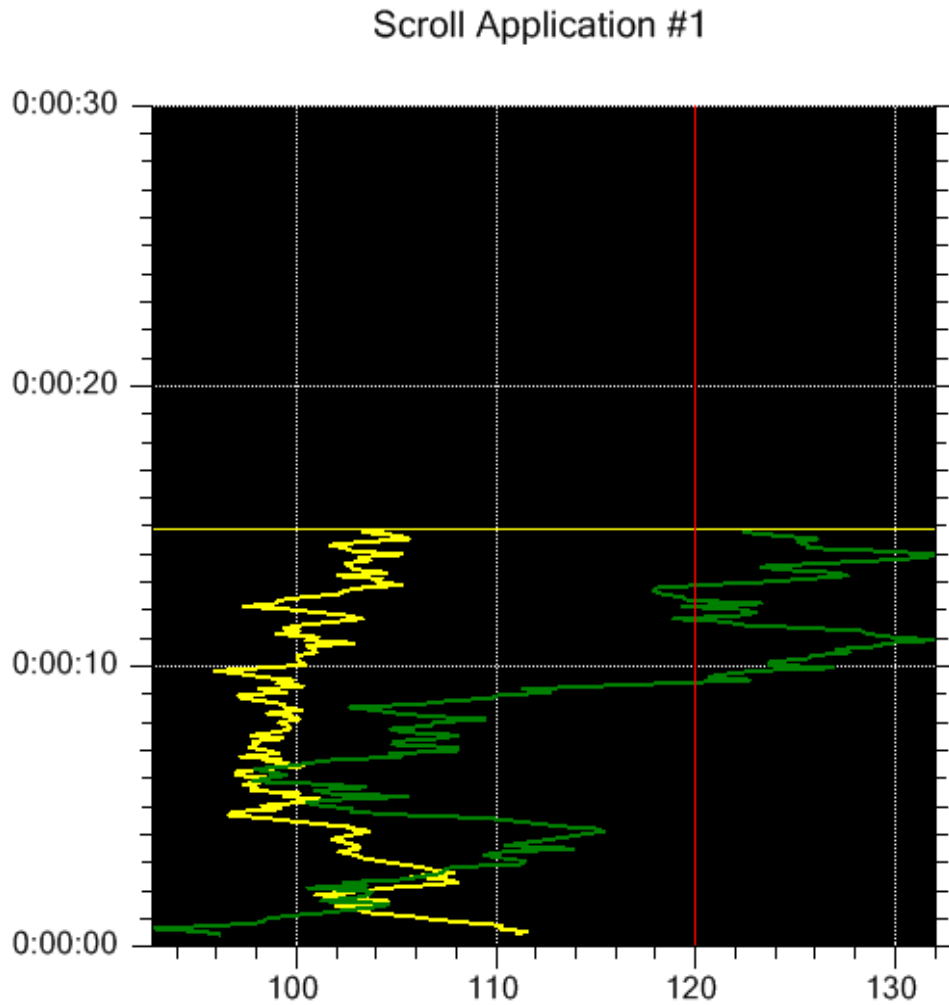


A display can have multiple scroll frames. The frames can be in separate plots and update in a synchronized fashion, or they can overlap the same plotting area.

RTScrollFrame

The scrolling algorithm used in this software is different than in earlier Quinn-Curtis real-time graphics products. Scrolling plots are no longer updated incrementally whenever the underlying data is updated. Instead, the underlying **RTProcessVar** data objects are updated as fast as you want. Scrolling graphs (all graphs for that matter) are only updated with the **ChartView.updateDraw()** method is called. What makes scrolling graphs appear to scroll is the scroll frame (**RTScrollFrame**). When a scroll frame is updated as a result of the **ChartView.updateDraw()** event, it analyzes the **RTSimpleSingleValuePlot** and **RTGroupMultiValuePlot** objects that have been attached to it and creates a coordinate system that matches the current and historical data associated with the plot objects. The plot objects in the scroll frame are drawn into this coordinate system. As data progresses forward in time the coordinate system is constantly being rescaled to include the most recent time values as part of the x-coordinate system. You can control whether or not the starting point of the scroll frame coordinate system remains fixed, whether it advances in sync

with the constantly changing end of the scroll frame. Other options allow the y-scale to be constantly rescaled to reflect the current dynamic range of the y-values in the scroll frame. The **RTScrollFrame** horizontally scrolls data with a numeric, time/date or elapsed time, time stamp.



An example of a vertical elapsed time scroll frame.

RTVerticalScrollFrame The **RTScrollFrame** has the limitation that it can only manage horizontal scrolling. The **RTVerticalScrollFrame** is much the same as **RTScrollFrame**, only it manages scrolling in the vertical direction. The **RTVerticalScrollFrame** vertically scrolls data with a numeric, time/date or elapsed time, time stamp.

Miscellaneous Classes Support classes are used to display special symbols used for alarm limits in the software, special round and rectangular shapes that can be used as backdrops for groupings of chart objects and PID control.

Miscellaneous Classes

GraphObj

RT3DFrame

GraphObj

RTGenShape

ChartObj

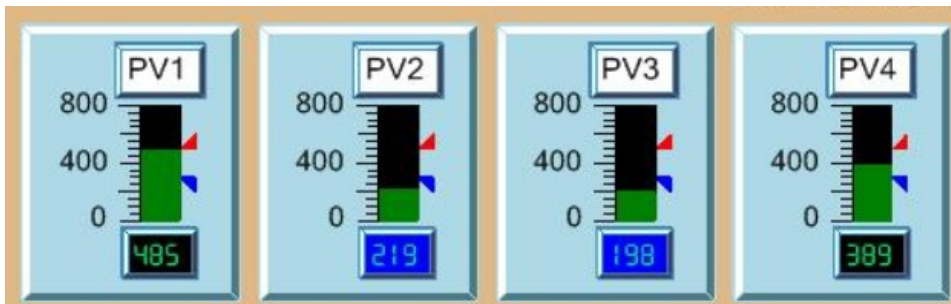
RTPIDControl

GraphObj

RTSymbol

ChartText

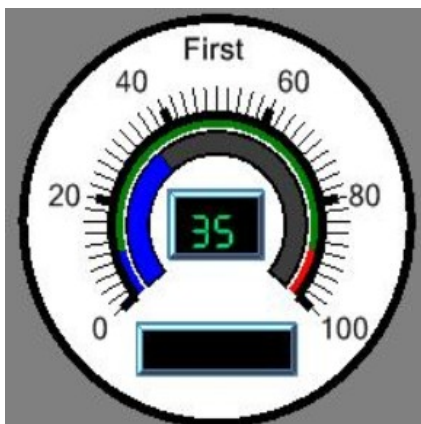
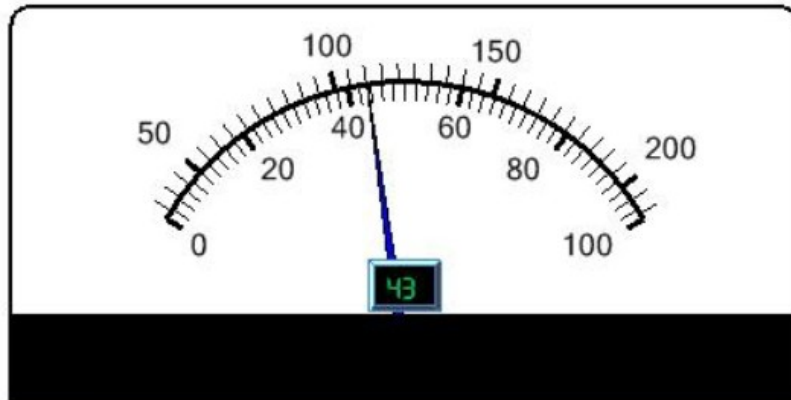
RTTextFrame



*The raised light blue panels are created using **RT3DFrame** objects.*

RT3DFrame

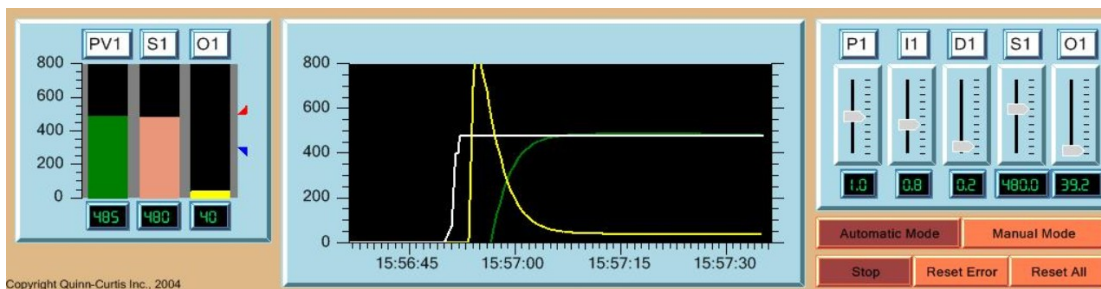
This class is used to draw 3D borders and provide the background for many of the other graph objects, most noticeably the **RTPanelMeter** classes. It can also be used directly in your program to provide 3D frames the visually group objects together in a faceplate format.



*The border rectangles in the top graph and the border circle in the bottom graph were created using **RTGenShape** objects.*

RTGenShape

This class is used to draw filled and unfilled rectangles, rectangles with rounded corners, general ellipses and aspect ratio corrected circles. These shapes can be used by the programmer to add visual enhancements to graphs.



A complete PID Control tuning center can be created using the PID control tools, bar indicators, scroll frames, buttons and track bars.

RTPIDControl	This class represents a simple control loop with support for proportional, integral and derivative control. It includes advanced features for anti-reset windup, error term smoothing, error term reset and rate limiting of control outputs.
RTSymbol	This class is used by the RTAlarmIndicator class to draw the alarm indicator symbols.
RTTextFrame	This adds a 3D border to the standard ChartText text object and recalculates justification parameters to take into account the thickness of the border. It is used by the RTPanelMeter classes to display text.

3. QCChart2D for *JavaScript/TypeScript* Class Summary

This chapter is a summary of the information in the **Chart2DJSTManual3** PDF file. It is not meant to replace that information. Refer to that manual for detailed information concerning these classes.

QCChart2D for *JavaScript/TypeScript* Class Summary

The following categories of classes realize these design considerations.

Chart view class	The chart view class is a container that manages the graph objects placed in the graph
Data classes	There are data classes for simple xy and group data types. There are also data classes that handle Date date/time data and elapsed time data.
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 interaction classes	These classes are event delegates that trap mouse 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	These classes render the chart image to a printer, to a variety of file formats including JPEG, and BMP, or to a JavaScript/TypeScript Image object.
Miscellaneous utility classes	Other classes use these for data storage, file I/O, and 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. Since the **ChartView** class is a container for other JavaScript/TypeScript components too.

Data Classes

ChartDataset

SimpleDataset

TimeSimpleDataset

ElapsedTimeSimpleDataset

GroupDataset

TimeGroupDataset

ElapsedTimeGroupDataset

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 **ChartCalendar** dates (a wrapper around the Date value class) 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.

GroupDataset	Represents group data, where every x-value can have one or more y-values.
TimeGroupDataset	A subclass of GroupDataset , it uses ChartCalendar dates (a wrapper around the Date value class) 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.

Scale Classes

ChartScale

LinearScale

LogScale

TimeScale

ElapsedTimeScale

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.

Coordinate Transform Classes

UserCoordinates

WorldCoordinates

WorkingCoordinates

PhysicalCoordinates
CartesianCoordinates
ElapsedTimeCoordinates
PolarCoordinates
AntennaCoordinates
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 drawing classes and contains routines for drawing lines, rectangles and text using JavaScript/TypeScript device coordinates.
WorldCoordinates	This class derives from the UserCoordinates class and maps a device independent world coordinate system on top of the JavaScript/TypeScript device coordinate system.
WorkingCoordinates	This class derives from the WorldCoordinates class and extends the physical coordinate system of the <i>plot area</i> (the area typically bounded by the charts axes) to include the complete <i>graph area</i> (the area of the chart outside of the <i>plot area</i>).
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 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.

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

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 ChartCalendar 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.

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

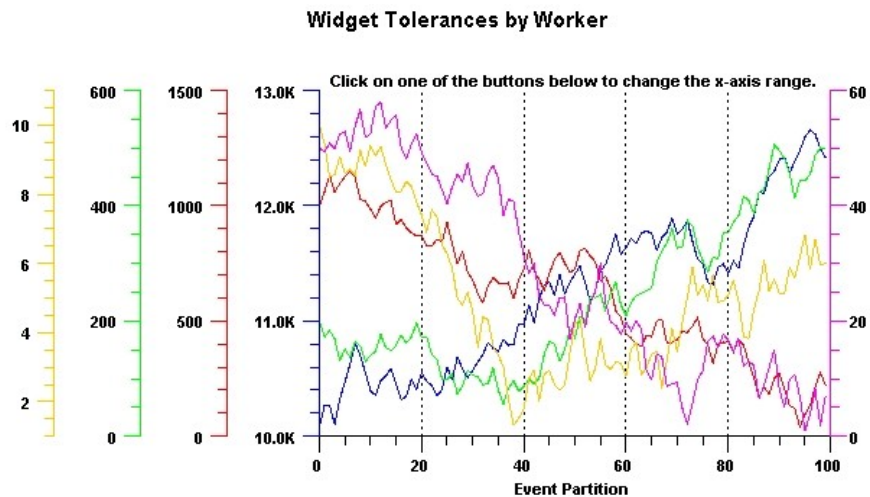
AntennaAxes

ElapsedTimeAxis

LogAxis

TimeAxis

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.

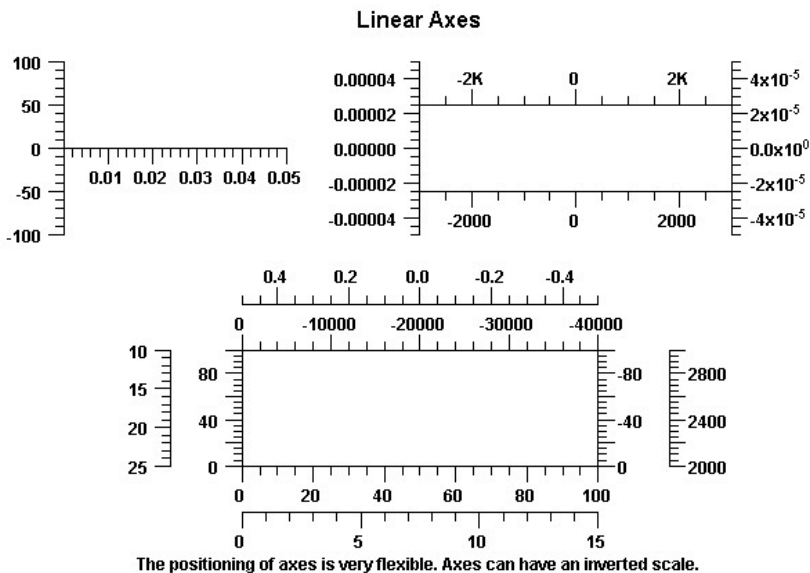


Graphs can have an UNLIMITED number of x- and y-axes.

68 QCChart2D Class Summary

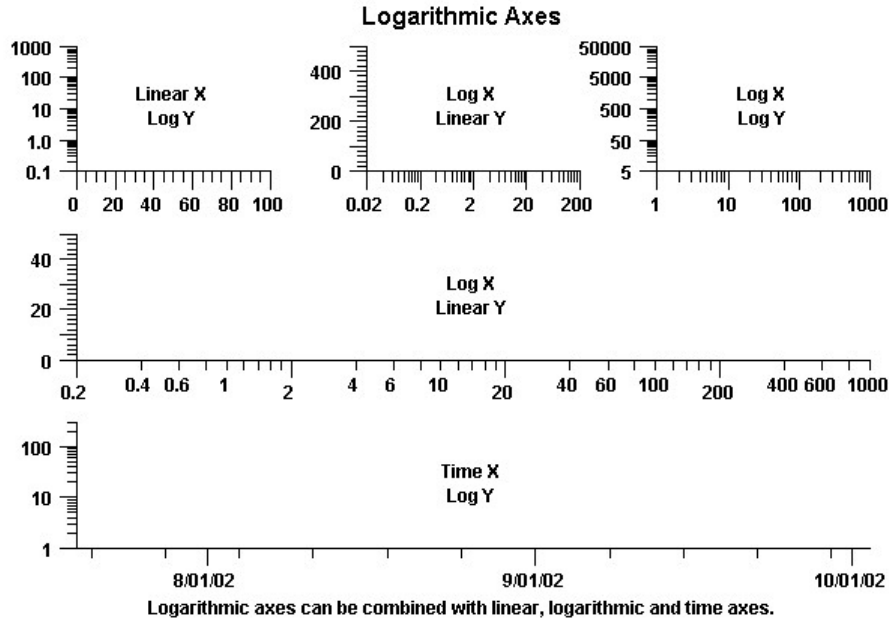
Axis

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



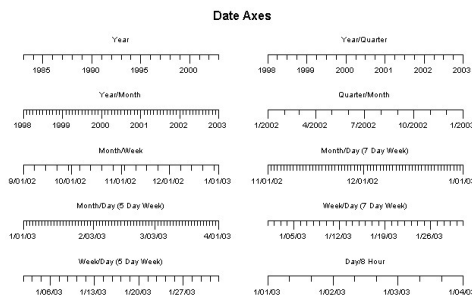
LinearAxis

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

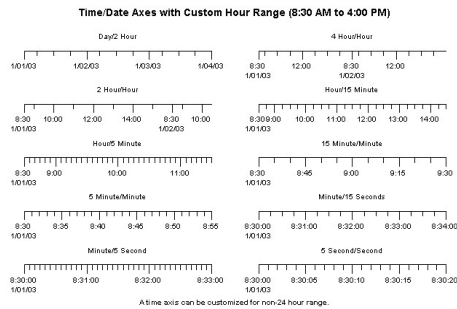
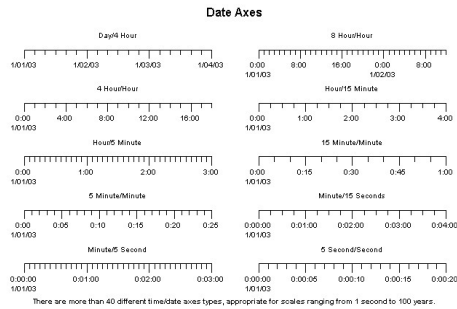
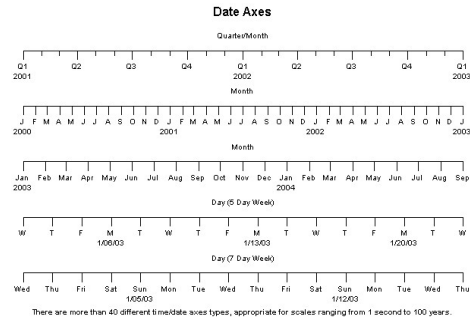


LogAxis

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.



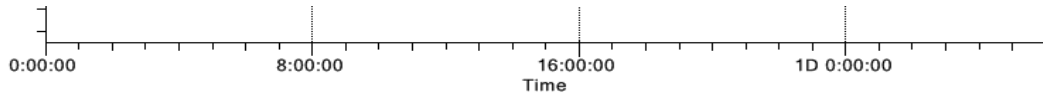
70 QCChart2D Class Summary



TimeAxis

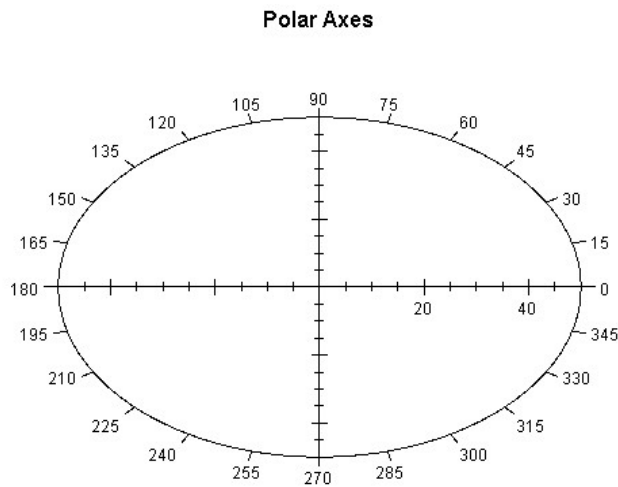
This class is the most complex of the axis classes. It supports time scales ranging from 1 millisecond to hundreds of years. Dates and times are specified using the JavaScript/TypeScript **ChartCalendar** class. The major and minor tick marks can fall on any time base, where a time base represents seconds, 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

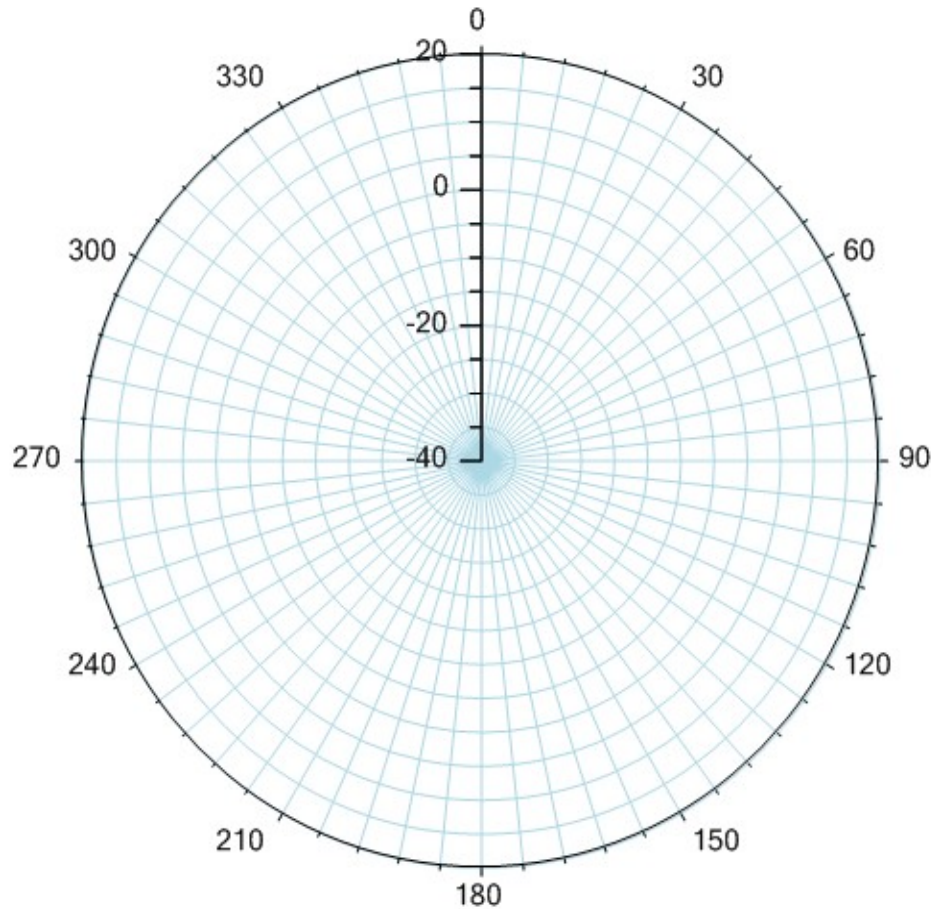
The elapsed time axis is very similar to the linear axis and is subclassed from that class. The main difference is the major and minor tick mark spacing calculated by the CalcAutoAxis method takes into account the base 60 of seconds per minute and minutes per hour, and the base 24 of hours per day. It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



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 $\pm$ 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 2π 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

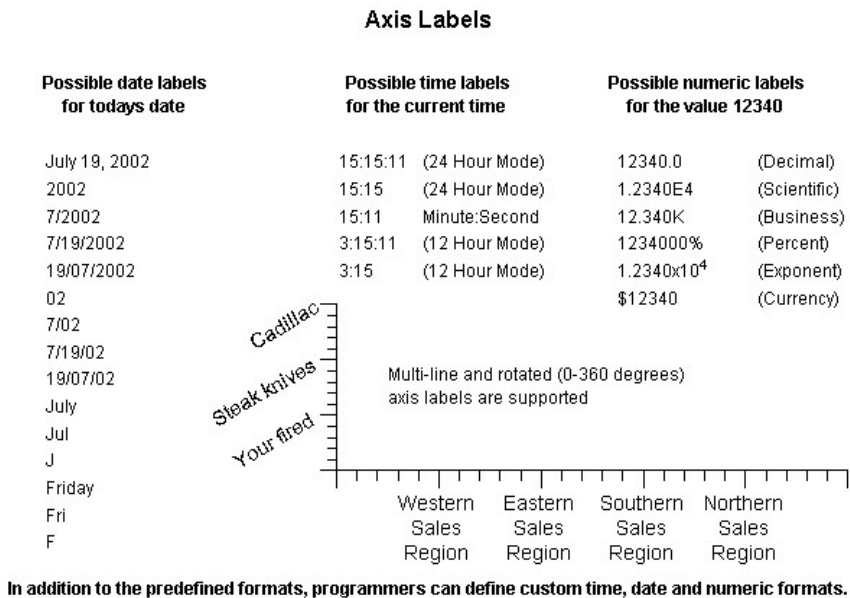
PolarAxesLabels

AntennaAxesLabels

TimeAxisLabels

ElapsedTimeAxisLabels

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.



- 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.

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 ($a + bi$), 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, where 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

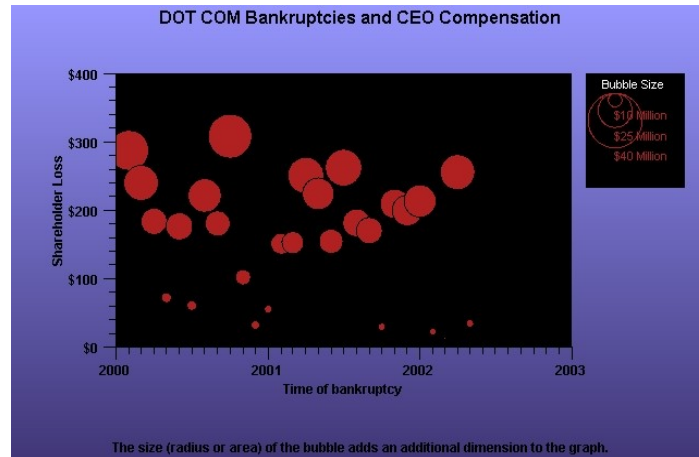
GroupPlot

- BoxWhiskerPlot**
- BubblePlot**
- CandlestickPlot**
- CellPlot**
- ErrorBarPlot**
- FloatingBarPlot**
- FloatingStackedBarPlot**
- GroupBarPlot**
- GroupVersaPlot**
- HistogramPlot**
- LineGapPlot**
- MultiLinePlot**
- OHLCPlot**
- StackedBarPlot**
- StackedLinePlot**
- GroupVeraPlot**

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.

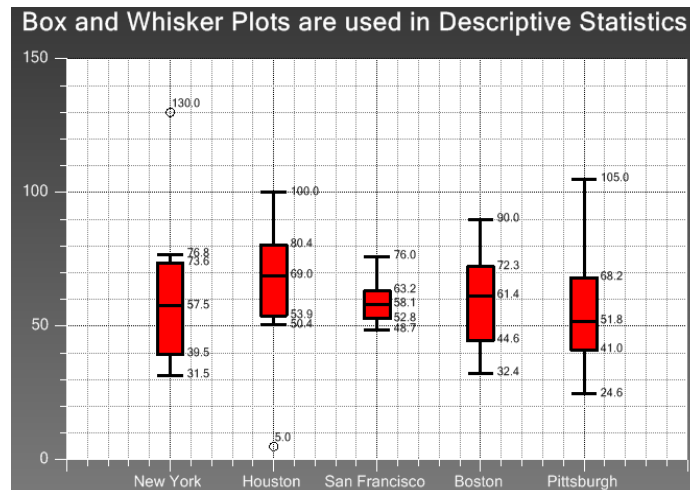
GroupPlot

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



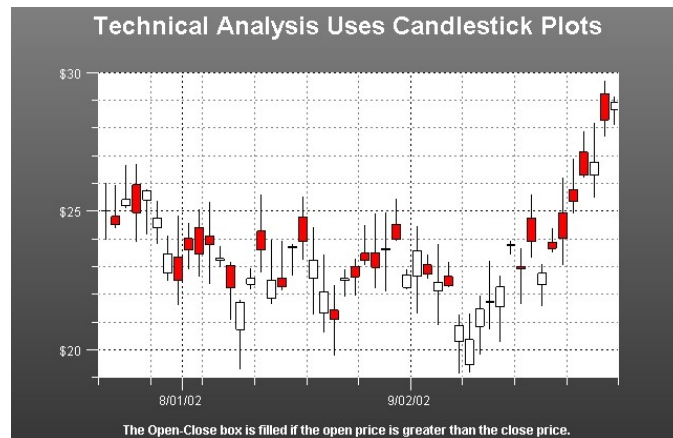
BubblePlot

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.



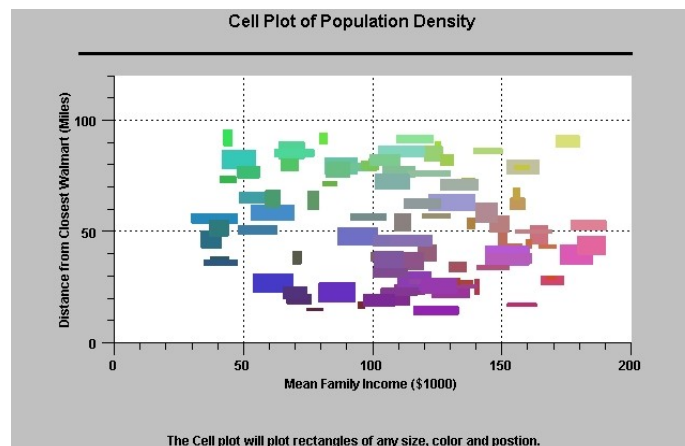
BoxWhiskerPlot

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.

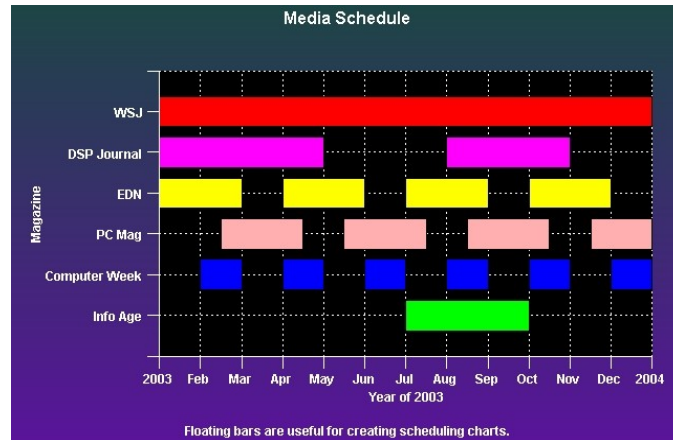


CellPlot

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.

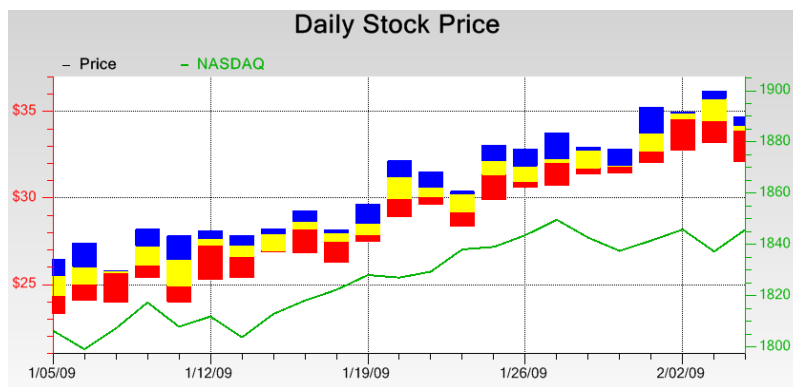
ErrorBarPlot

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.



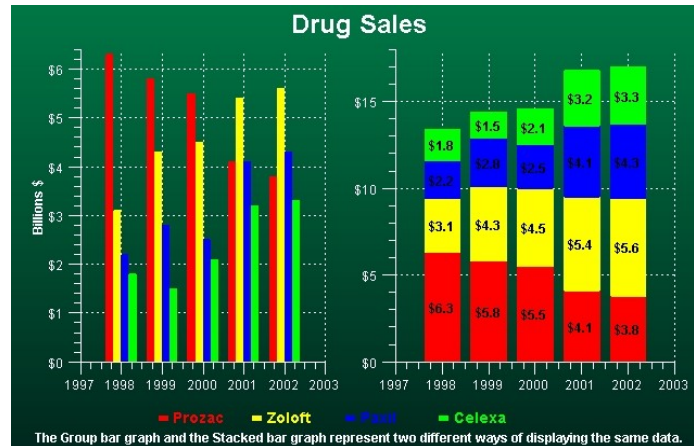
FloatingBarPlot

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

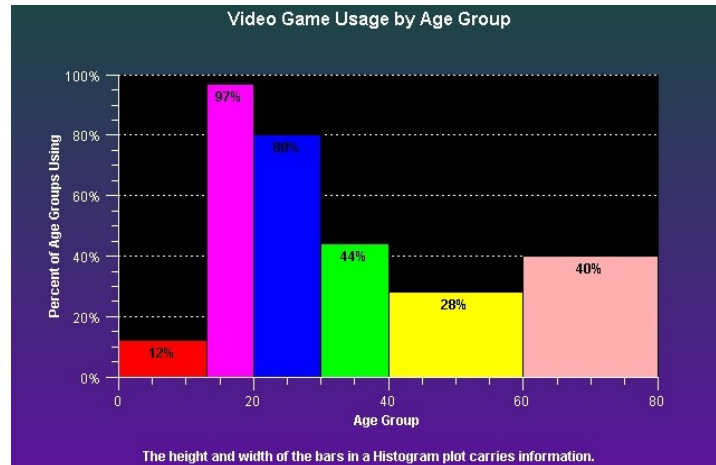
This class is a concrete implementation of the **GroupPlot** class and displays group data in a group bar format. Individual bars, the height of which corresponds to the group y-values of the dataset, display side by side, as a group, justified with respect to the x-position value for each group. The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

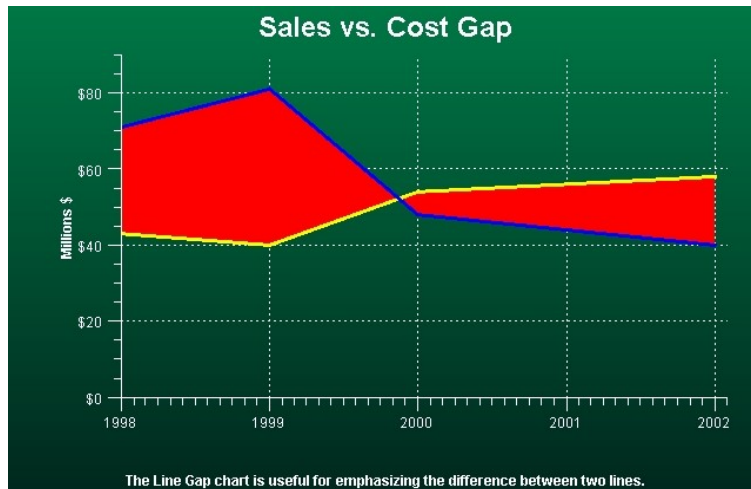
GroupVersaPlot

The **GroupVersaPlot** is a plot type that can be any of the eight group plot types: **GROUPBAR**, **STACKEDBAR**, **CANDLESTICK**, **OHLC**, **MULTILINE**, **STACKEDLINE**, **FLOATINGBAR** and **FLOATING_STACKED_BAR**. Use it 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.



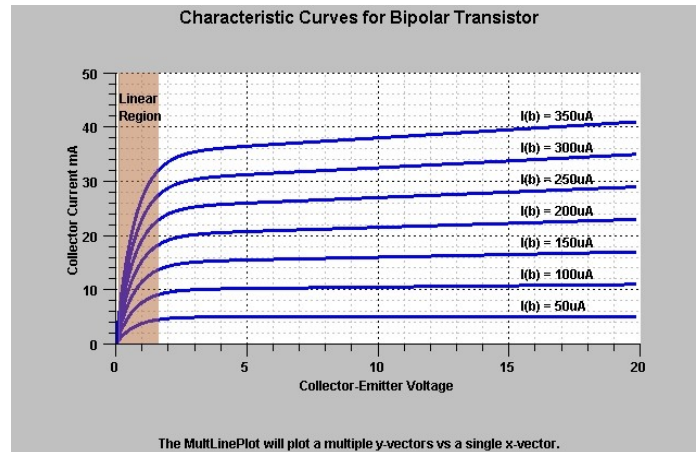
HistogramPlot

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.



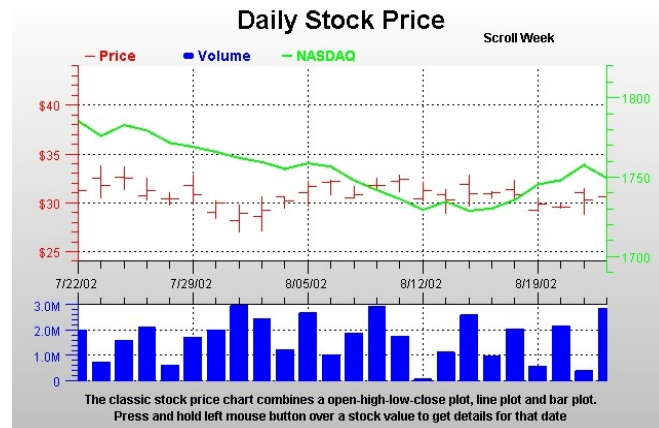
LineGapPlot

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.



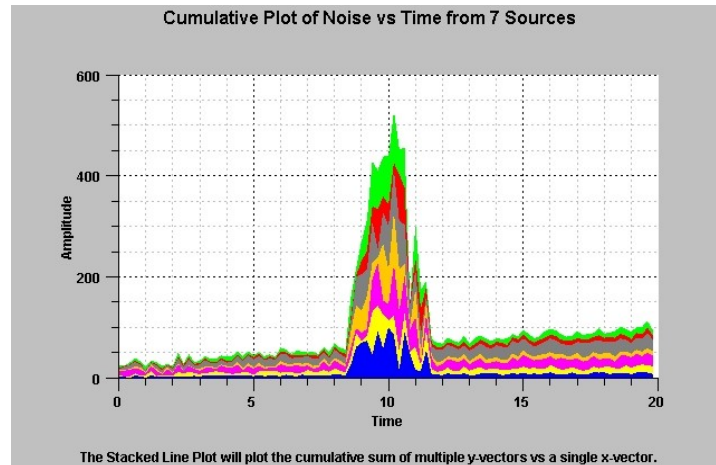
MultiLinePlot

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

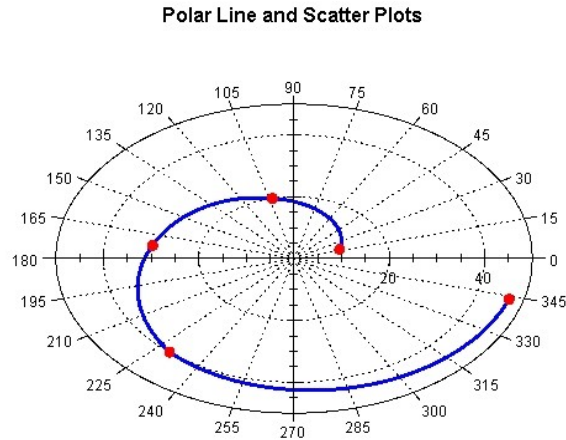
PolarLinePlot

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

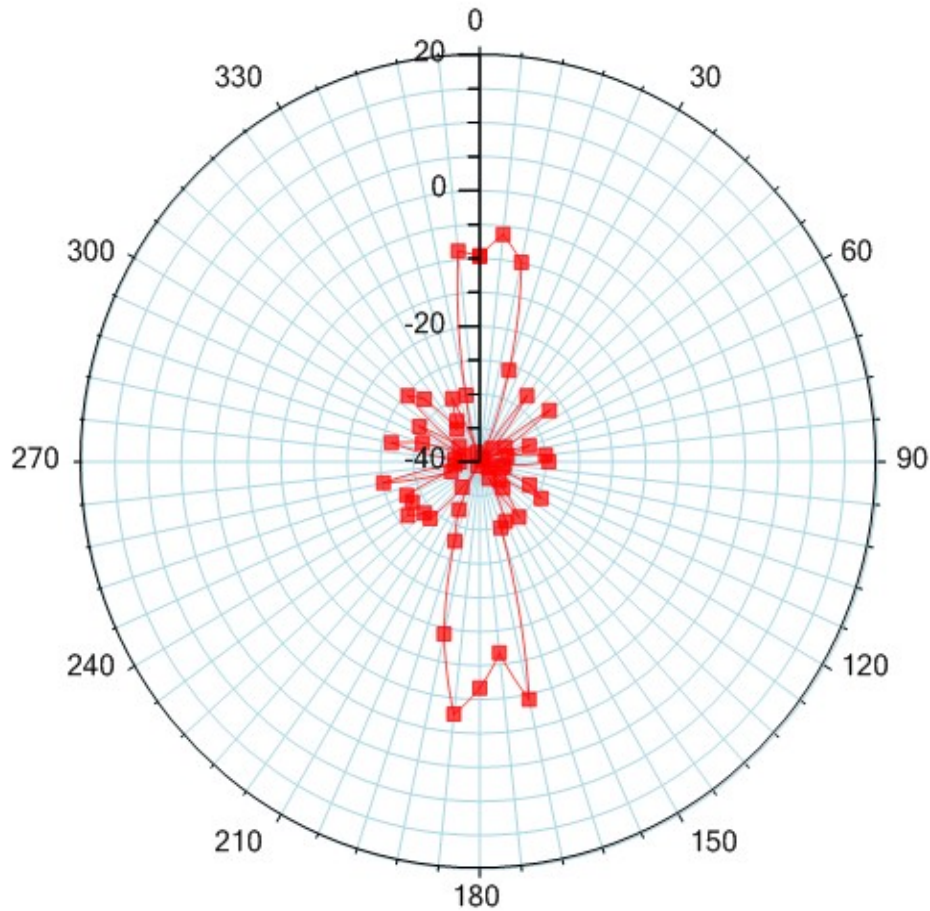
GraphObj

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.

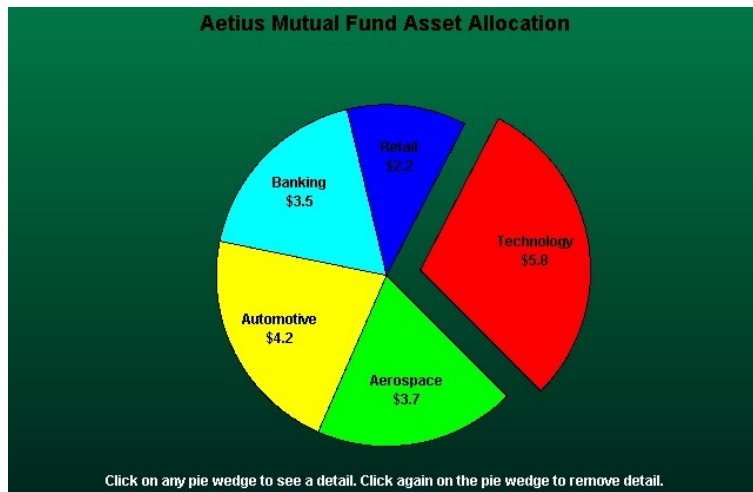


AntennaLineMarkerPlot

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.

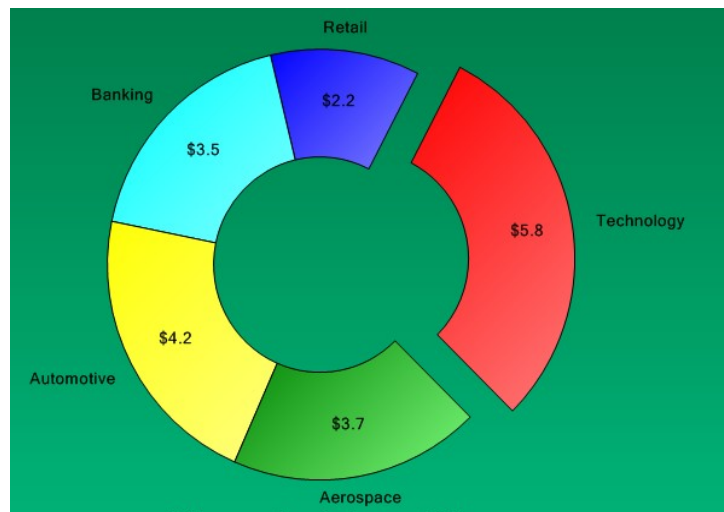
Pie and Ring Chart Classes

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 chart. 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

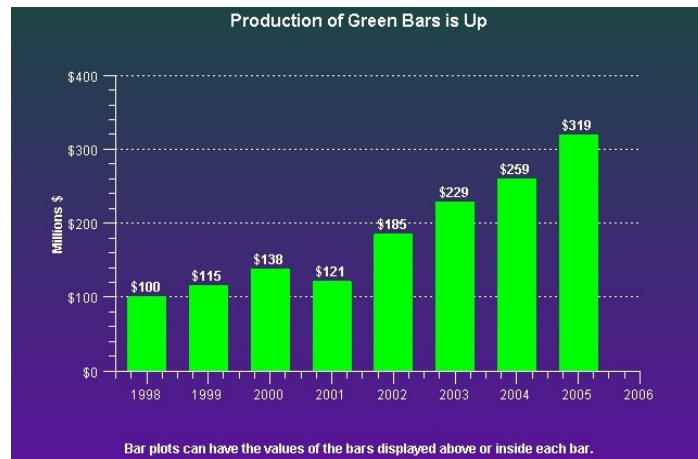
SimpleScatterPlot

SimpleVeraPlot

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.

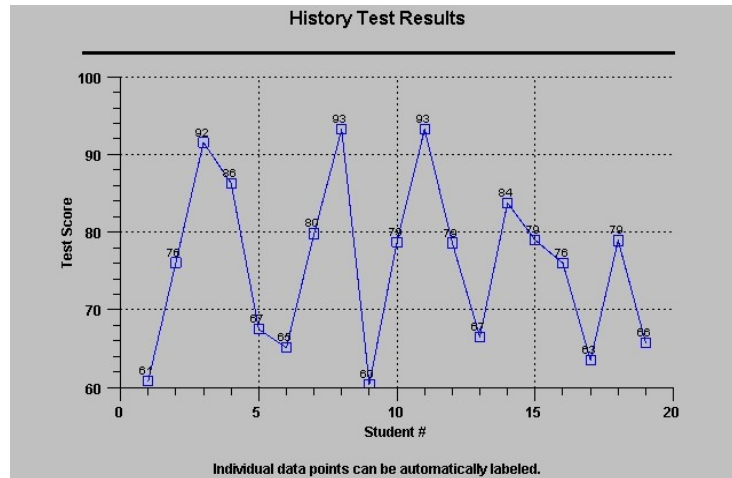
SimplePlot

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



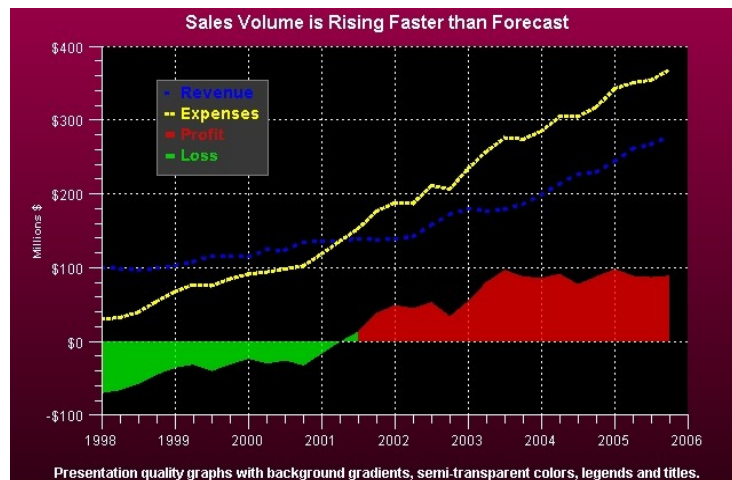
SimpleBarPlot

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.



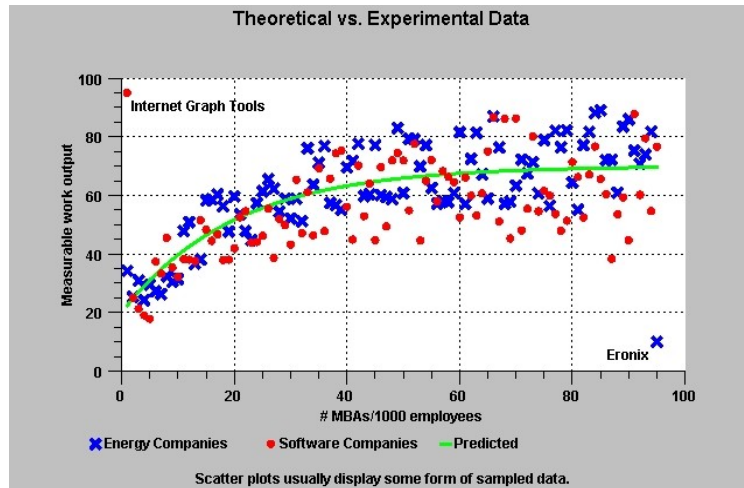
SimpleLineMarkerPlot

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** 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 **Date** 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 a HTMLImageElement class, defining a rectangle in chart coordinates that the image is placed in. JPEG and other image files can be imported class and displayed in a chart.
ChartShape	This class encapsulates a drawing path (GPath) class, placing the shape in a chart using a position defined in chart coordinates. A chart can display any object that can be defined using QCRTGraph GPath class.
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

MagniView
MoveCoordinates
ChartZoom

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

MouseListener	This class implements JavaScript/TypeScript delegates that trap generic mouse 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 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

MagniView

This class implements mouse controlled magnification for one or more simultaneous axes. This class implements a chart magnify class based on the **MouseListener** class. It uses two charts; the source chart and the target chart. The source chart displays the chart in its unmagnified state. The target chart displays the chart in the magnified state. The mouse positions a **MagniView** rectangle within the source chart, and the target chart is re-scaled and redrawn to match the extents of the **MagniView** rectangle from the source chart.

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.

File and Printer Rendering Classes

BufferedImage

BufferedImage

This class will convert a **ChartView** object to a JavaScript/TypeScript **Image** object. Optionally, the class saves the buffered image to an image file.

Miscellaneous Utility Classes

ChartCalendar

CSV

ChartDimension

ChartPoint2D

GroupPoint2D

DoubleArray

DoubleArray2D

BoolArray

nearestPointData

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) files.
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/TypeScript Point and PointF classes.
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.



96 QCChart2D Class Summary

Background
ChartImage
ChartShape

ChartSymbol
Marker
ChartZoom

4. Process Variable and Alarm Classes

RTProcessVar

RTAlarm

RTAlarmEventArgs

The **RTProcessVar** class is the core data storage object for all of the real-time indicator classes. The **RTProcessVar** class represents a single process variable, complete with limit values, an unlimited number of high and low alarms, historical data storage, and descriptive strings for use in displays.

Indicators that display the current value of a single process variable, the **RTBarIndicator**, the **RTMeterIndicator** and **RTPanelMeter** classes for example, reference back to a single **RTProcessVar** object. Indicators that display the current values of multiple process variables, the **RTMultiBarIndicator**, **RTMultiValueAnnunciator** and **RTFormControlGrid** classes, reference back to a collection of **RTProcessVar** objects. Even though an **RTProcessVar** object is in a multi-value indicator collection with other **RTProcessVar** objects, it maintains its own unique settings for limit values, alarm limits and descriptive strings.

The **RTSimpleSingleValuePlot** and **RTGroupMultiValuePlot** classes provide a link between the **RTProcessVar** class and the charting routines in the **QCChart2D** charting package. The **RTSimpleSingleValuePlot** class combines any of the **QCChart2D SimplePlot** classes with an **RTProcessVar** object, and the **RTGroupMultiValuePlot** class combines any of the **QCChart2D GroupPlot** classes with a collection of **RTProcessVar** objects. The **RTProcessVar** class manages a historical data buffer based on the **QCChart2D ChartDataset** class. Each time the current value of the **RTProcessVar** object is updated, it is time-stamped and its value appended to the internal **ChartDataset**. The time stamp can either be explicitly supplied in the update call, or it can be automatically derived from the system clock. From there it can be plotted in static or scrolling plots.

The **RTProcessVar** class contains a collection of **RTAlarm** objects. Each alarm object represents a unique alarm condition: either a greater than alarm or a less than alarm, based on the specified limit value. The **RTAlarm** class also specifies alarm text strings, alarm colors, and the alarm hysteresis value. An **RTProcessVar** object can hold an unlimited number of **RTAlarm** objects. Every time an **RTProcessVar** object is updated with new values, every alarm is checked and an alarm event is generated if the alarm conditions are met. The programmer can hook into the alarm events using alarm event delegates.

Real-Time Process Variable

Class RTProcessVar

ChartObj

|
+-- **RTPProcessVar**

Real-time data is stored in **RTPProcessVar** classes. The **RTPProcessVar** class is designed to represent a single process variable, complete with limit values, an unlimited number of high and low alarms, historical data storage, and descriptive strings for use in displays. It has two main constructors.

RTPProcessVar constructors

```
public static newRTPProcessVar(tagname: string, defaultattribute: ChartAttribute): RTPProcessVar  
  
public static newRTPProcessVarDataset(dataset: TimeSimpleDataset, defaultattribute:  
ChartAttribute): RTPProcessVar
```

Parameters

tagname

A string representing the tag name of the process variable.

dataset

A dataset that will be used to hold historical values for the process variable. If no tag name is supplied in the constructor the tag name for the process variable will be taken from the **ChartDataset.DataName** property of the dataset.

defaultattribute

Specifies the default attributes for the process variable.

Once created, the **RTPProcessVar** object is updated using the **setCurrentValue** method. The method has several overloads:

```
public setCurrentValue(timestamp: number, pv: number)  
public setCurrentValueNumber(pv: number)  
public setCurrentValueTimeSpan(ts: ChartTimeSpan, pv: number)  
public setCurrentValueDate(gv: Date, pv: number)
```

Parameters

gv

The time stamp as a **Date** object for the process variable.

ts

The time stamp as a **ChartTimeSpan** object for the process variable.

timestamp

The time stamp, in milliseconds, for the process variable.

pv

The value of the process variable.

If the time stamp value is not explicitly provided, as in the case of the last overloaded method, the current time as stored in the system clock is used as the time stamp.

Alarms are added to an **RTProcessVar** object using the **RTProcessVar.addAlarm** or **RTProcessVar.addCloneAlarm** methods. The **addCloneAlarm** method clones the passed in alarm object, so that the same **RTAlarmObject** can be used to initialize multiple **RTProcessVar** objects without a conflict occurring.

```
public addAlarm(alarmobj: RTAlarm)
public addCloneAlarm(alarmobj: RTAlarm): RTAlarm | null
```

Parameters

alarmobj

A reference to the **RTAlarm** object that is to be added to the process variables alarm list.

Configure the **RTAlarm** object and then add it to the **RTProcessVar** object. If you plan to use the **RTAlarm** event handlers, make sure that you create a unique **RTAlarm** object for every alarm added to a **RTProcessVar** object.

The most commonly used **RTProcessVar** properties are:

Selected Public Instance Properties

[AlarmStateEventEnable](#)

Get/Set the flag for the alarm state event enable. Set to true to enable alarm checking.

[AlarmTransitionEventEnable](#)

Get/Set the flag for the **AlarmTransitionEventHandler** delegate enable. Set to true to enable the **AlarmTransitionEventHandler** delegate.

[CurrentValue](#)

Get the process variable current value.

[DatasetEnableUpdate](#)

Get/Set to true to enable historical data collection in the process variable dataset.

[DefaultAttribute](#)

Get/Set the default attributes for the process variable.

[DefaultMaximumDisplayValue](#)

Get/Set maximum allowable display value for the process variable.

[DefaultMinimumDisplayValue](#)

Get/Set minimum allowable display value for the process variable.

[DetailedDescription](#)

Get/Set the process variable detailed description string.

[GoodValue](#)

Get/Set set to false designates that the current value is a bad value.

[MaximumValue](#)

Get/Set maximum allowable value for the process variable.

[MinimumValue](#)

Get/Set minimum allowable value for the

<u>PrevCurrentValue</u>	process variable. Get the process variable previous current value.
<u>PrevTimeStamp</u>	Get the process variable previous time stamp value.
<u>ProcessVarDataset</u>	Get/Set the process variable dataset.
<u>ShortDescription</u>	Get/Set the process variable short description string.
<u>TagName</u>	Get/Set the process variable tag string.
<u>TimeStamp</u>	Get the process variable time stamp value.
<u>UniqueIdentifier</u>	Get/Set the process variable unique identifier string.
<u>UnitsString</u>	Get/Set the process variable units string.

Public Instance Events

<u>AlarmStateEventHandler</u>	Delegate for notification each time the check of a process variable produces an alarm state condition.
<u>AlarmTransitionEventHandler</u>	Delegate for notification each time the check of a process variable produces a change of state in alarm state condition.

A complete listing of RTPProcessVar properties is found in the `_qcrgraphTS_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example of Creating an RTPProcessVar Object

The example below creates and updates an RTPProcessVar object that does not use alarms. The example was extracted from the Treadmill example program, method `InitializeGraph`. See the example in the RTAlarm section of the manual for one that uses alarms.

[TypeScript]

```
METSValue: number = 0;
METS: QCRTGraphTS.RTPProcessVar = new QCRTGraphTS.RTPProcessVar();
METSCumulativeValue: number = 0;
METSCumulative: QCRTGraphTS.RTPProcessVar = new QCRTGraphTS.RTPProcessVar();
.
.
.

let defaultattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN);

this.METSValue = 0;
this.METS = QCRTGraphTS.RTPProcessVar.newRTPProcessVar("METS", defaultattrib);
this.METS.ShortDescription = "Metabolic Equivalents";
this.METS.MinimumValue = 0;
this.METS.MaximumValue = 100;
```

```

this.METS.DefaultMinimumDisplayValue = 0;
this.METS.DefaultMaximumDisplayValue = 100;
this.METS.DatasetEnableUpdate = true;
// Make sure this is on for scrolling graphs
this.METS.setCurrentValue(0, this.METSValue);
this.METSCumulativeValue = 0;
this.METSCumulative = QCRTGraphTS.RTProcessVar.newRTProcessVar("METS Cumulative", defaultattrib);
this.METSCumulative.MinimumValue = 0;
this.METSCumulative.MaximumValue = 9999;
this.METSCumulative.DefaultMinimumDisplayValue = 0;
this.METSCumulative.DefaultMaximumDisplayValue = 999;
this.METSCumulative.setCurrentValueNumber(this.METSCumulativeValue);
.
.
.
this.METSValue = Math.max(0, (this.heartRateValue - 60) / (60.0));
this.METS.setCurrentValue(this.currentElapsedMSecs, this.METSValue);

```

[JavaScript]

```

var METSValue = 0;
var METS = new QCRTGraphTS.RTProcessVar();
var METSCumulativeValue = 0;
var METSCumulative = new QCRTGraphTS.RTProcessVar();
.
.
.

let defaultattrib = QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.GREEN,
1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN);

METSValue = 0;
METS = QCRTGraphTS.RTProcessVar.newRTProcessVar("METS", defaultattrib);
METS.ShortDescription = "Metabolic Equivalents";
METS.MinimumValue = 0;
METS.MaximumValue = 100;
METS.DefaultMinimumDisplayValue = 0;
METS.DefaultMaximumDisplayValue = 100;
METS.DatasetEnableUpdate = true;
// Make sure this is on for scrolling graphs
METS.setCurrentValue(0, METSValue);
.
.
.
METSValue = Math.max(0, (heartRateValue - 60) / (60.0));
METS.setCurrentValue(currentElapsedMSecs, METSValue);

```

Real-Time Alarms

Class RTAlarm

ChartObj

```

|
+-- RTAlarm

```

The **RTAlarm** class stores alarm information for the **RTProcessVar** class. The **RTAlarm** class specifies the type of the alarm, the alarm color, alarm text messages and alarm hysteresis value. The **RTProcessVar** classes can hold an unlimited number of **RTAlarm** objects in an internal **Array**.

RTAlarm constructors

```
public static newRTAlarm(processvar: RTProcessVar, alarmtype: number, alarmlimitvalue: number): RTAlarm

public static newRTAlarmProcessVar(processvar: RTProcessVar, alarmtype: number): RTAlarm

public static newRTAlarmAlarmType(alarmtype: number, alarmlimitvalue: number): RTAlarm

public static newRTAlarmProcessVarAlarmType(processvar: RTProcessVar, alarmtype: number, alarmlimitvalue: number): RTAlarm

public static newRTAlarmProcessVarAlarmTypeMessage(processvar: RTProcessVar, alarmtype: number, alarmlimitvalue: number, normalmessage: string, alarmmessage: string): RTAlarm

public static newRTAlarmProcessVarAlarmTypeMessageHysteresis(processvar: RTProcessVar, alarmtype: number, alarmlimitvalue: number, normalmessage: string, alarmmessage: string, hysteresisvalue: number): RTAlarm
```

Parameters

processvar

Specifies the process variable that the alarm is for. When you add an **RTAlarm** object to an **RTProcessVar** object, this field will be overwritten with a reference to the **RTProcessVar** object that was added to.

alarmtype

Specifies the alarm type: **RT_ALARM_NONE**, **RT_ALARM_LOWER_THAN**, or **RT_ALARM_GREATER_THAN**.

alarmlimitvalue

Specifies the alarm limit value.

normalmessage

Specifies the message displayed when there is no alarm.

alarmmessage

Specifies the alarm message.

hysteresisvalue

Specifies the hysteresis value of the alarm. This is used to prevent alarms from toggling between states due to noise in the system when the process variable is very close to an alarm threshold. After an alarm has been triggered, the process variable must cross the alarm threshold in the opposite direction by the hysteresis value before it falls out of alarm. For example, if an **RT_ALARM_GREATER_THAN** alarm threshold is 70, then the process value will always go into alarm once the threshold value of 70 is exceeded. If the hysteresis value is 2, then the process variable will not fall out of alarm until the process value is less than (alarmlimitvalue – hysteresisvalue) = (70 – 2) = 68. If you don't want hysteresis set it equal to 0.0.

The most commonly used **RTAlarm** properties are:

Selected Public Instance Properties

<u>AlarmLimitValue</u>	Get/Set the alarm limit value.
<u>AlarmMessage</u>	Get/Set the current alarm message.
<u>AlarmState</u>	Get/Set the alarm state, true if the last call to CheckAlarm show that the process variable currently in alarm.
<u>AlarmSymbolColor</u>	Get/Set the alarm symbol color.
<u>AlarmTextColor</u>	Get/Set the alarm text color.
<u>AlarmType</u>	Get/Set the alarm type: RT_ALARM_NONE, RT_ALARM_LOWERTHAN, or RT_ALARM_GREATERTHAN.
<u>HysteresisValue</u>	Get/Set the alarm hysteresis value.

A complete listing of RTAlarm properties is found in the `_qcrgraph.ts_.html` documentation file, located in the `QCRTGraphTS/docs/modules` subdirectory.

Once an **RTAlarm** object is added to an **RTProcessVar** object, alarm checking takes place every time the **RTProcessVar.setCurrentValue** method is called. Any displays dependent on the alarm will not change until the **ChartView.updateDraw** method is called, forcing a repaint of the view.

Example of RTAlarm objects added to an RTProcessVar object

The example below creates and updates an **RTProcessVar** object that uses alarms. It does not however generate alarm events that notify user-defined alarm handlers. The example was extracted from the Treadmill example program, method `InitializeGraph`. See the example in the **RTAlarmEventArgs** section for one that generates alarm events.

[TypeScript]

```
let defaultattrib: QCRTGraphTS.ChartAttribute =
  QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 1,
  QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN);

let lowheartratealarm: QCRTGraphTS.RTAlarm =
  QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_LOWERTHAN, 30);
lowheartratealarm.AlarmMessage = "Low Heart Rate";
lowheartratealarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.BLUE;
lowheartratealarm.AlarmTextColor = QCRTGraphTS.ChartColor.BLUE;
let highheartratealarm: QCRTGraphTS.RTAlarm =
  QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_GREATERTHAN, 160);
highheartratealarm.AlarmMessage = "High Heart Rate";
highheartratealarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.RED;
highheartratealarm.AlarmTextColor = QCRTGraphTS.ChartColor.RED;

this.heartRate = QCRTGraphTS.RTProcessVar.newRTProcessVar("Heart Rate", defaultattrib);
```

104 Process Variable and Alarm Classes

```
this.heartRate.MinimumValue = 0;
this.heartRate.MaximumValue = 300;
this.heartRate.DefaultMinimumDisplayValue = 0;
this.heartRate.DefaultMaximumDisplayValue = 200;
this.heartRate.addAlarm(lowheartratealarm);
this.heartRate.addAlarm(highheartratealarm);

this.heartRate.setCurrentValueNumber(this.heartRateValue);
.
.
.
this.heartRateValue = 60.0 + this.runnersPaceValue * 10.0 +
this.heartRate.setCurrentValueNumber(this.heartRateValue);
```

[JavaScript]

```
let defaultattrib = QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.GREEN,
1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN);

let lowheartratealarm =
QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_LOWERTHAN, 30);
lowheartratealarm.AlarmMessage = "Low Heart Rate";
lowheartratealarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.BLUE;
lowheartratealarm.AlarmTextColor = QCRTGraphTS.ChartColor.BLUE;
let highheartratealarm =
QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_GREATERTHAN, 160);
highheartratealarm.AlarmMessage = "High Heart Rate";
highheartratealarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.RED;
highheartratealarm.AlarmTextColor = QCRTGraphTS.ChartColor.RED;

heartRate = QCRTGraphTS.RTProcessVar.newRTProcessVar("Heart Rate", defaultattrib);
heartRate.MinimumValue = 0;
heartRate.MaximumValue = 300;
heartRate.DefaultMinimumDisplayValue = 0;
heartRate.DefaultMaximumDisplayValue = 200;
heartRate.addAlarm(lowheartratealarm);
heartRate.addAlarm(highheartratealarm);
heartRate.AlarmTransitionEventEnable = true;

heartRate.setCurrentValueNumber(heartRateValue);
.
.
.
heartRateValue = 60.0 + runnersPaceValue * 10.0 + treadmillElevationValue * 3.0;
heartRate.setCurrentValueNumber(heartRateValue);
```

Real-Time Alarms Event Handling

Class RTAlarmEventArgs

ChartObj

|
+-- RTAlarmEventArgs

The **RTProcessVar** class can throw an alarm event based on either the current alarm state, or an alarm transition from one alarm state to another. The **RTAlarmEventArgs** class is used to pass alarm data to the event handler. If you want the alarm event to be called only on the initial transition from the no-

alarm state to the alarm state, set the **RTProcessVar**. [AlarmTransitionEventEnable](#) to true and the **RTProcessVar.AlarmStateEventEnable** to false. In this case you will get one event when the process variable goes into alarm, and one when it comes out of alarm. If you want a continuous stream of alarm events, as long as the **RTAlarm** object is in alarm, set the **RTProcessVar**.

[AlarmTransitionEventEnable](#) to false and the **RTProcessVar.AlarmStateEventEnable** to true. The alarm events will be generated at the same rate as the **RTProcessVar.setCurrentValue()** method is called,

RTAlarmEventArgs constructors

You don't really need the constructors since **RTAlarmEventArgs** objects are created inside the **RTProcessVar** class when an alarm event needs to be generated. Here it is anyway.

```
public static newRTAlarmEventArgs(pv: RTProcessVar, alarm: RTAlarm, channel: number):
RTAlarmEventArgs
```

Parameters

pv

The **RTProcessVar** object associated with the alarm event.

alarm

The **RTAlarm** object associated with the alarm event.

channel

The channel number associated with the alarm event.

The most commonly used **RTAlarmEventArgs** properties are:

Selected Public Instance Properties

[AlarmChannel](#)

Get/Set the alarm channel object.

[EventAlarm](#)

Get/Set the **RTAlarm** object.associated with the alarm

[ProcessVar](#)

Get/Set the **RTProcessVar** object. associated with the alarm

A complete listing of **RTAlarmEventArgs** properties is found in the `_qcrgraphTS_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example

[TypeScript]

Setup and enable an alarm transition event handler in the following manner:

```
this.heartRate.AlarmTransitionEventEnable = true;
this.heartRate.AlarmTransitionEventHandler = (source: any, e: QCRTGraphTS.RTAlarmEventArgs) => {
    this.heartRate_HighAlarm(this.heartRate, e);
}
```

where the handler method is **this.heartRate_HighAlarm**.

```
heartRate_HighAlarm(sender: any, e: QCRTGraphTS.RTAlarmEventArgs) {
    let pv: QCRTGraphTS.RTProcessVar | null = e.ProcessVar;
}
```

Setup and enable an alarm state event handler in an identical manner:

```
this.heartRate.AlarmStateEventEnable = true;
this.heartRate.AlarmStateEventHandler = (source: any, e: QCRTGraphTS.RTAlarmEventArgs) => {
    this.heartRate_HighAlarm(this.heartRate, e);
}
```

where the handler method is **this.heartRate_HighAlarm**.

```
heartRate_HighAlarm(sender: any, e: QCRTGraphTS.RTAlarmEventArgs) {
    let pv: QCRTGraphTS.RTProcessVar | null = e.ProcessVar;
}
```

Example of RTProcessVar, RTAlarm and Alarm Event Handlers

The example below creates and updates an **RTProcessVar** object that uses alarms and a user-defined alarm event handler. The example was extracted from the Treadmill example program, method **InitializeGraph**. See the example in the **RTAlarmEventArgs** section for one that generates alarm events.

[TypeScript]

```
let defaultattrib: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN);

let lowheartratealarm: QCRTGraphTS.RTAlarm =
    QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_LOWERTHAN, 30);
lowheartratealarm.AlarmMessage = "Low Heart Rate";
lowheartratealarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.BLUE;
lowheartratealarm.AlarmTextColor = QCRTGraphTS.ChartColor.BLUE;
let highheartratealarm: QCRTGraphTS.RTAlarm =
    QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_GREATERTHAN, 160);
highheartratealarm.AlarmMessage = "High Heart Rate";
highheartratealarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.RED;
highheartratealarm.AlarmTextColor = QCRTGraphTS.ChartColor.RED;

this.heartRate = QCRTGraphTS.RTProcessVar.newRTProcessVar("Heart Rate", defaultattrib);
this.heartRate.MinimumValue = 0;
this.heartRate.MaximumValue = 300;
this.heartRate.DefaultMinimumDisplayValue = 0;
this.heartRate.DefaultMaximumDisplayValue = 200;
this.heartRate.addAlarm(lowheartratealarm);
this.heartRate.addAlarm(highheartratealarm);

this.heartRate.AlarmTransitionEventEnable = true;
this.heartRate.AlarmTransitionEventHandler = (source: any, e: QCRTGraphTS.RTAlarmEventArgs) => {
    this.heartRate_HighAlarm(this.heartRate, e);
}
```



```

this.heartRate.setCurrentValueNumber(this.heartRateValue);
.
.
.
this.heartRateValue = 60.0 + this.runnersPaceValue * 10.0 +
this.heartRate.setCurrentValueNumber(this.heartRateValue);
.
.
.

heartRate_HighAlarm(sender: any, e: QCRTGraphTS.RTAlarmEventArgs) {
  let pv: QCRTGraphTS.RTProcessVar | null = e.ProcessVar;
}

```

[JavaScript]

```

let defaultattrib = QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.GREEN,
1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN);

let lowheartratealarm =
QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_LOWERTHAN, 30);
lowheartratealarm.AlarmMessage = "Low Heart Rate";
lowheartratealarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.BLUE;
lowheartratealarm.AlarmTextColor = QCRTGraphTS.ChartColor.BLUE;
let highheartratealarm =
QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_GREATERTHAN, 160);
highheartratealarm.AlarmMessage = "High Heart Rate";
highheartratealarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.RED;
highheartratealarm.AlarmTextColor = QCRTGraphTS.ChartColor.RED;

heartRate = QCRTGraphTS.RTProcessVar.newRTProcessVar("Heart Rate", defaultattrib);
heartRate.MinimumValue = 0;
heartRate.MaximumValue = 300;
heartRate.DefaultMinimumDisplayValue = 0;
heartRate.DefaultMaximumDisplayValue = 200;
heartRate.addAlarm(lowheartratealarm);
heartRate.addAlarm(highheartratealarm);

heartRate.AlarmTransitionEventEnable = true;
heartRate.AlarmTransitionEventHandler = (source, e) => {
  heartRate_HighAlarm(heartRate, e);
}

heartRate.setCurrentValueNumber(heartRateValue);
.
.
.
heartRateValue = 60.0 + runnersPaceValue * 10.0 + treadmillElevationValue * 3.0;
heartRate.setCurrentValueNumber(heartRateValue);
.
.
.

async function heartRate_HighAlarm(sender, e) {
  let pv = e.ProcessVar;
}

.
.
.

```

5. Panel Meter Classes

RTNumericPanelMeter

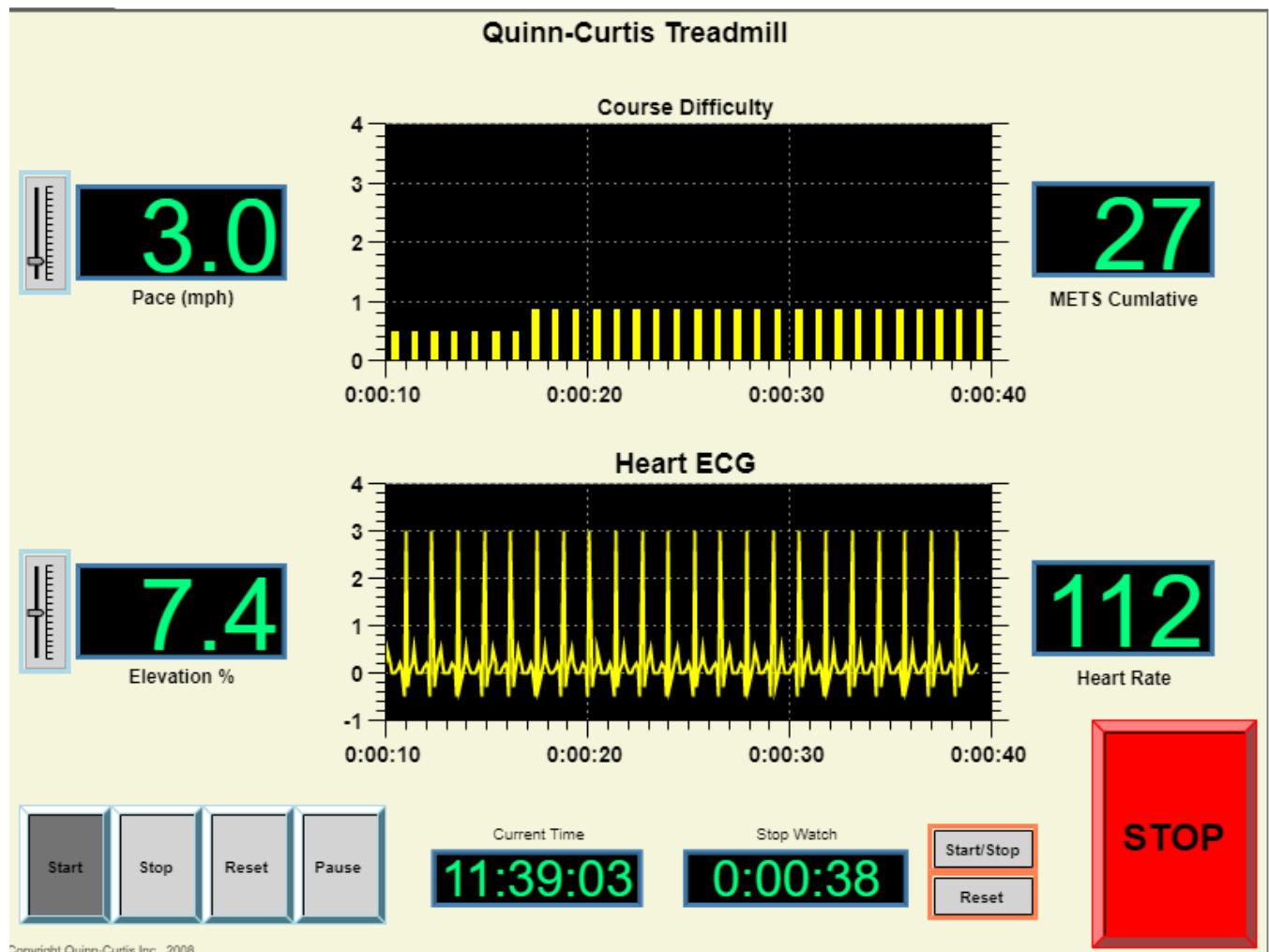
RTAlarmPanelMeter

RTStringPanelMeter

RTTimePanelMeter

RTElapsedTimePanelMeter

RTFormControlPanelMeter



The **RTPanelMeter** derived classes are special cases of the single value indicator classes that are used throughout the software to display real-time data in a text format. Panel meters are available for numeric values, string values, time/date values and alarm values. All of the panel meter classes have a great many options for controlling the text font, color, size, border and background of the panel meter rectangle. **RTPanelMeter** objects are used in two ways. First, they can be standalone, and once attached to an **RTProcessVar** object they can be added to a **ChartView** as any other **GraphObj** derived class. Second,

they can be attached to most of the single channel and multiple channel indicators, such as **RTBarIndicator**, **RTMultiBarIndicator**, **RTMeterIndicator** and **RTAnnunciator** objects, where they provide text output in addition to the indicators graphical output.

LED/LCD Style Font

In our example programs the panel meters often use a simple 7-segment or 14 segment display font. It gives the displays an anachronistic look typical of older, dedicated instruments. These fonts are not found in the standard collection of fonts used by web browsers. But, you can find implementations online, which is what we use when needed.



The example above is from our MiniScope example. First you need to get the browser page to download the font family from some online source. The line below downloads the segment7 font from the online source fontlibrary.org.

```
<link rel="stylesheet" media="screen" href="https://fontlibrary.org/face/segment7"
type="text/css"/>
```

The link is placed in the header of the HTML page which displays graphs created using our software. See the MiniScope.html file in the MiniScope example. This places it in the global namespace of the HTML page, and you can then reference fonts contained within the font family in the JavaScript/TypeScript modules which are part of that page. In this case, the font family from fontlibrary.org named segment7 contains only one font, named *Segment7Standard*. See: <https://fontlibrary.org/en/font/segment7> . So *Segment7Standard* is the font name you use when creating one of the ChartFont objects used to manage fonts used in our graphs.

Next, you define the font in your JavaScript/TypeScript code using the segment7standard name.

[TypeScript]

```
font48Numeric: QCRTGraphTS.ChartFont =
QCRTGraphTS.ChartFont.newChartFont("Segment7Standard", QCRTGraphTS.ChartFont.REGULAR, 48);
```

[JavaScript]

```
font48Numeric = QCRTGraphTS.ChartFont.newChartFont("Segment7Standard", QCRTGraphTS.ChartFont.REGU
LAR, 48);
```

See the MiniScope.js or MiniScope.ts file for an example

The segment7 font used above has an annoying limitation. It doesn't represent colons and uses a dash in its place. This doesn't represent time-of-day all that well. There is another LED/LCD-style font that we use in some places where we want a time-of-day colon. It is also from the fontlibrary.org collection and

the font family is named lcd. See <https://fontlibrary.org/en/font/lcd> . The example below is the display of the WeatherStation example. It uses the lcd font for the Inside Temp display, and the Current Time display.



In the JavaScript/TypeScript code the associated ChartFont object is initiated using the font name: lcd14.

[TypeScript]

```
font42Numeric: QCRTGraphTS.ChartFont = QCRTGraphTS.ChartFont.newChartFont("lcd14", QCRTGraphTS.Ch
artFont.REGULAR, 42);
```

[JavaScript]

```
font42Numeric: QCRTGraphTS.ChartFont = QCRTGraphTS.ChartFont.newChartFont("lcd14", QCRTGraphTS.Ch
artFont.REGULAR, 42);
```

You see that while it displays the colon in the Current Time fine, the decimal point it uses in the Inside Temp is elongated. Perhaps it is a trade-off between the US-style decimal point (a period), and the European style decimal point (a comma). There are other LED/LCD style fonts available so you may be able to find better ones for your specific applications.

Panel Meters

Class RTPanelMeter

ChartPlot

RTPlot

RTSingleValueIndicator

RTPanelMeter

The **RTPanelMeter** is an abstract class for the other panel meter classes. While it cannot be instantiated, it does contain properties and methods common to all panel meters. A summary of these properties is listed below.

Selected Public Instance Properties

AlarmIndicatorColorMode (inherited from RTSingleValueIndicator)	Get/Set whether the color of the indicator objects changes on an alarm. Use one of the constants: RT_INDICATOR_COLOR_NO_ALARM_CHANGE, RT_INDICATOR_COLOR_CHANGE_ON_ALARM..
ContrastTextAlarmColor (inherited from RTPanelMeter)	Set/Get a contrast color to use for text when the object is in alarm, and the background color of the panel meter changes.
CurrentProcessValue (inherited from RTSingleValueIndicator)	Get the current process value of the primary channel.
Frame3DEnable (inherited from RTPanelMeter)	Set/Get to true to enable a 3D frame for the panel meter.
PanelMeterNudge (inherited from RTPanelMeter)	Set/Get the xy values of the PanelMeterNudge property. The PanelMeterNudge property moves the relative position, using window device coordinates, of the text relative to the specified location of the text.
PanelMeterPosition (inherited from RTPanelMeter)	Set/Get the panel meter position value. Use one of the panel meter position constants. See table for positioning constants.
PositionReference (inherited from RTPanelMeter)	Set/Get an RTPanelMeter object used as a positioning reference for this RTPanelMeter object.
PrimaryChannel (inherited from RTPlot)	Set/Get the primary channel of the indicator.
RTDataSource (inherited from RTSingleValueIndicator)	Get/Set the array list holding the RTProcessVar variables for the indicator.
RTPlotObj (inherited from RTPanelMeter)	Set/Get the reference RTPlot object.
TextColor	Set/Get the text color of the panel meter.

A complete listing of **RTPanelMeter** properties is found in the `_qcrgraphts_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Positioning Panel Meters

The most complicated thing about panel meters is getting them positioned where you want. There are over 30 positioning constants that can be used to position panel meters with respect to graph objects, the plot area, and graph area of the associated graph. In addition to the positioning constants, you can explicitly place the panel meter anywhere that you want in a graph using the `CUSTOM_POSITION` position constant in conjunction with the **RTPanelMeter.setLocation** method. The table below summarizes the panel meter positioning constants used in the software.

Positioning Constants

CUSTOM_POSITION	Custom position specified using the RTPanelMeter.setLocation method. Position can be set using the DEV_POS, PHYS_POS, NORM_GRAPH_POS, NORM_PLOT_POS coordinate systems. Set the justification of the panel meter box using the StringLabel or NumericLabel template of the specific panel meter class.
CENTERED_BAR	Used when the panel meter is attached to a bar indicator. Centers the panel meter inside the bar indicator. If the object is not a bar indicator the panel meter is centered inside the plotting area. Text justification is set to (JUSTIFY_CENTER, JUSTIFY_CENTER)
OUTSIDE_BAR	Used when the panel meter is attached to a bar indicator. Places the panel meter on the outside edge of the bar indicator. If the object is not a bar indicator the panel meter is placed on the outside edge of the plotting area maximum. Text justification depends on the bar orientation: Vertical Bars (JUSTIFY_CENTER, JUSTIFY_MIN), Horizontal Bars (JUSTIFY_MIN, JUSTIFY_CENTER).
INSIDE_BAR	Used when the panel meter is attached to a bar indicator. Places the panel meter on the inside edge of the bar indicator. If the object is not a bar indicator the panel meter is placed on the inside edge of the plotting area maximum. Text justification depends on the bar orientation: Vertical Bars (JUSTIFY_CENTER, JUSTIFY_MAX), Horizontal Bars (JUSTIFY_MAX, JUSTIFY_CENTER).
INSIDE_BARBASE	Used when the panel meter is attached to a bar indicator. Places the panel meter on the inside edge of the bar base of the indicator. If the object is not a bar indicator the panel meter is placed on the inside edge of the plotting area minimum. Text justification depends on the bar orientation: Vertical Bars (JUSTIFY_CENTER, JUSTIFY_MIN), Horizontal Bars (JUSTIFY_MIN, JUSTIFY_CENTER).
OUTSIDE_BARBASE	Used when the panel meter is attached to a bar indicator. Places the panel meter on the outside edge of the bar base of the indicator. If the object is not a bar indicator the panel meter is placed on the outside edge of the plotting area maximum. Text justification depends on the bar orientation: : Vertical Bars (JUSTIFY_CENTER, JUSTIFY_MAX), Horizontal Bars (JUSTIFY_MAX, JUSTIFY_CENTER).
INSIDE_INDICATOR	Same as INSIDE_BAR.

OUTSIDE_INDICATOR	Same as OUTSIDE_BAR.
BELOW_REFERENCED_TEXT	Used when it is desired that the panel meter be positioned next to another object. Places the panel meter below the reference object. Specify the position reference object using the PanelMeter.setPositionReference method. Text justification is set to (JUSTIFY_CENTER, JUSTIFY_MAX).
ABOVE_REFERENCED_TEXT	Used when it is desired that the panel meter be positioned next to another object. Places the panel meter above the reference object. Specify the position reference object using the PanelMeter.setPositionReference method. Text justification is set to (JUSTIFY_CENTER, JUSTIFY_MIN).
RIGHT_REFERENCED_TEXT	Used when it is desired that the panel meter be positioned next to another object. Places the panel meter to the right of the reference object. Specify the position reference object using the PanelMeter.setPositionReference method. Text justification is set to (JUSTIFY_MIN, JUSTIFY_CENTER).
LEFT_REFERENCED_TEXT	Used when it is desired that the panel meter be positioned next to another object. Places the panel meter to the left of the reference object. Specify the position reference object using the PanelMeter.setPositionReference method. Text justification is set to (JUSTIFY_MAX, JUSTIFY_CENTER).
BELOW_CENTERED_PLOTAREA	Positions the panel meter centered, below the plot area. Text justification is set to (JUSTIFY_CENTER, JUSTIFY_MAX).
ABOVE_CENTERED_PLOTAREA	Positions the panel meter centered, above the plot area. Text justification is set to (JUSTIFY_CENTER, JUSTIFY_MIN).
LEFT_CENTERED_PLOTAREA	Positions the panel meter centered, to the left of the plot area. Text justification is set to (JUSTIFY_MAX, JUSTIFY_CENTER).
RIGHT_CENTERED_PLOTAREA	Positions the panel meter centered, to the right of the plot area. Text justification is set to (JUSTIFY_MIN, JUSTIFY_CENTER).
GRAPHAREA_TOP	Positions the panel meter centered, at the top edge of the graph area. Text justification is set to (JUSTIFY_CENTER, JUSTIFY_MAX).
GRAPHAREA_BOTTOM.	Positions the panel meter centered, at the bottom edge of the graph area. Text justification is set to (JUSTIFY_CENTER, JUSTIFY_MIN).

RADIUS_BOTTOM	Used when the panel meter is attached to a meter indicator. Places the panel meter at the bottom, at the radius of the MeterCoordinates system. You can set the text justification however you want.
RADIUS_TOP	Used when the panel meter is attached to a meter indicator. Places the panel meter at the top, at the radius of the MeterCoordinates system. You can set the text justification however you want.
RADIUS_LEFT	Used when the panel meter is attached to a meter indicator. Places the panel meter on the left, at the radius of the MeterCoordinates system. You can set the text justification however you want.
RADIUS_RIGHT	Used when the panel meter is attached to a meter indicator. Places the panel meter on the right, at the radius of the MeterCoordinates system. You can set the text justification however you want.
RADIUS_CENTER	Used when the panel meter is attached to a meter indicator. Places the panel meter at the center of the radius of the MeterCoordinates system. You can set the text justification however you want.
OUTSIDE_RADIUS_BOTTOM	Used when the panel meter is attached to a meter indicator. Places the panel meter at the bottom, centered on the outside edge of the radius of the MeterCoordinates system. Text justification is (JUSTIFY_CENTER, JUSTIFY_MAX).
INSIDE_RADIUS_BOTTOM	Used when the panel meter is attached to a meter indicator. Places the panel meter at the bottom, centered on the inside edge of the radius of the MeterCoordinates system. Text justification is (JUSTIFY_CENTER, JUSTIFY_MIN).
CENTER_RADIUS_BOTTOM	Used when the panel meter is attached to a meter indicator. Places the panel meter at the bottom, centered on the inside edge of the radius of the MeterCoordinates system. Text justification is (JUSTIFY_CENTER, JUSTIFY_CENTER).
OUTSIDE_RADIUS_TOP	Used when the panel meter is attached to a meter indicator. Places the panel meter at the top, centered on the outside edge of the radius of the MeterCoordinates system. Text justification is (JUSTIFY_CENTER, JUSTIFY_MIN).
INSIDE_RADIUS_TOP	Used when the panel meter is attached to a meter indicator. Places the panel meter at the top, centered on the inside edge of the radius of the MeterCoordinates system. Text justification is

	(JUSTIFY_CENTER, JUSTIFY_MAX).
CENTER_RADIUS_TOP	Used when the panel meter is attached to a meter indicator. Places the panel meter at the top, centered on the inside edge of the radius of the MeterCoordinates system. Text justification is (JUSTIFY_CENTER, JUSTIFY_CENTER).

A particularly useful property is **RTPanelMeter.PanelMeterNudge**. After you get the panel meter positioned approximately where you want, you may find that it just a couple of pixels too close to some other object, whether it be an indicator, axis, or text object. You can nudge the panel meter in any direction with respect to its calculated position. The **PanelMeterNudge** property uses device (or pixel) coordinates.

[TypeScript]

```
let panelmetertagattrib: QCRTGraphTS.ChartAttribute =
  QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
let panelmeter3: QCRTGraphTS.RTStringPanelMeter =
  QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(meterframe,
    this.processVar1, panelmetertagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.RADIUS_TOP;
panelmeter3.PanelMeterNudge.Y = -8;
panelmeter3.StringTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
panelmeter3.StringTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
panelmeter3.StringTemplate.TextFont = this.font12Bold;
meterneedle.addPanelMeter(panelmeter3);
this.chartVu.addChartObject(meterneedle);
```

[JavaScript]

```
let panelmetertagattrib =
  QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
let panelmeter3 =
  QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(meterframe,
    processVar1, panelmetertagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.RADIUS_TOP;
panelmeter3.PanelMeterNudge.Y = -8;
panelmeter3.StringTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
panelmeter3.StringTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
panelmeter3.StringTemplate.TextFont = font12Bold;
meterneedle.addPanelMeter(panelmeter3);
chartVu.addChartObject(meterneedle);
```

Numeric Panel Meter

Class RTNumericPanelMeter

ChartPlot

RTPlot

RTSingleValueIndicator
RTPanelMeter
RTNumericPanelMeter

The **RTNumericPanelMeter** class displays the floating point numeric value of an **RTProcessVar** object. It contains a template based on the **NumericLabel** class that is used to specify the font and numeric format information associated with the panel meter.

RTNumericPanelMeter constructors

```
public static newRTNumericPanelMeterCoords(transform: PhysicalCoordinates): RTNumericPanelMeter

public static newRTNumericPanelMeterCoordsAttrib(transform: PhysicalCoordinates, attrib:
ChartAttribute): RTNumericPanelMeter

public static newRTNumericPanelMeter(transform: PhysicalCoordinates, datasource: RTProcessVar,
attrib: ChartAttribute): RTNumericPanelMeter
```

Parameters

transform

The coordinate system for the new **RTNumericPanelMeter** object.

datasource

The process variable associated with the panel meter.

attrib

The color attributes of the panel meter indicator.

Selected Public Instance Properties

NumericTemplate	Set/Get the NumericLabel template for the panel meter numeric value. The text properties associated with the panel meter are set using this property. In addition, the format of the numeric value and the number of digits to the right of the decimal point is also set here.

A complete listing of **RTNumericPanelMeter** properties is found in the `_qcrgraphpts_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example

The panel meter below, extracted from the Treadmill example, method **InitializeRightPanelMeters**, is an independent panel meter that displays the **METSCumulative** process variable. The positioning is accomplished using the **RTPanelMeter.setLocation** method, which in this case places the panel meter using normalized graph coordinates. The size of the panel meter is determined by two things, the font size, which in this case is 64, and the number of digits used in the display. In order to keep the panel

meter box from constantly changing size if the number of digits changes, the panel meter box is sized to accommodate the range of values specified by the **RTProcessVar.DefaultMinimumDisplayValue** and **RTProcessVar.DefaultMaximumDisplayValue**.



[TypeScript]

```
public InitializeRightPanelMeters() {
    let numericfont: QCRTGraphTS.ChartFont = this.font64Numeric;
    let trackbarTitlefont: QCRTGraphTS.ChartFont = this.font12Bold;
    if (!this.chartVu) return;
    let pTransform1: QCRTGraphTS.CartesianCoordinates =
QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);

    let panelmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter1: QCRTGraphTS.RTNumericPanelMeter =
QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(pTransform1, this.METSCumulative,
panelmeterattrib);
    panelmeter1.NumericTemplate.TextFont = numericfont;
    panelmeter1.NumericTemplate.DecimalPos = 0;
    panelmeter1.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
    panelmeter1.setLocationXYType(0.81, 0.285, QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
    this.chartVu.addChartObject(panelmeter1);
    :
    :
}
```

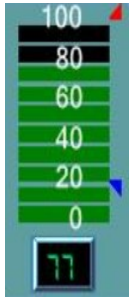
[JavaScript]

```
function InitializeRightPanelMeters() {
    let numericfont = font64Numeric;
    let trackbarTitlefont = font12Bold;
    if (!chartVu) return;
    let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
    let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter1 = QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(pTransform1,
METSCumulative, panelmeterattrib);
    panelmeter1.NumericTemplate.TextFont = numericfont;
    panelmeter1.NumericTemplate.DecimalPos = 0;
    panelmeter1.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
    panelmeter1.setLocationXYType(0.81, 0.285, QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
    chartVu.addChartObject(panelmeter1);
    :
    :
    :
```

Example of RTNumericPanelMeter used with an RTBarIndicator

The panel meter below, extracted from the HybridCar example, method **InitializeBatteryChargeGraph**, is an **RTNumericPanelMeter** attached to a **RTBarIndicator** bar plot object.. The position of the panel meter is **OUTSIDE_PLOTAREA_MIN**, which places underneath the dynamic bar. The size of the panel meter is determined by two things, the font size, which in this case is 14, and the number of digits used in the display.

Note: Unlike the previous example, the panel meter is not added to the **ChartView** (*chartVu* above); instead it is added to the **RTBarIndicator** (*barplot* below). The panel meter will assume the values of the **RTProcessVar** used by the bar plot.



[TypeScript]

```
let barplot: QCRTGraphTS.RTBarIndicator =
  QCRTGraphTS.RTBarIndicator.newRTBarIndicator(pTransform1, this.batteryCharge, barwidth, barbase,
    attrib1, barjust, barorient);
barplot.SegmentSpacing = 10;
barplot.SegmentWidth = 7;
barplot.IndicatorBackground =
  QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.BLACK, 0,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
barplot.SegmentValueRoundMode = QCRTGraphTS.ChartConstants.RT_CEILING_VALUE;
barplot.SegmentCornerRadius = 0;
barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SEGMENTED_SUBTYPE;
let baralarms: QCRTGraphTS.RTAlarmIndicator =
  QCRTGraphTS.RTAlarmIndicator.newRTAlarmIndicatorAxisRTPlot(baraxis, barplot);
this.chartVu.addChartObject(baralarms);
let panelmeterattrib: QCRTGraphTS.ChartAttribute =
  QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let paneltagmeterattrib: QCRTGraphTS.ChartAttribute =
  QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 0,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
let panelmeter: QCRTGraphTS.RTNumericPanelMeter =
  QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeterCoordsAttrib(pTransform1,
    panelmeterattrib);
panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MIN;
panelmeter.TextColor = QCRTGraphTS.ChartColor.SPRINGGREEN;
panelmeter.NumericTemplate.TextFont = this.font14Numeric;
panelmeter.NumericTemplate.DecimalPos = 0;
panelmeter.AlarmIndicatorColorMode =
  QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
panelmeter.PanelMeterNudge = QCRTGraphTS.ChartPoint2D.newChartPoint2D(0, 4);
barplot.addPanelMeter(panelmeter);
```

[JavaScript]

```

let barplot = QCRTGraphTS.RTBarIndicator.newRTBarIndicator(pTransform1,batteryCharge, barwidth,
barbase, attrib1, barjust, barorient);
barplot.SegmentSpacing = 10;
barplot.SegmentWidth = 7;
barplot.IndicatorBackground =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.BLACK, 0,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
barplot.SegmentValueRoundMode = QCRTGraphTS.ChartConstants.RT_CEILING_VALUE;
barplot.SegmentCornerRadius = 0;
barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SEGMENTED_SUBTYPE;
let baralarms = QCRTGraphTS.RTAlarmIndicator.newRTAlarmIndicatorAxisRTPlot(baraxis, barplot);
chartVu.addChartObject(baralarms);
let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let paneltagmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 0,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
let panelmeter = QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeterCoordsAttrib(pTransform1,
panelmeterattrib);
panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MIN;
panelmeter.TextColor = QCRTGraphTS.ChartColor.SPRINGGREEN;
panelmeter.NumericTemplate.TextFont =font14Numeric;
panelmeter.NumericTemplate.DecimalPos = 0;
panelmeter.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
panelmeter.PanelMeterNudge = QCRTGraphTS.ChartPoint2D.newChartPoint2D(0, 4);
barplot.addPanelMeter(panelmeter);artVu.addChartObject(panelmeter1);

```

Alarm Panel Meter

Class RTAlarmPanelMeter

ChartPlot

RTPlot

RTSingleValueIndicator

RTPanelMeter

RTAlarmPanelMeter

The **RTAlarmPanelMeter** class displays the alarm state of an **RTProcessVar** object. It pulls alarm text and color information out of the associated **RTProcessVar** object. It contains a template based on the **StringLabel** class that is used to specify the font and numeric format information associated with the panel meter.

RTAlarmPanelMeter constructors

```

public static newRTAlarmPanelMeterCoords(transform: PhysicalCoordinates): RTAlarmPanelMeter

public static newRTAlarmPanelMeterCoordsAttrib(transform: PhysicalCoordinates, attrib:
ChartAttribute): RTAlarmPanelMeter

public static newRTAlarmPanelMeter(transform: PhysicalCoordinates, datasource: RTProcessVar,
attrib: ChartAttribute): RTAlarmPanelMeter

```

Parameters*transform*The coordinate system for the new **RTAlarmPanelMeter** object.*datasource*

The process variable associated with the panel meter.

attrib

The color attributes of the panel meter indicator.

Selected Public Instance Properties

AlarmTemplate	Get/Set the string template defining the panel meter alarm string format. The text properties associated with the panel meter are set using this property.

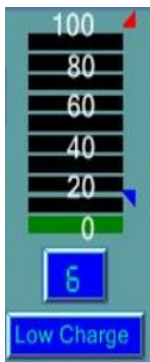
A complete listing of **RTAlarmPanelMeter** properties is found in the `_qcrgraphts_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example of RTAlarmPanelMeter used with RTBarIndicator

The panel meter below, extracted from the HybridCar example, method

InitializeBatteryChargeGraph, adds an **RTAlarmPanelMeter** underneath the numeric panel meter.

Note: The **RTAlarmPanelMeter** uses the `BELOW_REFERENCED_TEXT` positioning constant, and sets the **RTAlarmPanelMeter.setPositionReference** to the numeric panel meter.



[TypeScript]

```
let barplot: QCRTGraphTS.RTBarIndicator =
QCRTGraphTS.RTBarIndicator.newRTBarIndicator(pTransform1, this.batteryCharge, barwidth, barbase,
attrib1, barjust, barorient);
.
.
.
```

```

let panelmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let paneltagmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 0,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
let panelmeter: QCRTGraphTS.RTNumericPanelMeter =
QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeterCoordsAttrib(pTransform1,
palmeterattrib);
panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MIN;
panelmeter.TextColor = QCRTGraphTS.ChartColor.SPRINGGREEN;
panelmeter.NumericTemplate.TextFont = this.font14Numeric;
panelmeter.NumericTemplate.DecimalPos = 0;
panelmeter.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
panelmeter.PanelMeterNudge = QCRTGraphTS.ChartPoint2D.newChartPoint2D(0, 4);
barplot.addPanelMeter(panelmeter);

let panelmeter2: QCRTGraphTS.RTAlarmPanelMeter =
QCRTGraphTS.RTAlarmPanelMeter.newRTAlarmPanelMeterCoordsAttrib(pTransform1, panelmeterattrib);
panelmeter2.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
panelmeter2.TextColor = QCRTGraphTS.ChartColor.SPRINGGREEN;
panelmeter2.AlarmTemplate.TextFont = this.font10;
panelmeter2.setPositionReference(panelmeter);
panelmeter2.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
barplot.addPanelMeter(panelmeter2);
.
.
.
this.chartVu.addChartObject(barplot);

```

[JavaScript]

```

let barplot = QCRTGraphTS.RTBarIndicator.newRTBarIndicator(pTransform1,batteryCharge, barwidth,
barbase, attrib1, barjust, barorient);
.
.
.
let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let paneltagmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 0,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
let panelmeter = QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeterCoordsAttrib(pTransform1,
palmeterattrib);
panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MIN;
panelmeter.TextColor = QCRTGraphTS.ChartColor.SPRINGGREEN;
panelmeter.NumericTemplate.TextFont =font14Numeric;
panelmeter.NumericTemplate.DecimalPos = 0;
panelmeter.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
panelmeter.PanelMeterNudge = QCRTGraphTS.ChartPoint2D.newChartPoint2D(0, 4);
barplot.addPanelMeter(panelmeter);

let panelmeter2 = QCRTGraphTS.RTAlarmPanelMeter.newRTAlarmPanelMeterCoordsAttrib(pTransform1,
palmeterattrib);
panelmeter2.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
panelmeter2.TextColor = QCRTGraphTS.ChartColor.SPRINGGREEN;
panelmeter2.AlarmTemplate.TextFont =font10;
panelmeter2.setPositionReference(panelmeter);
panelmeter2.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
barplot.addPanelMeter(panelmeter2);
.
.
.
chartVu.addChartObject(barplot);

```

String Panel Meter

Class **RTStringPanelMeter**

ChartPlot

RTPlot

RTSingleValueIndicator

RTPanelMeter

RTStringPanelMeter

The **RTStringPanelMeter** class displays a string, either an arbitrary string, or a string based on string data in the associated **RTProcessVar** object. It is usually used to display a channels tag string and units string, but it can also be used to display longer descriptive strings. It contains a template based on the **StringLabel** class that is used to specify the font and string format information associated with the panel.

RTStringPanelMeter constructors

```
public static newRTStringPanelMeterCoords(transform: PhysicalCoordinates): RTStringPanelMeter

public static newRTStringPanelMeterCoordsAttrib(transform: PhysicalCoordinates, attrib:
ChartAttribute): RTStringPanelMeter

public static newRTStringPanelMeter(transform: PhysicalCoordinates, datasource: RTProcessVar,
attrib: ChartAttribute): RTStringPanelMeter

public static newRTStringPanelMeterCoordsAttribStringType(transform: PhysicalCoordinates, attrib:
ChartAttribute, stringtype: number): RTStringPanelMeter

public static newRTStringPanelMeterCoordsProcessVarAttribStringType(transform:
PhysicalCoordinates, datasource: RTProcessVar, attrib: ChartAttribute, stringtype: number):
RTStringPanelMeter
```

Parameters

transform

The coordinate system for the new **RTStringPanelMeter** object.

datasource

The process variable associated with the panel meter.

attrib

The color attributes of the panel meter indicator.

stringtype

Specifies what string to display, whether it is one of the process variable strings, or a custom string. Use one of the Panel Meter string constants: **RT_CUSTOM_STRING**,

RT_TAG_STRING , RT_UNITS_STRING. Specify a custom string and use the **StringTemplate.TextString** property to set the string.

Selected Public Instance Properties

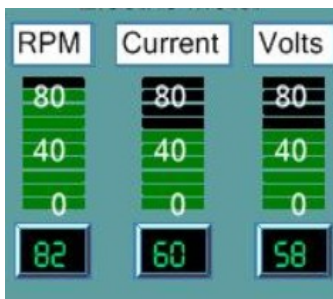
StringTemplate	Get/Set the string template defining the panel meter string format. The text properties associated with the panel meter are set using this property.

A complete listing of **RTStringPanelMeter** properties is found in the `_qcertgraphs_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example for RTStringPanelMeter used with RTMultiBarIndicator

The panel meter below, extracted from the HybridCar example, method **InitializeMotorVariablesGraph**, adds an **RTStringPanelMeter** above the motor variables multi-bar indicator. It is used to display the process variable tag name as the title for each bar in the multi-value indicator.

Note: The **RTStringPanelMeter** only needs to be added once to the **RTMultiBarIndicator**. It automatically picks up on the tag name for each **RTProcessVar** object referenced by the **RTMultiBarIndicator**.



```
let barplot: QCRTGraphTS.RTMultiBarIndicator =
QCRTGraphTS.RTMultiBarIndicator.newRTMultiBarIndicator(pTransform1, motorvars, barwidth,
barspace, barbase, attribarray, barjust, barorient);
.
.
.

let panelmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let paneltagmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 0,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
```

```

let panelmeter: QCRTGraphTS.RTNumericPanelMeter =
  QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeterCoordsAttrib(pTransform1,
    panelmeterattrib);
.
.
.

barplot.addPanelMeter(panelmeter);

let panelmeter3: QCRTGraphTS.RTStringPanelMeter =
  QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsAttribStringType(pTransform1,
    paneltagmeterattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MAX;
panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
panelmeter3.StringTemplate.TextFont = this.font14;
panelmeter3.PanelMeterNudge.Y = -4;
panelmeter3.AlarmIndicatorColorMode =
  QCRTGraphTS.ChartConstants.RT_INDICATOR_COLOR_NO_ALARM_CHANGE;
barplot.addPanelMeter(panelmeter3);

this.chartVu.addChartObject(barplot);

```

[JavaScript]

```

let barplot = QCRTGraphTS.RTMultiBarIndicator.newRTMultiBarIndicator(pTransform1, motorvars,
  barwidth, barspace, barbase, attribarray, barjust, barorient);
.
.
.
let panelmeterattrib =
  QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
  QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let paneltagmeterattrib =
  QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 0,
  QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
let panelmeter = QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeterCoordsAttrib(pTransform1,
  panelmeterattrib);
.
.
.
barplot.addPanelMeter(panelmeter);

let panelmeter3 =
  QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsAttribStringType(pTransform1,
  paneltagmeterattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MAX;
panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
panelmeter3.StringTemplate.TextFont =font14;
panelmeter3.PanelMeterNudge.Y = -4;
panelmeter3.AlarmIndicatorColorMode =
  QCRTGraphTS.ChartConstants.RT_INDICATOR_COLOR_NO_ALARM_CHANGE;
barplot.addPanelMeter(panelmeter3);

chartVu.addChartObject(barplot);

```

Time/Date Panel Meter

Class RTTimePanelMeter

ChartPlot
 RTPlot
 RTSingleValueIndicator
 RTPanelMeter
 RTTimePanelMeter

The **RTTimePanelMeter** class displays the time/date value of the time stamp of the associated **RTProcessVar** object. It contains a template based on the **TimeLabel** class that is used to specify the font and time/date format information associated with the panel meter.

RTTimePanelMeter constructors

```

public static newRTTimePanelMeterCoords(transform: PhysicalCoordinates): RTTimePanelMeter

public static newRTTimePanelMeterCoordsAttrib(transform: PhysicalCoordinates, attrib:
ChartAttribute): RTTimePanelMeter

public static newRTTimePanelMeter(transform: PhysicalCoordinates, datasource: RTProcessVar,
attrib: ChartAttribute) : RTTimePanelMeter

```

Parameters

transform

The coordinate system for the new **RTTimePanelMeter** object.

datasource

The process variable associated with the panel meter.

attrib

The color attributes of the panel meter indicator.

Selected Public Instance Properties

<u>TimeTemplate</u>	Set/Get the TimeLabel template for the panel meter time/date value. The text properties associated with the panel meter are set using this property. In addition, the time or calendar format of the time/date value is also set here.

A complete listing of **RTTimePanelMeter** properties is found in the `_qcertgraphs_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example for RTTimePanelMeter

The panel meter below, extracted from the Treadmill example, method **InitializeElapsedTimePanelMeter**, adds an **RTTimePanelMeter** as an independent panel meter at the bottom of the display. In this example the plot area of the coordinate system is set for the position of the **RTTimePanelMeter** using `pTransform1.setGraphBorderDiagonal(..)`. It is positioned inside the plot

area using the `INSIDE_INDICATOR` position constant. A string panel meter places a title above the time panel meter.



[TypeScript]

```
public InitializeElapsedTimePanelMeter() {
    if (!this.chartVu) return;
    let pTransform1: QCRTGraphTS.CartesianCoordinates =
QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
    pTransform1.setGraphBorderDiagonal(0.32, .95, 0.52, 0.98);
    let panelmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter: QCRTGraphTS.RTTimePanelMeter =
QCRTGraphTS.RTTimePanelMeter.newRTTimePanelMeter(pTransform1, this.timeOfDay, panelmeterattrib);
    // panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.INSIDE_INDICATOR;
    panelmeter.TimeTemplate.TextFont = this.font32Numeric;
    panelmeter.TimeTemplate.TimeFormat = QCRTGraphTS.ChartConstants.TIMEDATEFORMAT_24HMS;
    panelmeter.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_INDICATOR_COLOR_NO_ALARM_CHANGE;
    this.chartVu.addChartObject(panelmeter);
    let panelmetertagattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BEIGE, 0,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BEIGE);
    let panelmeter3: QCRTGraphTS.RTStringPanelMeter =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(pTransform1,
this.timeOfDay, panelmetertagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
    panelmeter3.StringTemplate.TextFont = this.font10;
    panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.ABOVE_REFERENCED_TEXT;
    panelmeter3.setPositionReference(panelmeter);
    panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
    this.chartVu.addChartObject(panelmeter3);
}
```

[JavaScript]

```
function InitializeElapsedTimePanelMeter() {
    if (!chartVu) return;
    let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
    pTransform1.setGraphBorderDiagonal(0.32, .95, 0.52, 0.98);
    let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter = QCRTGraphTS.RTTimePanelMeter.newRTTimePanelMeter(pTransform1, timeOfDay,
panelmeterattrib);
    // panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.INSIDE_INDICATOR;
    panelmeter.TimeTemplate.TextFont = font32Numeric;
    panelmeter.TimeTemplate.TimeFormat = QCRTGraphTS.ChartConstants.TIMEDATEFORMAT_24HMS;
    panelmeter.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_INDICATOR_COLOR_NO_ALARM_CHANGE;
    chartVu.addChartObject(panelmeter);
}
```

```

    let panelmetertagattrib =
    QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.BEIGE, 0,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BEIGE);
    let panelmeter3 =
    QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(pTransform1,
    timeOfDay, panelmetertagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
    panelmeter3.StringTemplate.TextFont = font10;
    panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.ABOVE_REFERENCED_TEXT;
    panelmeter3.setPositionReference(panelmeter);
    panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
    chartVu.addChartObject(panelmeter3);
}

```

Class RTElapsedTimePanelMeter

ChartPlot

RTPlot

RTSingleValueIndicator

RTPanelMeter

RTElapsedTimePanelMeter

The **RTElapsedTimePanelMeter** class displays the elapsed time value of the time stamp of the associated **RTProcessVar** object, interpreting the numeric value of the time stamp in milliseconds. It contains a template based on the **ElapsedTimeLabel** class that is used to specify the font and time/date format information associated with the panel meter.

RTTimePanelMeter constructors

```

public static newRTElapsedTimePanelMeterCoords(transform: PhysicalCoordinates):
RTElapsedTimePanelMeter

public static newRTElapsedTimePanelMeterCoordsAttrib(transform: PhysicalCoordinates, attrib:
ChartAttribute): RTElapsedTimePanelMeter

public static newRTElapsedTimePanelMeter(transform: PhysicalCoordinates, datasource:
RTProcessVar, attrib: ChartAttribute): RTElapsedTimePanelMeter

```

Parameters

transform

The coordinate system for the new **RTTimePanelMeter** object.

datasource

The process variable associated with the panel meter.

attrib

The color attributes of the panel meter indicator.

Selected Public Instance Properties

ElapsedTimeTemplate	Set/Get the ElapsedTimeLabel template for the panel meter time/date value. The text properties associated with the panel meter are set using

	this property. In addition, the time or calendar format of the time/date value is also set here.

A complete listing of **RTElapsedTimePanelMeter** properties is found in the `_qcrgraphts_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example for RTElapsedTimePanelMeter

The panel meter below, extracted from the Treadmill example, method **InitializeStopWatchTimePanelMeter**, adds an **RTElapsedTimePanelMeter** as an independent panel meter at the bottom of the display. In this example the plot area of the coordinate system is set for the position of the **RTElapsedTimePanelMeter** using `pTransform1.setGraphBorderDiagonal(..)`. It is positioned inside the plot area using the `INSIDE_INDICATOR` position constant. A string panel meter places a title above the time panel meter.



[TypeScript]

```
public InitializeStopWatchPanelMeter() {
    if (!this.chartVu) return;
    let pTransform1: QCRTGraphTS.CartesianCoordinates =
QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
    pTransform1.setGraphBorderDiagonal(0.53, .95, 0.72, 0.98);
    let panelmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter: QCRTGraphTS.RTElapsedTimePanelMeter =
QCRTGraphTS.RTElapsedTimePanelMeter.newRTElapsedTimePanelMeter(pTransform1, this.stopWatch,
panelmeterattrib);
    //panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.INSIDE_INDICATOR;
    panelmeter.TimeTemplate.TextFont = this.font32Numeric;
    panelmeter.TimeTemplate.TimeFormat = QCRTGraphTS.ChartConstants.TIMEDATEFORMAT_24HMS;
    panelmeter.TimeTemplate.DecimalPos = 0;
    panelmeter.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_INDICATOR_COLOR_NO_ALARM_CHANGE;
    this.chartVu.addChartObject(panelmeter);
    let panelmetertagattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BEIGE, 0,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BEIGE);
    let panelmeter3: QCRTGraphTS.RTStringPanelMeter =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(pTransform1,
this.stopWatch, panelmetertagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
    panelmeter3.StringTemplate.TextFont = this.font10;
    panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.ABOVE_REFERENCED_TEXT;
    panelmeter3.setPositionReference(panelmeter);
    panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
    this.chartVu.addChartObject(panelmeter3);
}
```

```
}
```

[JavaScript]

```
function InitializeStopWatchPanelMeter() {
    if (!chartVu) return;
    let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
    pTransform1.setGraphBorderDiagonal(0.53, .95, 0.72, 0.98);
    let panelmeterattrib =
    QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter = QCRTGraphTS.RTElapsedTimePanelMeter.newRTElapsedTimePanelMeter(pTransform1,
    stopWatch, panelmeterattrib);
    //panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.INSIDE_INDICATOR;
    panelmeter.TimeTemplate.TextFont = font32Numeric;
    panelmeter.TimeTemplate.TimeFormat = QCRTGraphTS.ChartConstants.TIMEDATEFORMAT_24HMS;
    panelmeter.TimeTemplate.DecimalPos = 0;
    panelmeter.AlarmIndicatorColorMode =
    QCRTGraphTS.ChartConstants.RT_INDICATOR_COLOR_NO_ALARM_CHANGE;
    chartVu.addChartObject(panelmeter);
    let panelmetertagattrib =
    QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.BEIGE, 0,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BEIGE);
    let panelmeter3 =
    QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(pTransform1,
    stopWatch, panelmetertagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
    panelmeter3.StringTemplate.TextFont = font10;
    panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.ABOVE_REFERENCED_TEXT;
    panelmeter3.setPositionReference(panelmeter);
    panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
    chartVu.addChartObject(panelmeter3);
}
```

Form Control Panel Meter

Class RTFormControlPanelMeter

ChartPlot

RTPlot

RTSingleValueIndicator

RTPanelMeter

RTFormControlPanelMeter

The **RTFormControlPanelMeter** encapsulates an **RTFormControl** object (buttons and track bars primarily, though others will also work) in a panel meter format. This allows it to use the **RTPanelMeter** positioning constants to position the form controls with respect to indicators, plot areas, text, and other panel meters.

RTFormControlPanelMeter constructors

```

public static newRTFormControlPanelMeterCoords(transform: PhysicalCoordinates):
RTFormControlPanelMeter

public static newRTFormControlPanelMeterCoordsAttrib(transform: PhysicalCoordinates, formcontrol:
MouseListener, attrib: ChartAttribute): RTFormControlPanelMeter

public static newRTFormControlPanelMeter(transform: PhysicalCoordinates, datasource:
RTProcessVar,
formcontrol: MouseListener, attrib: ChartAttribute): RTFormControlPanelMeter

public static newRTFormControlPanelMeterArray(transform: PhysicalCoordinates, formcontrolarray:
MouseListener[], attrib: ChartAttribute): RTFormControlPanelMeter

```

Parameters*transform*The coordinate system for the new **RTFormControlPanelMeter** object.*formcontrol*A reference to the **Form.Control** assigned to this panel meter.*datasource*

The process variable associated with the control.

attrib

The color attributes of the panel meter indicator.

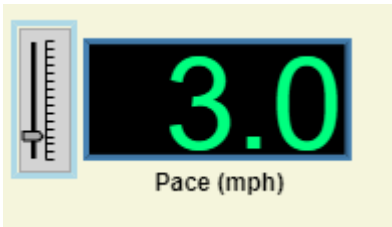
Selected Instance Properties

ControlSizeMode	Set/Get to the size mode for the Control. Use one of the Control size mode constants: RT_ORIG_CONTROL_SIZE, RT_MIN_CONTROL_SIZE, RT_INDICATORRECT_CONTROL_SIZE.

A complete listing of **RTFormControlPanelMeter** properties is found in the `_qcrgraphts_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example for RTControlTrackbar in an RTFormControlPanelMeter

The panel meter below, extracted from the Treadmill example, method **InitializeLeftPanelMeters**, adds an **RTFormControlPanelMeter** as an independent panel meter at the left of the display. In this example the plot area of the coordinate system is set for the position of the **RTFormPanelMeter** using **pTransform1.setGraphBorderDiagonal(..)**. It is positioned inside the plot area using the `CUSTOM_POSITION` position constant. The lower left corner of the form control is placed at the (0.0, 0.0) position of the plot area in `PHYS_POS` coordinates. The size of the form control is set to the size of the plot area, (width = 1.0, height = 1.0) in `PHYS_POS` coordinates.



[TypeScript]

```

let trackbarfont: QCRTGraphTS.ChartFont = this.font64Numeric;
let trackbartitlefont: QCRTGraphTS.ChartFont = this.font12Bold;
if (!this.chartVu) return;

let pTransform1: QCRTGraphTS.CartesianCoordinates =
QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
pTransform1.setGraphBorderDiagonal(0.01, .25, 0.05, 0.38);
let attrib1: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.LIGHTBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.LIGHTBLUE);
this.runnersPaceTrackbar =
QCRTGraphTS.RTControlTrackBar.newRTControlTrackBarOrientMinMaxValue(QCRTGraphTS.ChartConstants.VE
RT_DIR, 0, 15, 0.1);
this.runnersPaceTrackbar.Orientation = QCRTGraphTS.ChartConstants.VERT_DIR;
this.runnersPaceTrackbar.RTValue = 3;
this.chartVu.addMouseListener(this.runnersPaceTrackbar);

let formControlTrackBar1: QCRTGraphTS.RTFormControlPanelMeter =
QCRTGraphTS.RTFormControlPanelMeter.newRTFormControlPanelMeterCoordsAttrib(pTransform1,
this.runnersPaceTrackbar, attrib1);
formControlTrackBar1.RTDataSource = this.runnersPace;
formControlTrackBar1.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
formControlTrackBar1.setLocationXYType(0.01, 0.3, QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);

// Must be in same units as SetLocation
let panelmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let panelmeter1: QCRTGraphTS.RTNumericPanelMeter =
QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(pTransform1, this.runnersPace,
panelmeterattrib);
panelmeter1.NumericTemplate.TextFont = trackbarfont;
panelmeter1.NumericTemplate.DecimalPos = 1;
panelmeter1.PanelMeterPosition = QCRTGraphTS.ChartConstants.RIGHT_REFERENCED_TEXT;
panelmeter1.setPositionReference(formControlTrackBar1);
formControlTrackBar1.addPanelMeter(panelmeter1);

let panelmetertagattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BEIGE, 0,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BEIGE);
let panelmeter2: QCRTGraphTS.RTStringPanelMeter =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(pTransform1,
this.runnersPace, panelmetertagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
panelmeter2.setPositionReference(panelmeter1);
panelmeter2.StringTemplate.TextFont = trackbartitlefont;
panelmeter2.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
panelmeter2.TextColor = QCRTGraphTS.ChartColor.BLACK;
formControlTrackBar1.addPanelMeter(panelmeter2);
this.chartVu.addChartObject(formControlTrackBar1);

```

[JavaScript]

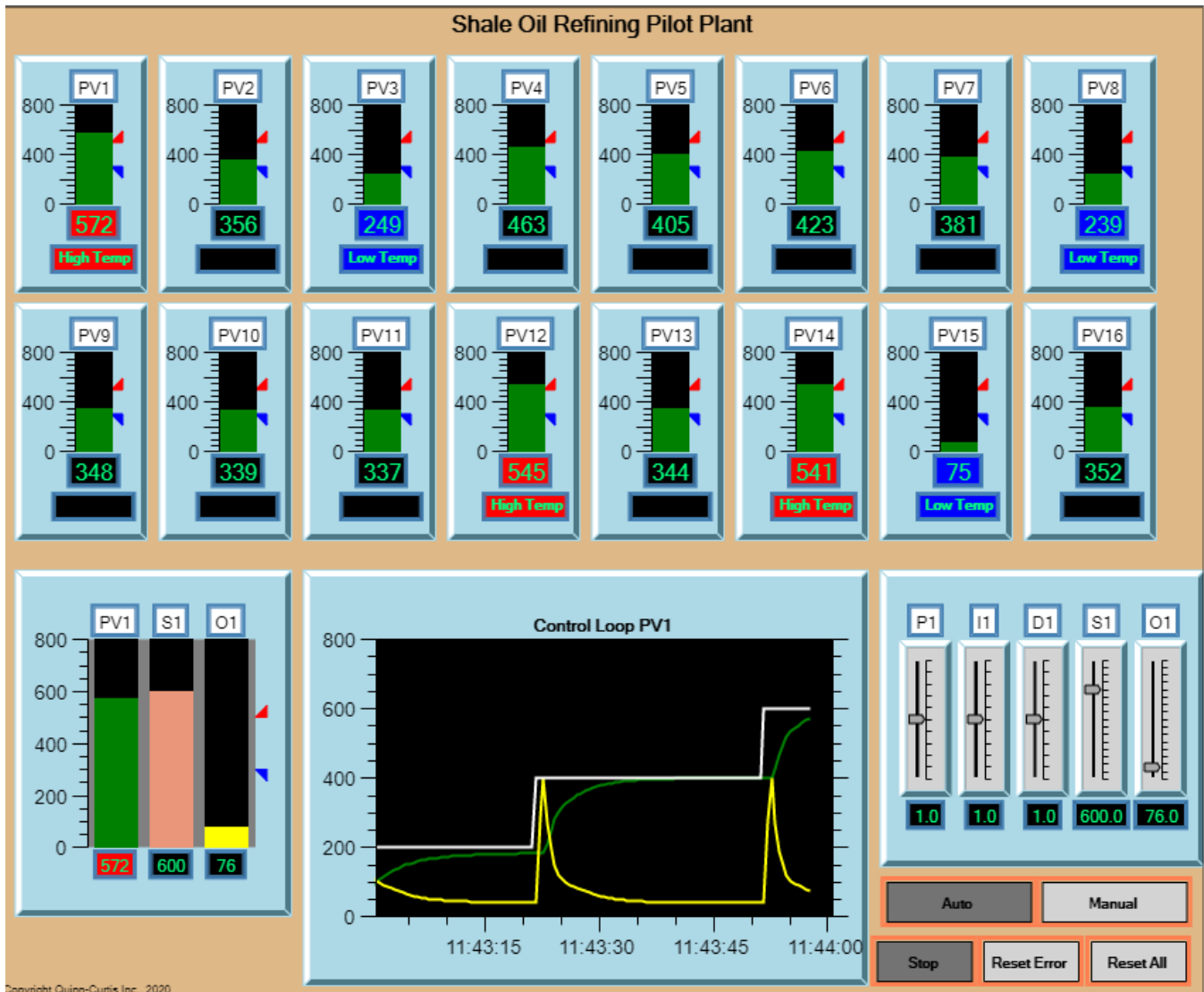
```

let trackbarfont = font64Numeric;
let trackbarTitlefont = font12Bold;
if (!chartVu) return;
let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
pTransform1.setGraphBorderDiagonal(0.01, .25, 0.05, 0.38);
let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.LIGHTBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.LIGHTBLUE);
runnersPaceTrackbar =
QCRTGraphTS.RTControlTrackBar.newRTControlTrackBarOrientMinMaxValue(QCRTGraphTS.ChartConstants.VE
RT_DIR, 0, 15, 0.1);
runnersPaceTrackbar.Orientation = QCRTGraphTS.ChartConstants.VERT_DIR;
runnersPaceTrackbar.RTValue = 3;
chartVu.addMouseListener(runnersPaceTrackbar);
// MUST USE RTValue to set double value
let formControlTrackBar1 =
QCRTGraphTS.RTFormControlPanelMeter.newRTFormControlPanelMeterCoordsAttrib(pTransform1,
runnersPaceTrackbar, attrib1);
formControlTrackBar1.RTDataSource = runnersPace;
formControlTrackBar1.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
formControlTrackBar1.setLocationXYType(0.01, 0.3, QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
//formControlTrackBar1.FormControlSize = QCRTGraphTS.ChartDimension.newChartDimension(1, 1);
// Must be in same units as SetLocation
let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let panelmeter1 = QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(pTransform1,
runnersPace, panelmeterattrib);
panelmeter1.NumericTemplate.TextFont = trackbarfont;
panelmeter1.NumericTemplate.DecimalPos = 1;
panelmeter1.PanelMeterPosition = QCRTGraphTS.ChartConstants.RIGHT_REFERENCED_TEXT;
panelmeter1.setPositionReference(formControlTrackBar1);
formControlTrackBar1.addPanelMeter(panelmeter1);
let panelmetertagattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.BEIGE, 0,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BEIGE);
let panelmeter2 =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(pTransform1,
runnersPace, panelmetertagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
panelmeter2.setPositionReference(panelmeter1);
panelmeter2.StringTemplate.TextFont = trackbarTitlefont;
panelmeter2.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
panelmeter2.TextColor = QCRTGraphTS.ChartColor.BLACK;
formControlTrackBar1.addPanelMeter(panelmeter2);
chartVu.addChartObject(formControlTrackBar1);

```

6. Single Channel Bar Indicator

RTBarIndicator



An **RTBarIndicator** is used to display the current value of an **RTProcessVar** using the height or width of a bar. One end of each bar is always fixed at the specified base value. Bars can change their size either in vertical or horizontal direction. Sub types within the **RTBarIndicator** class support segmented bars, custom segmented bars with variable width segments, and pointer bar indicators. Panel meters can

be attached to the bar indicator, where they provide text for numeric read outs, alarm warnings, descriptions and titles.

Bar Indicator

Class RTBarIndicator

ChartPlot

RTPlot

RTSingleValueIndicator

RTBarIndicator

The bar indicator is a relatively simple plot object that resides in the plot area of the specified coordinate system. It is usually combined with axes and axis labels, though this is not required. Since the bar indicator does not include axes or axis labels as option, it is up to the user to explicitly create axis and axis label objects for the bar indicator graph. The axis and axis labels routines make this easy to do.

RTBarIndicator constructors

```
public static newRTBarIndicatorCoords(transform: PhysicalCoordinates): RTBarIndicator

public static newRTBarIndicatorCoordsProcessVarAttrib(transform: PhysicalCoordinates, datasource:
RTProcessVar, barwidth: number, barbase: number, attrib: ChartAttribute, barjust: number,
barorient: number): RTBarIndicator

public static newRTBarIndicator(transform: PhysicalCoordinates, datasource: RTProcessVar,
barwidth: number, barbase: number, attrib: ChartAttribute, barjust: number, barorient: number):
RTBarIndicator
```

Parameters

transform

The coordinate system for the new **RTBarIndicator** object.

datasource

The process variable associated with the bar indicator.

barwidth

The width of the bar in physical units .

barbase

The base of the bar in physical units.

attrib

The color attributes of the bar indicator.

barjust

The justification of bars. Use one of the bar justification constants: JUSTIFY_MIN, JUSTIFY_CENTER, JUSTIFY_MAX..

barorient

The orientation of the bar indicator: HORIZ_DIR or VERT_DIR.

Selected Public Instance Properties[AlarmIndicatorColorMode](#)

(inherited from

RTSingleValueIndicator)

Get/Set whether the color of the indicator objects changes on an alarm. Use one of the constants: RT_INDICATOR_COLOR_NO_ALARM_CHANGE, RT_INDICATOR_COLOR_CHANGE_ON_ALARM.

[BarDatapointLabelPosition](#)(inherited from **ChartPlot**)

Bar plots that support the display of data point values have the option of displaying the data point's numeric values above the bar, below the bar, or centered in the bar. Use one of the data point label position constants: INSIDE_BAR, OUTSIDE_BAR, or CENTERED_BAR.

[BarJust](#) (inherited from**ChartPlot**)

Set/Get the justification of bars in bar graph plot objects. Use one of the bar justification constants: JUSTIFY_MIN, JUSTIFY_CENTER, or JUSTIFY_MAX.

[BarOffset](#)

Set/Get the bar offset from its fixed x or y value in physical units.

[BarOrient](#) (inherited from**ChartPlot**)

Set/Get the orientation (HORIZ_DIR or VERT_DIR) for bar plots.

[BarSpacing](#) (inherited from **RTPlot**)

Set/Get the spacing between adjacent items in multi-channel plots.

[BarWidth](#) (inherited from**ChartPlot**)

Set/Get the width of bars, in physical coordinates, for bar plots.

[ChartObjAttributes](#) (inherited from **GraphObj**)

Sets the attributes for a chart object using a **ChartAttribute** object.

[ChartObjClipping](#) (inherited from**GraphObj**)

Sets the object clipping mode. Use one of the object clipping constants: NO_CLIPPING, GRAPH_AREA_CLIPPING, PLOT_AREA_CLIPPING, or INHERIT_CLIPPING.

[ChartObjComponent](#) (inherited from **GraphObj**)

Sets the reference to the **ChartView** component that the chart object is placed in

[ChartObjEnable](#) (inherited from**GraphObj**)

Enables/Disables the chart object. A chart object is drawn only if it is enabled. A chart object is enabled by default.

[ChartObjScale](#) (inherited from **GraphObj**)

Sets the reference to the PhysicalCoordinates object that the chart object is placed in

[CurrentProcessValue](#) (inherited from **RTSingleValueIndicator**)

Get the current process value of the primary channel.

[FillBaseValue](#) (inherited from **ChartPlot**)

Set/Get the base value, in physical coordinates, of solid (bars and filled areas) plot objects.

[FillColor](#) (inherited from**GraphObj**)

Sets the fill color for the chart object.

[IndicatorBackground](#)
[IndicatorBackgroundEnable](#)

[IndicatorSubType](#)

[LabelTemplateDecimalPos](#)
 (inherited from **ChartPlot**)
[LabelTemplateNumericFormat](#)
 (inherited from **ChartPlot**)

[LineColor](#) (inherited from
GraphObj)

[LineStyle](#) (inherited from
GraphObj)

[LineWidth](#) (inherited from
GraphObj)

[NumChannels](#) (inherited from
RTPlot)

[PlotLabelTemplate](#) (inherited
 from **ChartPlot**)

[PointerSymbolNum](#)

[PrimaryChannel](#) (inherited from
RTPlot)

[RTDataSource](#) (inherited from
RTSingleValueIndicator)

[SegmentCornerRadius](#)

[SegmentSpacing](#)

Get/Set the background attribute of the bar indicator.
 Set to true to enable the display of the bar indicator background.

Get/Set the bar indicator sub type:

RT_BAR_SOLID_SUBTYPE,
 RT_BAR_SEGMENTED_SUBTYPE,
 RT_BAR_SINGLE_SEGMENT_SUBTYPE,
 RT_POINTER_SUBTYPE.

Set/Get number of digits to the right of the decimal point in the **PlotLabelTemplate** property.

Set/Get the numeric format of the
PlotLabelTemplate property.

Sets the line color for the chart object.

Sets the line style for the chart object.

Sets the line width for the chart object.

Get the number of channels in the indicator.

Set/Get the plot objects data point template. If the plot supports it, this **NumericLabel** object is used as a template to size, color and format the data point numeric values.

Set/Get the symbol used for the pointer symbol indicator subtype, RT_POINTER_SUBTYPE. Use one of the constants: RT_NO_SYMBOL, RT_LEFT_LOW_ALARM_SYMBOL, RT_LEFT_SETPOINT_SYMBOL.

RT_LEFT_HIGH_ALARM_SYMBOL,
 RT_RIGHT_LOW_ALARM_SYMBOL,
 RT_RIGHT_SETPOINT_SYMBOL,
 RT_RIGHT_HIGH_ALARM_SYMBOL,
 RT_TOP_LOW_ALARM_SYMBOL,
 RT_TOP_SETPOINT_SYMBOL,
 RT_TOP_HIGH_ALARM_SYMBOL,
 RT_BOTTOM_LOW_ALARM_SYMBOL,
 RT_BOTTOM_SETPOINT_SYMBOL,
 RT_BOTTOM_HIGH_ALARM_SYMBOL.

Set/Get the primary channel of the indicator.

Get/Set the array list holding the **RTProcessVar** variables for the indicator.

Get/Set the corner radius used to draw the segment rounded rectangles.

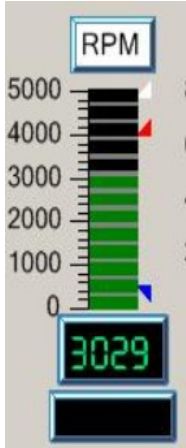
Get/Set the segments spacing for the

	RT_BAR_SEGMENTED_SUBTYPE and RT_BAR_SINGLE_SEGMENT_SUBTYPE bar indicator sub types.
SegmentValueRoundMode	Set/Get the segment value round mode. Specifies that the current process value is rounded up in calculating how many segments to display in RT_BAR_SEGMENTED_SUBTYPE and RT_BAR_SINGLE_SEGMENT_SUBTYPE modes. Use one of the constants: RT_FLOOR_VALUE, RT_CEILING_VALUE.
SegmentWidth	Get/Set the thickness of segments for the RT_BAR_SEGMENTED_SUBTYPE and RT_BAR_SINGLE_SEGMENT_SUBTYPE bar indicator sub types.
ShowDatapointValue (inherited from ChartPlot)	If the plot supports it, this method will turn on/off the display of data values next to the associated data point.
StepMode (inherited from ChartPlot)	Set/Get the plot objects step mode. Use one of the line plot step constants: NO_STEP, STEP_START, STEP_END, or STEP_NO_RISE_LINE.
ZOrder (inherited from GraphObj)	Sets the z-order of the object in the chart. Every object has a z-order value. Each z-order value does NOT have to be unique. If z-order sorting is turned on in the ChartView object, objects are sorted by z-order before they are drawn.

A complete listing of **RTBarIndicator** properties is found in the `_qcertgraphs_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example for an RTBarIndicator Segmented Bar Indicator

The bar indicator example below, extracted from the Dynamometer example, method **InitializeEngine1RPMIndicator**, creates the segmented bar RPM indicator in the upper left corner of the graph. It demonstrates how the plot area is defined for the bar indicator, how to create axes and axis labels. An **RTAlarmIndicator** is also created to display the alarm limit symbols to the right of the bar indicator. The image below also includes an **RTStringPanelMeter** for the “RPM” tag, an **RTNumericPanelMeter** for the numeric readout below the bar indicator, and an **RTAlarmPanelMeter** below that. See the Dynamometer example program for the complete program listing that creates all of these objects.



[TypeScript]

```

let pTransform1: QCRTGraphTS.CartesianCoordinates =
  QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 5000);
pTransform1.setGraphBorderDiagonal(0.05, 0.175, 0.08, 0.35);
let background: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(pTransform1,
  QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
this.chartVu.addChartObject(background);
let attrib1: QCRTGraphTS.ChartAttribute =
  QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.GREEN,
  1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN);
let barbase: number = 0;
let barwidth: number = 1;
let barjust: number = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
let barorient: number = QCRTGraphTS.ChartConstants.VERT_DIR;
let baraxis: QCRTGraphTS.LinearAxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
  QCRTGraphTS.ChartConstants.Y_AXIS);
this.chartVu.addChartObject(baraxis);
let barAxisLab: QCRTGraphTS.NumericAxisLabels =
  QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(baraxis);
barAxisLab.AxisLabelsFormat = QCRTGraphTS.ChartConstants.BUSINESSFORMAT;
this.chartVu.addChartObject(barAxisLab);
let barplot: QCRTGraphTS.RTBarIndicator =
  QCRTGraphTS.RTBarIndicator.newRTBarIndicator(pTransform1, this.EngineRPM1, barwidth, barbase,
  attrib1, barjust, barorient);
barplot.SegmentSpacing = 400;
barplot.SegmentWidth = 250;
barplot.IndicatorBackground =
  QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.BLACK,
  1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
barplot.SegmentValueRoundMode = QCRTGraphTS.ChartConstants.RT_CEILING_VALUE;
barplot.SegmentCornerRadius = 0;
barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SEGMENTED_SUBTYPE;
let baralarms: QCRTGraphTS.RTAlarmIndicator =
  QCRTGraphTS.RTAlarmIndicator.newRTAlarmIndicator(baraxis, barplot);
this.chartVu.addChartObject(baralarms);
let panelmeterattrib: QCRTGraphTS.ChartAttribute =
  QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE,
  3, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let paneltagmeterattrib: QCRTGraphTS.ChartAttribute =
  QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE,
  3, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
let panelmeter: QCRTGraphTS.RTNumericPanelMeter =
  QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeterCoordsAttrib(pTransform1,
  panelmeterattrib);
panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MIN;
panelmeter.PanelMeterNudge.Y = 6;
panelmeter.TextColor = this.brightGREEN;

```



```

panelmeter.NumericTemplate.TextFont = this.font18Numeric;
panelmeter.NumericTemplate.DecimalPos = 0;
panelmeter.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
barplot.addPanelMeter(panelmeter);
let panelmeter2: QCRTGraphTS.RTAlarmPanelMeter =
QCRTGraphTS.RTAlarmPanelMeter.newRTAlarmPanelMeterCoordsAttrib(pTransform1, panelmeterattrib);
panelmeter2.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
panelmeter2.TextColor = this.brightGREEN;
panelmeter2.AlarmTemplate.TextFont = this.font10Bold;
panelmeter2.setPositionReference(panelmeter);
panelmeter2.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
barplot.addPanelMeter(panelmeter2);
let panelmeter3: QCRTGraphTS.RTStringPanelMeter
=QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsAttribStringType(pTransform1,
paneltagmeterattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MAX;
panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
panelmeter3.StringTemplate.TextFont = this.font12;
panelmeter3.PanelMeterNudge =QCRTGraphTS.ChartPoint2D.newChartPoint2D(0, -6);
panelmeter3.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_INDICATOR_COLOR_NO_ALARM_CHANGE;
barplot.addPanelMeter(panelmeter3);
this.chartVu.addChartObject(barplot);

```

[JavaScript]

```

let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 5000);
pTransform1.setGraphBorderDiagonal(0.05, 0.175, 0.08, 0.35);
let background = QCRTGraphTS.Background.newBackground(pTransform1,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
chartVu.addChartObject(background);
let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.GREEN, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN);
let barbase = 0;
let barwidth = 1;
let barjust = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
let barorient = QCRTGraphTS.ChartConstants.VERT_DIR;
let baraxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
QCRTGraphTS.ChartConstants.Y_AXIS);
chartVu.addChartObject(baraxis);
let barAxisLab = QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(baraxis);
barAxisLab.AxisLabelsFormat = QCRTGraphTS.ChartConstants.BUSINESSFORMAT;
chartVu.addChartObject(barAxisLab);
let barplot = QCRTGraphTS.RTBarIndicator.newRTBarIndicator(pTransform1, EngineRPM1, barwidth,
barbase, attrib1, barjust, barorient);
barplot.SegmentSpacing = 400;
barplot.SegmentWidth = 250;
barplot.IndicatorBackground =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
barplot.SegmentValueRoundMode = QCRTGraphTS.ChartConstants.RT_CEILING_VALUE;
barplot.SegmentCornerRadius = 0;
barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SEGMENTED_SUBTYPE;
let baralarms = QCRTGraphTS.RTAlarmIndicator.newRTAlarmIndicator(baraxis, barplot);
chartVu.addChartObject(baralarms);
let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let paneltagmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
let panelmeter = QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeterCoordsAttrib(pTransform1,
panelmeterattrib);
panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MIN;
panelmeter.PanelMeterNudge.Y = 6;
panelmeter.TextColor = brightGREEN;

```

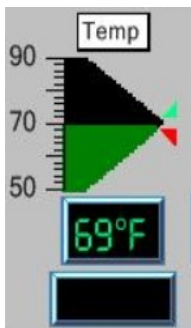
```

panelmeter.NumericTemplate.TextFont = font18Numeric;
panelmeter.NumericTemplate.DecimalPos = 0;
panelmeter.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
barplot.addPanelMeter(panelmeter);
let panelmeter2 = QCRTGraphTS.RTAlarmPanelMeter.newRTAlarmPanelMeterCoordsAttrib(pTransform1,
panelmeterattrib);
panelmeter2.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
panelmeter2.TextColor = brightGREEN;
panelmeter2.AlarmTemplate.TextFont = font10Bold;
panelmeter2.setPositionReference(panelmeter);
panelmeter2.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
barplot.addPanelMeter(panelmeter2);
let panelmeter3 =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsAttribStringType(pTransform1,
paneltagmeterattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MAX;
panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
panelmeter3.StringTemplate.TextFont = font12;
panelmeter3.PanelMeterNudge = QCRTGraphTS.ChartPoint2D.newChartPoint2D(0, -6);
panelmeter3.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_INDICATOR_COLOR_NO_ALARM_CHANGE;
barplot.addPanelMeter(panelmeter3);
chartVu.addChartObject(barplot);

```

Example for an RTBarIndicator Custom Segmented Bar Indicator

The custom bar indicator example below, extracted from the HomeAutomation example, method **InitializeCustomBarIndicator**, uses a segmented bar indicator to display the temperature. It uses a special feature that allows the width of the each bar segment to be calculated as a function of the height. This is done by subclassing the **RTBarIndicator** class and overriding the **getCustomBarWidth** and **getCustomBarOffse** methods. In the **CustomRTBarIndicator** example below, the width of the bar is calculated using a function based on the deviation of the current temperature from the *temperatureSetpoint* value. Calculating the bar width as a function of the bar height only works with the segmented bar subtypes. If you want a solid bar, make the **RTBarIndicator.SegmentWidth** and **RTBarIndicator.SegmetSpacing** values small, and the same, as in the example below.



[TypeScript]

```

class CustomRTBarIndicatorThermostat extends QCRTGraphTS.RTBarIndicator {
    temperatureSetpoint: number = 70;

```

```

    public constructor() {
        super();
    }

    public static newCustomRTBarIndicatorThermostat(transform: QCRTGraphTS.PhysicalCoordinates,
datasource: QCRTGraphTS.RTProcessVar,
barwidth: number, barbase: number, attrib: QCRTGraphTS.ChartAttribute, barjust: number,
barorient: number): CustomRTBarIndicatorThermostat {
    let result: CustomRTBarIndicatorThermostat = new CustomRTBarIndicatorThermostat();
    result.setChartObjScale(transform);
    result.setBarOrient(barorient);
    result.initRTBarIndicator(datasource, barwidth, barbase, attrib, barjust);

    return result;
}

    public getCustomBarOffset(v: number): number {
    let offset: number = 0;
    return offset;
}

    public getCustomBarWidth(v: number): number {
    // Calculate width as fraction of initial bar width
    let width: number = 1;
    // Bar widest at setpoint, narrowest at endpoints
    // Clamp width to 0.05 to 1.0 range
    width = Math.max(0.05, (this.BarWidth - Math.abs((0.04 * (v - this.temperatureSetpoint)))));
    width = Math.min(1, width);
    return width;
}

    /// <summary>
    /// Set/Get local setpoint
    /// </summary>
    public get TemperatureSetpoint(): number {
    return this.temperatureSetpoint;
    }
    public set TemperatureSetpoint(value: number) {
    this.temperatureSetpoint = value;
    }
}

this.barplot = CustomRTBarIndicatorThermostat.newCustomRTBarIndicatorThermostat(pTransform1,
this.currentTemperature1, barwidth, barbase, attrib1, barjust, barorient);
this.barplot.IndicatorBackground =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
this.barplot.SegmentSpacing = 1;
this.barplot.SegmentWidth = 1;
this.barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SEGMENTED_SUBTYPE;
this.barplot.SegmentValueRoundMode = QCRTGraphTS.ChartConstants.RT_CEILING_VALUE;
.
.
.

this.chartVu.addChartObject(this.barplot);
.
.
.
}

    public InitializeCustomBarIndicator() {

    let x1: number = 0.2;
    let y1: number = 0.25;
    let x2: number = 0.5;

```

```

let y2: number = 0.6;
if (!this.chartVu) return;
let pTransform1: QCRTGraphTS.CartesianCoordinates =
  QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0,
    this.currentTemperature1.DefaultMinimumDisplayValue, 1,
    this.currentTemperature1.DefaultMaximumDisplayValue);
pTransform1.setGraphBorderDiagonal(x1, y1, x2, y2);
let background: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(pTransform1,
  QCRTGraphTS.ChartConstants.GRAPH_BACKGROUND, QCRTGraphTS.ChartColor.WHITE);
this.chartVu.addChartObject(background);
let attrib1: QCRTGraphTS.ChartAttribute =
  QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 3,
  QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN);
let barbase: number = this.currentTemperature1.DefaultMinimumDisplayValue;
let barwidth: number = 1;
let barjust: number = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
let barorient: number = QCRTGraphTS.ChartConstants.VERT_DIR;
let baraxis: QCRTGraphTS.LinearAxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
  QCRTGraphTS.ChartConstants.Y_AXIS);
this.chartVu.addChartObject(baraxis);
let barAxisLab: QCRTGraphTS.NumericAxisLabels =
  QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(baraxis);
this.chartVu.addChartObject(barAxisLab);
this.barplot = CustomRTBarIndicatorThermostat.newCustomRTBarIndicatorThermostat(pTransform1,
  this.currentTemperature1, barwidth, barbase, attrib1, barjust, barorient);
this.barplot.IndicatorBackground =
  QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
  QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
this.barplot.SegmentSpacing = 1;
this.barplot.SegmentWidth = 1;
this.barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SEGMENTED_SUBTYPE;
this.barplot.SegmentValueRoundMode = QCRTGraphTS.ChartConstants.RT_CEILING_VALUE;
.
.
.

this.chartVu.addChartObject(this.barplot);
.
.
.
}

```

[JavaScript]

```

class CustomRTBarIndicatorThermostat extends QCRTGraphTS.RTBarIndicator {
  constructor() {
    super();
    this.temperatureSetpoint = 70;
  }
  static newCustomRTBarIndicatorThermostat(transform, datasource, barwidth, barbase, attrib,
    barjust, barorient) {
    let result = new CustomRTBarIndicatorThermostat();
    result.setChartObjScale(transform);
    result.setBarOrient(barorient);
    result.initRTBarIndicator(datasource, barwidth, barbase, attrib, barjust);
    return result;
  }
  getCustomBarOffset(v) {
    let offset = 0;
    return offset;
  }
  getCustomBarWidth(v) {
    // Calculate width as fraction of initial bar width
    let width = 1;
    // Bar widest at setpoint, narrowest at endpoints
    // Clamp width to 0.05 to 1.0 range

```

```

        width = Math.max(0.05, (this.BarWidth - Math.abs((0.04 * (v -
this.temperatureSetpoint)))));
        width = Math.min(1, width);
        return width;
    }
    /// <summary>
    /// Set/Get local setpoint
    /// </summary>
    get TemperatureSetpoint() {
        return this.temperatureSetpoint;
    }
    set TemperatureSetpoint(value) {
        this.temperatureSetpoint = value;
    }
}

InitializeCustomBarIndicator() {
    let x1 = 0.2;
    let y1 = 0.25;
    let x2 = 0.5;
    let y2 = 0.6;
    if (!this.chartVu)
        return;
    let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0,
this.currentTemperature1.DefaultMinimumDisplayValue, 1,
this.currentTemperature1.DefaultMaximumDisplayValue);
    pTransform1.setGraphBorderDiagonal(x1, y1, x2, y2);
    let background = QCRTGraphTS.Background.newBackground(pTransform1,
QCRTGraphTS.ChartConstants.GRAPH_BACKGROUND, QCRTGraphTS.ChartColor.WHITE);
    this.chartVu.addChartObject(background);
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN);
    let barbase = this.currentTemperature1.DefaultMinimumDisplayValue;
    let barwidth = 1;
    let barjust = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
    let barorient = QCRTGraphTS.ChartConstants.VERT_DIR;
    let baraxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
QCRTGraphTS.ChartConstants.Y_AXIS);
    this.chartVu.addChartObject(baraxis);
    let barAxisLab = QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(baraxis);
    this.chartVu.addChartObject(barAxisLab);
    this.barplot = CustomRTBarIndicatorThermostat.newCustomRTBarIndicatorThermostat(pTransform1,
this.currentTemperature1, barwidth, barbase, attrib1, barjust, barorient);
    this.barplot.IndicatorBackground =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    this.barplot.SegmentSpacing = 1;
    this.barplot.SegmentWidth = 1;
    this.barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SEGMENTED_SUBTYPE;
    this.barplot.SegmentValueRoundMode = QCRTGraphTS.ChartConstants.RT_CEILING_VALUE;
    .
    .
    .

    this.chartVu.addChartObject(this.barplot);
    .
    .
    .
}

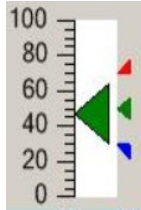
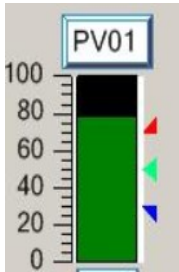
'InitializeCustomBarIndicator

```

Example for an RTBarIndicator Solid Bar Indicator and Pointer Indicator

Setting up the solid bar and pointer indicators are pretty much identical to the segmented bar indicator. The examples below are extracted from the Bargraphs example program, file SolidBars, method

InitializeBar1. The default value for the **IndicatorSubType** property is **RT_BAR_SOLID_SUBTYPE** so that does not even need to be set.



[TypeScript]

```
private InitializeBar1() {

    if (!this.chartVu) return;
    let pTransform1: QCRTGraphTS.CartesianCoordinates =
        QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(-0.1, 0, 1.1, 100);
    pTransform1.setGraphBorderDiagonal(0.1, .1, 0.15, 0.3);
    let background: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(pTransform1,
        QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.WHITE);
    this.chartVu.addChartObject(background);
    let attrib1: QCRTGraphTS.ChartAttribute =
        QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
            QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN);
    let barbase: number = 0;
    let barwidth: number = 1;
    let barjust: number = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
    let barorient: number = QCRTGraphTS.ChartConstants.VERT_DIR;
    let baraxis: QCRTGraphTS.LinearAxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
        QCRTGraphTS.ChartConstants.Y_AXIS);
    this.chartVu.addChartObject(baraxis);
    let barAxisLab: QCRTGraphTS.NumericAxisLabels =
        QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(baraxis);
    this.chartVu.addChartObject(barAxisLab);
    let barplot: QCRTGraphTS.RTBarIndicator =
        QCRTGraphTS.RTBarIndicator.newRTBarIndicator(pTransform1, this.processVar1, barwidth, barbase,
            attrib1, barjust, barorient);
    barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SOLID_SUBTYPE;
    .
    .
    .

    this.chartVu.addChartObject(barplot);
}
```

Use

```
barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_POINTER_SUBTYPE;
```

for the pointer version of the RTBarIndicator.

[JavaScript]

```

InitializeBar1() {

if (!this.chartVu) return;
let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(-0.1, 0, 1.1, 100);
pTransform1.setGraphBorderDiagonal(0.1, .1, 0.15, 0.3);
let background = QCRTGraphTS.Background.newBackground(pTransform1,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.WHITE);
this.chartVu.addChartObject(background);
let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN);
let barbase = 0;
let barwidth = 1;
let barjust = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
let barorient = QCRTGraphTS.ChartConstants.VERT_DIR;
let baraxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
QCRTGraphTS.ChartConstants.Y_AXIS);
this.chartVu.addChartObject(baraxis);
let barAxisLab = QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(baraxis);
this.chartVu.addChartObject(barAxisLab);
let barplot = QCRTGraphTS.RTBarIndicator.newRTBarIndicator(pTransform1, this.processVar1,
barwidth, barbase, attrib1, barjust, barorient);
barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SOLID_SUBTYPE;
.
.
.

this.chartVu.addChartObject(barplot);

}

```

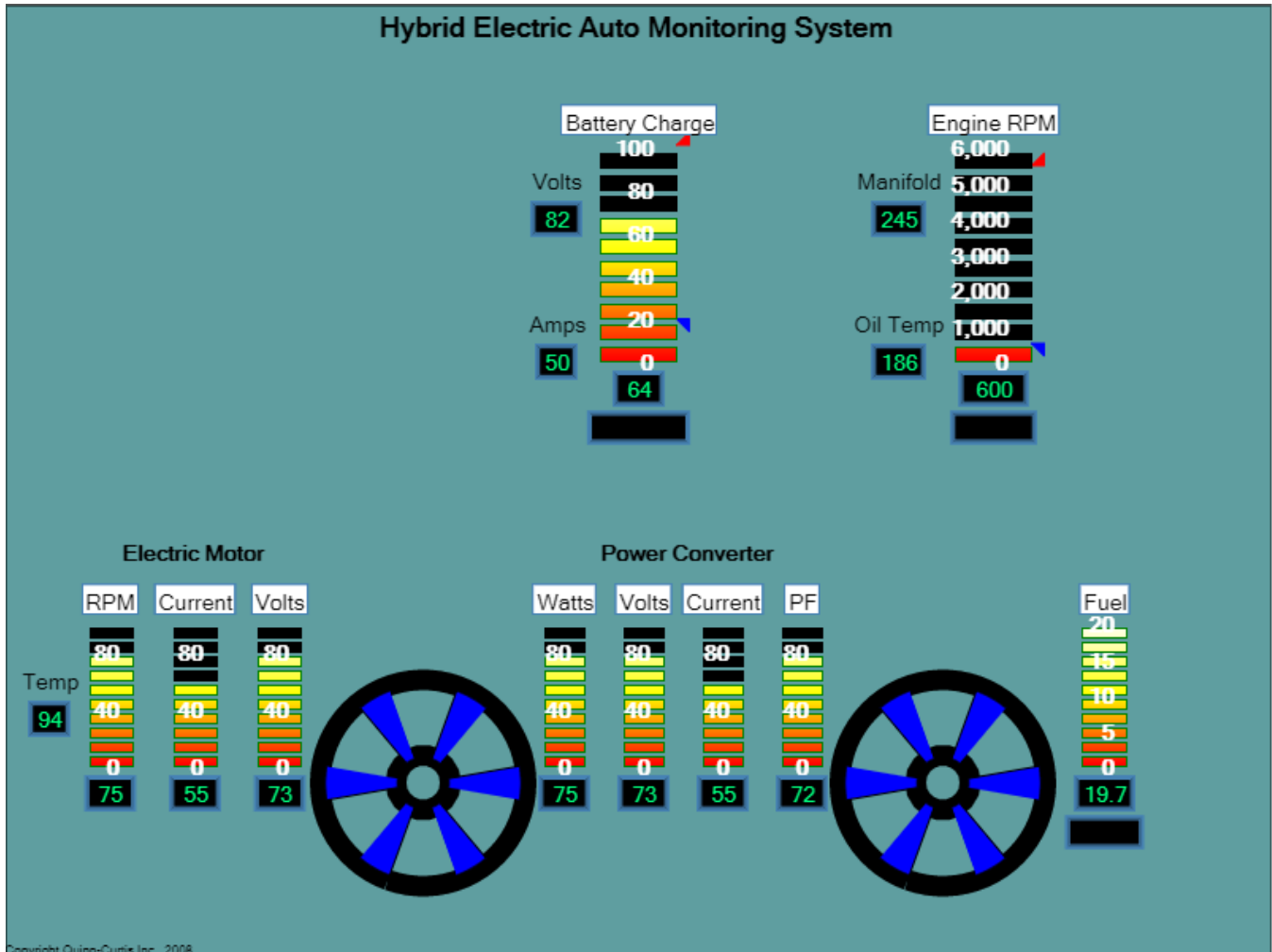
Use

```
barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_POINTER_SUBTYPE;
```

for the pointer version of the RTBarIndicator.

7. Multiple Channel Bar Indicator

RTMultiBarIndicator



An **RTMultiBarIndicator** is used to display the current value of a collection of **RTProcessVar** objects using a group of bars changing size. The bars are always fixed at the specified base value. Bars can change their size either in vertical or horizontal direction. Sub types within the **RTMultiBarIndicator** class support segmented bars, custom segmented bars with variable width segments, and pointer bar indicators.

Multiple Channel Bar Indicator

Class RTMultiBarIndicator

ChartPlot

RTPlot

RTMultiValueIndicator

RTMultiBarIndicator

The multi-bar indicator displays a collection of **RTProcessVar** objects that are related, or at the very least comparable, when bar graphed against one another using the same physical coordinate system.. It is usually combined with axes and axis labels, though this is not required. Since the bar indicator does not include axes or axis labels as option, it is up to the user to explicitly create axis and axis label objects for the bar indicator graph. The axis and axis labels routines make this easy to do.

When an **RTPanelMeter** object is added to an **RTMultiBarIndicator**, it is used as a template to create a multiple panel meters, one for each bar of the multi-bar indicator. The panel meter for each bar will reference the process variable information associated with that bar, stored in the **RTProcessVar** objects attached to the multi-bar indicator.

RTMultiBarIndicator constructors

```
public static newRTMultiBarIndicatorCoords(transform: PhysicalCoordinates): RTMultiBarIndicator

public static newRTMultiBarIndicatorCoordsProcessVar(transform: PhysicalCoordinates, datasource:
RTProcessVar[], barwidth: number, barspacing: number, barbase: number, attribs: ChartAttribute[],
barjust: number, barorient: number): RTMultiBarIndicator

public static newRTMultiBarIndicator(transform: PhysicalCoordinates, datasource: RTProcessVar[],
barwidth: number, barspacing: number, barbase: number, attribs: ChartAttribute[], barjust:
number, barorient: number): RTMultiBarIndicator

public static newRTMultiBarIndicatorFull(transform: PhysicalCoordinates, datasource:
RTMultiProcessVar, barwidth: number, barspacing: number, barbase: number, attribs:
ChartAttribute[], barjust: number, barorient: number): RTMultiBarIndicator
```

Parameters

transform

The coordinate system for the new **RTMultiBarIndicator** object.

datasource

An array of the process variables associated with the bar indicator.

barwidth

The width of each bar in physical units.

barspacing

The space between adjacent bars in physical units.

barbase

The base of the bar in physical units.

attribs

An array of the color attributes of the bar indicator.

barjust

The justification of bars. Use one of the bar justification constants: JUSTIFY_MIN, JUSTIFY_CENTER, JUSTIFY_MAX..

barorient

The orientation of the bar indicator: HORIZ_DIR or VERT_DIR.

Selected Public Instance Properties

<u>AlarmIndicatorColorMode</u> (inherited from RTSingleValueIndicator)	Get/Set whether the color of the indicator objects changes on an alarm. Use one of the constants: RT_INDICATOR_COLOR_NO_ALARM_CHANGE, RT_INDICATOR_COLOR_CHANGE_ON_ALARM..
<u>BarDatapointLabelPosition</u> (inherited from ChartPlot)	Bar plots that support the display of data point values have the option of displaying the data point's numeric values above the bar, below the bar, or centered in the bar. Use one of the data point label position constants: INSIDE_BAR, OUTSIDE_BAR, or CENTERED_BAR.
<u>BarJust</u> (inherited from ChartPlot)	Set/Get the justification of bars in bar graph plot objects. Use one of the bar justification constants: JUSTIFY_MIN, JUSTIFY_CENTER, or JUSTIFY_MAX.
<u>BarOffset</u>	Set/Get the bar offset from its fixed x or y value in physical units.
<u>BarOrient</u> (inherited from ChartPlot)	Set/Get the orientation (HORIZ_DIR or VERT_DIR) for bar plots.
<u>BarSpacing</u> (inherited from RTPlot)	Set/Get the spacing between adjacent items in multi-channel plots.
<u>BarWidth</u> (inherited from ChartPlot)	Set/Get the width of bars, in physical coordinates, for bar plots.
<u>ChartObjAttributes</u> (inherited from GraphObj)	Sets the attributes for a chart object using a ChartAttribute object.
<u>ChartObjClipping</u> (inherited from GraphObj)	Sets the object clipping mode. Use one of the object clipping constants: NO_CLIPPING, GRAPH_AREA_CLIPPING, PLOT_AREA_CLIPPING, or INHERIT_CLIPPING.
<u>ChartObjComponent</u> (inherited from GraphObj)	Sets the reference to the ChartView component that the chart object is placed in
<u>ChartObjEnable</u> (inherited from GraphObj)	Enables/Disables the chart object. A chart object is drawn only if it is enabled. A chart object is enabled by default.
<u>ChartObjScale</u> (inherited from GraphObj)	Sets the reference to the PhysicalCoordinates object that the chart object is placed in
<u>CurrentProcessValue</u> (inherited from RTSingleValueIndicator)	Get the current process value of the primary channel.
<u>FillBaseValue</u> (inherited from ChartPlot)	Set/Get the base value, in physical coordinates, of solid (bars and filled areas) plot objects.
<u>FillColor</u> (inherited from GraphObj)	Sets the fill color for the chart object.
<u>IndicatorBackground</u>	Get/Set the background attribute of the bar indicator.
<u>IndicatorBackgroundEnable</u>	Set to true to enable the display of the bar indicator background.

[IndicatorSubType](#)

Get/Set the bar indicator sub type:

RT_BAR_SOLID_SUBTYPE,
RT_BAR_SEGMENTED_SUBTYPE,
RT_BAR_SINGLE_SEGMENT_SUBTYPE,
RT_POINTER_SUBTYPE.

[LabelTemplateDecimalPos](#)

(inherited from **ChartPlot**)

Set/Get number of digits to the right of the decimal point in the **PlotLabelTemplate** property.

[LabelTemplateNumericFormat](#)

(inherited from **ChartPlot**)

Set/Get the numeric format of the **PlotLabelTemplate** property.

[LineColor](#)

(inherited from **GraphObj**)

Sets the line color for the chart object.

[LineStyle](#)

(inherited from **GraphObj**)

Sets the line style for the chart object.

[LineWidth](#)

(inherited from **GraphObj**)

Sets the line width for the chart object.

[NumChannels](#)

(inherited from **RTPlot**)

Get the number of channels in the indicator.

[PlotLabelTemplate](#)

(inherited from **ChartPlot**)

Set/Get the plot objects data point template. If the plot supports it, this **PlotLabelTemplate** object is used as a template to size, color and format the data point numeric values.

[PointerSymbolNum](#)

Set/Get the symbol used for the pointer symbol indicator subtype, RT_POINTER_SUBTYPE. Use one of the constants: RT_NO_SYMBOL, RT_LEFT_LOW_ALARM_SYMBOL, RT_LEFT_SETPOINT_SYMBOL,

RT_LEFT_HIGH_ALARM_SYMBOL ,
RT_RIGHT_LOW_ALARM_SYMBOL,
RT_RIGHT_SETPOINT_SYMBOL,
RT_RIGHT_HIGH_ALARM_SYMBOL,
RT_TOP_LOW_ALARM_SYMBOL ,
RT_TOP_SETPOINT_SYMBOL,
RT_TOP_HIGH_ALARM_SYMBOL,
RT_BOTTOM_LOW_ALARM_SYMBOL,
RT_BOTTOM_SETPOINT_SYMBOL,
RT_BOTTOM_HIGH_ALARM_SYMBOL.

Set/Get the primary channel of the indicator.

[PrimaryChannel](#)

(inherited from **RTPlot**)

[RTDataSource](#)

(inherited from **RTSingleValueIndicator**)

Get/Set the array list holding the **RTProcessVar** variables for the indicator.

[SegmentCornerRadius](#)

Get/Set the corner radius used to draw the segment rounded rectangles.

[SegmentSpacing](#)

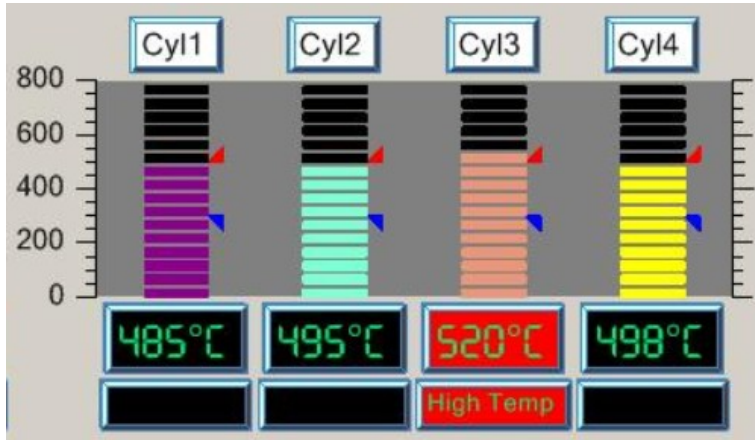
Get/Set the segments spacing for the RT_BAR_SEGMENTED_SUBTYPE and RT_BAR_SINGLE_SEGMENT_SUBTYPE bar indicator sub types.

SegmentValueRoundMode	Set/Get the segment value round mode. Specifies that the current process value is rounded up in calculating how many segments to display in RT_BAR_SEGMENTED_SUBTYPE and RT_BAR_SINGLE_SEGMENT_SUBTYPE modes. Use one of the constants: RT_FLOOR_VALUE, RT_CEILING_VALUE.
SegmentWidth	Get/Set the thickness of segments for the RT_BAR_SEGMENTED_SUBTYPE and RT_BAR_SINGLE_SEGMENT_SUBTYPE bar indicator sub types.
ShowDatapointValue (inherited from ChartPlot)	If the plot supports it, this method will turn on/off the display of data values next to the associated data point.
StepMode (inherited from ChartPlot)	Set/Get the plot objects step mode. Use one of the line plot step constants: NO_STEP, STEP_START, STEP_END, or STEP_NO_RISE_LINE.
ZOrder (inherited from GraphObj)	Sets the z-order of the object in the chart. Every object has a z-order value. Each z-order value does NOT have to be unique. If z-order sorting is turned on in the ChartView object, objects are sorted by z-order before they are drawn.

A complete listing of **RTMultiBarIndicator** properties is found in the _qcertgraphs_.html documentation file, located in the QCRTGraphTS\docs\modules subdirectory.

Example for an RTMultiBarIndicator Segmented Bar Indicator

The multi-bar indicator example below, extracted from the Dynamometer example, method **InitializeEngine1TempIndicator**, creates the 4-bar segmented bar temperature indicator in the left upper section of the graph. It demonstrates how the plot area is defined for the multi-bar indicator, how to create axes and axis labels. An **RTAlarmIndicator** is also created to display the alarm limit symbols to the right of the bar indicator. The image below also includes an **RTStringPanelMeter** for the “Cyl#” tag, an **RTNumericPanelMeter** for the numeric readout below each bar indicator, and an **RTAlarmPanelMeter** below that. See the Dynamometer example program for the complete program listing that creates all of these objects.



[TypeScript]

```
private InitializeEngine1TempIndicator() {
    if (!this.chartVu) return;
    let pTransform1: QCRTGraphTS.CartesianCoordinates =
    QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 800);
    pTransform1.setGraphBorderDiagonal(0.15, .175, 0.48, 0.35);
    let background: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(pTransform1,
    QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
    this.chartVu.addChartObject(background);
    let attrib1: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.DARKMAGENTA,
    1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.DARKMAGENTA);
    let attrib2: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.AQUAMARINE,
    1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.AQUAMARINE);
    let attrib3: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.DARKSALMON,
    1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.DARKSALMON);
    let attrib4: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.YELLOW,
    1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.YELLOW);
    let attribArray: QCRTGraphTS.ChartAttribute[] = [
        attrib1,
        attrib2,
        attrib3,
        attrib4];
    let barspace: number = 0.25;
    let barwidth: number = 0.1;
    let barbase: number = 0;
    let barjust: number = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
    let barorient: number = QCRTGraphTS.ChartConstants.VERT_DIR;
    let baraxis: QCRTGraphTS.LinearAxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
    QCRTGraphTS.ChartConstants.Y_AXIS);
    this.chartVu.addChartObject(baraxis);
    let baraxis2: QCRTGraphTS.LinearAxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
    QCRTGraphTS.ChartConstants.Y_AXIS);
    baraxis2.setAxisIntercept(pTransform1.getStopX());
    baraxis2.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MAX);
    this.chartVu.addChartObject(baraxis2);
    let barAxisLab: QCRTGraphTS.NumericAxisLabels =
    QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(baraxis);
    this.chartVu.addChartObject(barAxisLab);
    // This uses an RTMultiProcessVar (EngineCylinders1) to initialize the indicator
    let barplot: QCRTGraphTS.RTMultiBarIndicator =
    QCRTGraphTS.RTMultiBarIndicator.newRTMultiBarIndicatorFull(pTransform1, this.EngineCylinders1,
    barwidth, barspace, barbase, attribArray, barjust, barorient);
    barplot.SegmentSpacing = 50;
    barplot.SegmentWidth = 30;
}
```

152 *Multiple Channel Bar Indicator*

```
    barplot.IndicatorBackground =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.BLACK,
1,QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    barplot.SegmentValueRoundMode = QCRTGraphTS.ChartConstants.RT_CEILING_VALUE;
    barplot.SegmentCornerRadius = 0;
    barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SEGMENTED_SUBTYPE;
    let baralarms: QCRTGraphTS.RTMultiAlarmIndicator =
QCRTGraphTS.RTMultiAlarmIndicator.newRTMultiAlarmIndicator(baraxis, barplot);
    this.chartVu.addChartObject(baralarms);
.
. // Add panel meters to barplot
.
    this.chartVu.addChartObject(barplot);
}
```

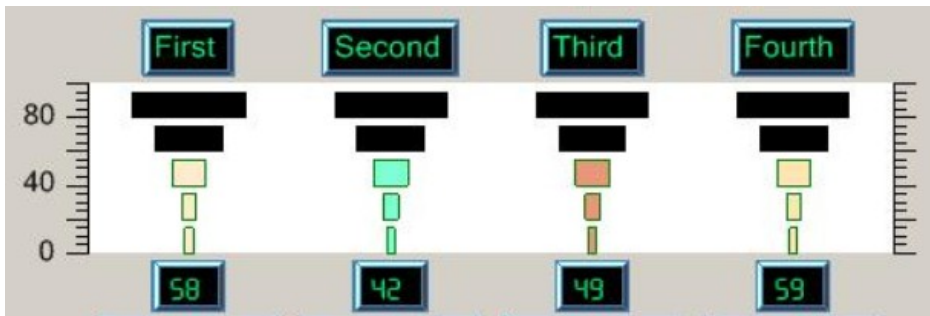
[JavaScript]

```
function InitializeEngine1TempIndicator() {
    if (!chartVu) return;
    let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 800);
    pTransform1.setGraphBorderDiagonal(0.15, .175, 0.48, 0.35);
    let background = QCRTGraphTS.Background.newBackground(pTransform1,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
    chartVu.addChartObject(background);
    let attrib1 =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.DARKMAGENTA, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.DARKMAGENTA);
    let attrib2 = QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.AQUAMARINE,
1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.AQUAMARINE);
    let attrib3 = QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.DARKSALMON,
1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.DARKSALMON);
    let attrib4 = QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.YELLOW, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.YELLOW);
    let attribArray = [
        attrib1,
        attrib2,
        attrib3,
        attrib4
    ];
    let barspace = 0.25;
    let barwidth = 0.1;
    let barbase = 0;
    let barjust = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
    let barorient = QCRTGraphTS.ChartConstants.VERT_DIR;
    let baraxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
QCRTGraphTS.ChartConstants.Y_AXIS);
    chartVu.addChartObject(baraxis);
    let baraxis2 = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
QCRTGraphTS.ChartConstants.Y_AXIS);
    baraxis2.setAxisIntercept(pTransform1.getStopX());
    baraxis2.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MAX);
    chartVu.addChartObject(baraxis2);
    let barAxisLab = QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(baraxis);
    chartVu.addChartObject(barAxisLab);
    // This uses an RTMultiProcessVar (EngineCylinders1) to initialize the indicator
    let barplot = QCRTGraphTS.RTMultiBarIndicator.newRTMultiBarIndicatorFull(pTransform1,
EngineCylinders1, barwidth, barspace, barbase, attribArray, barjust, barorient);
    barplot.SegmentSpacing = 50;
    barplot.SegmentWidth = 30;
    barplot.IndicatorBackground =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    barplot.SegmentValueRoundMode = QCRTGraphTS.ChartConstants.RT_CEILING_VALUE;
    barplot.SegmentCornerRadius = 0;
    barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SEGMENTED_SUBTYPE;
    let baralarms = QCRTGraphTS.RTMultiAlarmIndicator.newRTMultiAlarmIndicator(baraxis, barplot);
    chartVu.addChartObject(baralarms);
.
.
.
}
```

```
chartVu.addChartObject(barplot);
}
```

Example for an RTMultiBarIndicator Custom Segmented Bar Indicator

The custom bar indicator example below is extracted from the Bargraphs example, file **SolidBars**, method **InitializeBar3**. It uses a special feature that allows the width of the each bar segment to be calculated as a function of the height. This is done by subclassing the **RTBarIndicator** and **RTMultiBarIndicator** classes and overriding the **getCustomBarWidth** and **getCustomBarOffset** methods. In the example below, the width and offset of the bar is calculated using a function based on the height value. Calculating the bar width as a function of the bar height only works with the segmented bar subtypes. If you want a solid bar with custom widths, make the **RTBarIndicator.SegmentWidth** and **RTBarIndicator.SegmetSpacing** values equal and small.



[TypeScript]

```
export class CustomRTMultiBarIndicator extends QCRTGraphTS.RTMultiBarIndicator {
    public constructor() {
        super();
    }

    public static newCustomRTMultiBarIndicator(transform: QCRTGraphTS.PhysicalCoordinates,
        datasource: QCRTGraphTS.RTProcessVar[], barwidth: number,
        barspacing: number, barbase: number, attribs: QCRTGraphTS.ChartAttribute[], barjust:
number, barorient: number): CustomRTMultiBarIndicator {
        let result: CustomRTMultiBarIndicator = new CustomRTMultiBarIndicator();
        result.setChartObjScale(transform);
        result.setBarOrient(barorient);
        result.initRTMultiBarPlot(datasource, barwidth, barspacing, barbase, attribs, barjust);
        return result;
    }

    public getCustomBarOffset(v: number): number {
        // This centers the bar segments
        let offset: number = ((this.BarWidth / 2)
            - (this.getCustomBarWidth(v) / 2));
        return offset;
    }

    public getCustomBarWidth(v: number): number {
        let width: number = 0.5;
        width = (0.01 + ((v / 100) * ((v / 100) * this.BarWidth)));
        return width;
    }
}
```

154 Multiple Channel Bar Indicator

```
    }  
}  
  
private InitializeBar3() {  
    let processVarArray: QCRTGraphTS.RTProcessVar[] = [  
        this.processVar1,  
        this.processVar2,  
        this.processVar3,  
        this.processVar4];  
    if (!this.chartVu) return;  
    let pTransform1: QCRTGraphTS.CartesianCoordinates =  
QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 100);  
    pTransform1.setGraphBorderDiagonal(0.05, .475, 0.5, 0.65);  
    let background: QCRTGraphTS.Background =  
QCRTGraphTS.Background.newBackground(pTransform1, QCRTGraphTS.ChartConstants.PLOT_BACKGROUND,  
QCRTGraphTS.ChartColor.WHITE);  
    this.chartVu.addChartObject(background);  
    let attrib1: QCRTGraphTS.ChartAttribute =  
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 1,  
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLANCHEDALMOND);  
    let attrib2: QCRTGraphTS.ChartAttribute =  
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 1,  
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.AQUAMARINE);  
    let attrib3: QCRTGraphTS.ChartAttribute =  
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 1,  
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.DARKSALMON);  
    let attrib4: QCRTGraphTS.ChartAttribute =  
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 1,  
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.MOCCASIN);  
    let attribArray: QCRTGraphTS.ChartAttribute[] = [  
        attrib1,  
        attrib2,  
        attrib3,  
        attrib4];  
    let barspace: number = 0.25;  
    let barwidth: number = 0.2;  
    let barbase: number = 0;  
    let barjust: number = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;  
    let barorient: number = QCRTGraphTS.ChartConstants.VERT_DIR;  
    let baraxis: QCRTGraphTS.LinearAxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,  
QCRTGraphTS.ChartConstants.Y_AXIS);  
    this.chartVu.addChartObject(baraxis);  
    let baraxis2: QCRTGraphTS.LinearAxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,  
QCRTGraphTS.ChartConstants.Y_AXIS);  
    baraxis2.setAxisIntercept(pTransform1.getStopX());  
    baraxis2.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MAX);  
    this.chartVu.addChartObject(baraxis2);  
    let barAxisLab: QCRTGraphTS.NumericAxisLabels =  
QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(baraxis);  
    this.chartVu.addChartObject(barAxisLab);  
    let barplot: CustomRTMultiBarIndicator =  
CustomRTMultiBarIndicator.newCustomRTMultiBarIndicator(pTransform1, processVarArray, barwidth,  
barspace, barbase, attribArray, barjust, barorient);  
    barplot.SegmentSpacing = 20;  
    barplot.SegmentWidth = 15;  
  
    .  
    .  
    .  
  
    barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SEGMENTED_SUBTYPE;  
    barplot.SegmentValueRoundMode = QCRTGraphTS.ChartConstants.RT_CEILING_VALUE;  
    this.chartVu.addChartObject(barplot);  
}
```

[JavaScript]

```
export class CustomRTMultiBarIndicator extends QCRTGraphTS.RTMultiBarIndicator {  
    constructor() {  
        super();  
    }  
}
```



```

    }
    static newCustomRTMultiBarIndicator(transform, datasource, barwidth, barspacing, barbase,
    attribs, barjust, barorient) {
        let result = new CustomRTMultiBarIndicator();
        result.setChartObjScale(transform);
        result.setBarOrient(barorient);
        result.initRTMultiBarPlot(datasource, barwidth, barspacing, barbase, attribs, barjust);
        return result;
    }
    getCustomBarOffset(v) {
        // This centers the bar segments
        let offset = ((this.BarWidth / 2)
            - (this.getCustomBarWidth(v) / 2));
        return offset;
    }
    getCustomBarWidth(v) {
        let width = 0.5;
        width = (0.01 + ((v / 100) * ((v / 100) * this.BarWidth)));
        return width;
    }
}

```

```

InitializeBar3() {
    let processVarArray = [
        this.processVar1,
        this.processVar2,
        this.processVar3,
        this.processVar4
    ];
    if (!this.chartVu)
        return;
    let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 100);
    pTransform1.setGraphBorderDiagonal(0.05, .475, 0.5, 0.65);
    let background = QCRTGraphTS.Background.newBackground(pTransform1,
    QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.WHITE);
    this.chartVu.addChartObject(background);
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLANCHEDALMOND);
    let attrib2 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.AQUAMARINE);
    let attrib3 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.DARKSALMON);
    let attrib4 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.MOCCASIN);
    let attribArray = [
        attrib1,
        attrib2,
        attrib3,
        attrib4
    ];
    let barspace = 0.25;
    let barwidth = 0.2;
    let barbase = 0;
    let barjust = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
    let barorient = QCRTGraphTS.ChartConstants.VERT_DIR;
    let baraxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
    QCRTGraphTS.ChartConstants.Y_AXIS);
    this.chartVu.addChartObject(baraxis);
    let baraxis2 = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
    QCRTGraphTS.ChartConstants.Y_AXIS);
    baraxis2.setAxisIntercept(pTransform1.getStopX());
    baraxis2.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MAX);
    this.chartVu.addChartObject(baraxis2);
    let barAxisLab = QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(baraxis);
    this.chartVu.addChartObject(barAxisLab);
    let barplot = CustomRTMultiBarIndicator.newCustomRTMultiBarIndicator(pTransform1,
    processVarArray, barwidth, barspace, barbase, attribArray, barjust, barorient);
    barplot.SegmentSpacing = 20;
    barplot.SegmentWidth = 15;
}

```

156 *Multiple Channel Bar Indicator*

```
.  
    barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SEGMENTED_SUBTYPE;  
    barplot.SegmentValueRoundMode = QCRTGraphTS.ChartConstants.RT_CEILING_VALUE;  
    this.chartVu.addChartObject(barplot);  
}
```

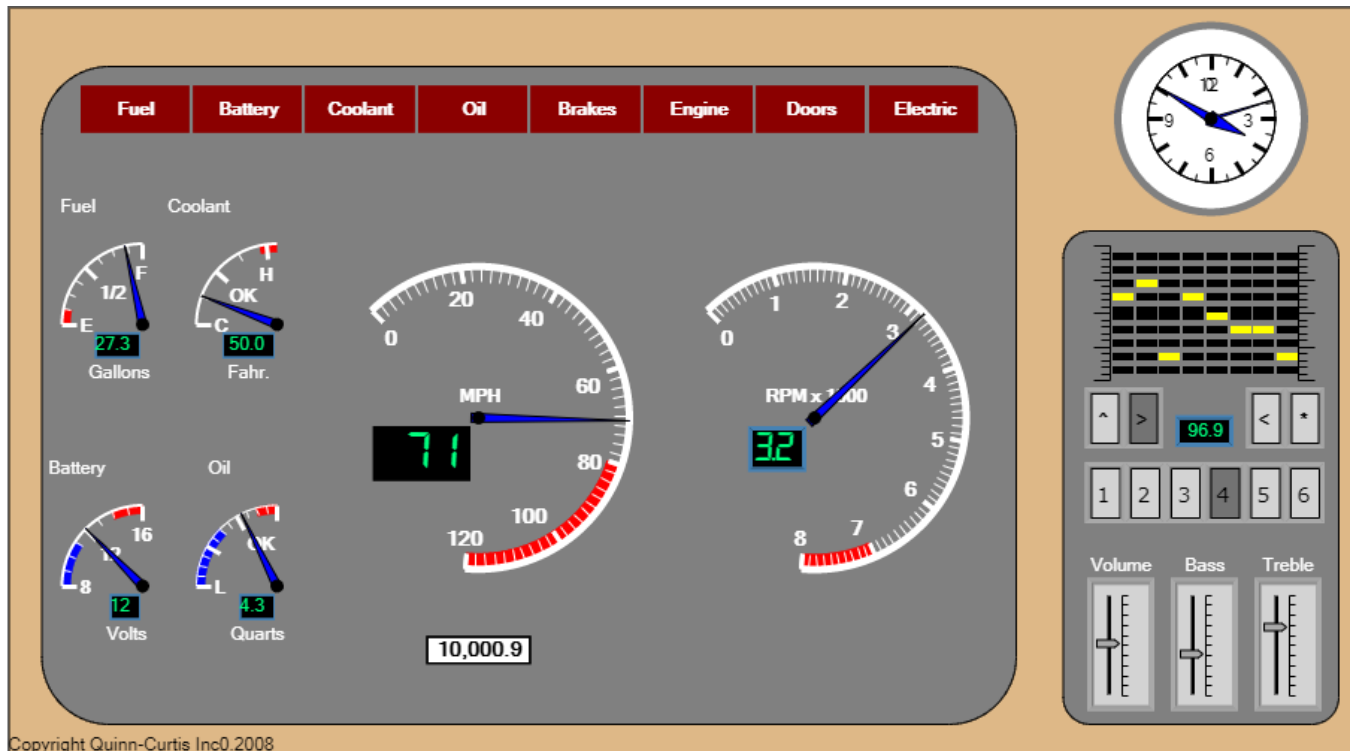

8. Meters Coordinates, Meter Axes and Meter Axis Labels

RTMeterCoordinates

RTMeterAxis

RTMeterAxisLabels

RTMeterStringAxisLabels



Familiar examples of analog meter indicators are voltmeters, car speedometers, pressure gauges, compasses and analog clock faces. A meter usually consists of a meter coordinate system, meter axes, meter axis labels, and a meter indicator (the needle, arc or symbol used to display the current value). It can also have panel meters (**RTPanelMeter** derived objects) that display the meter title, numeric readout and alarm state. The first three objects, the meter coordinate system, meter axis and meter axis labels are described in this chapter, while the meter indicator types are described in the next.

Meter Coordinates

Class **RTMeterCoordinates**

PolarCoordinates

RTMeterCoordinates

A meter coordinate system has more properties than a simple Cartesian coordinate system, or even a polar coordinate system. Because of the variation in meter styles, a meter coordinate system needs to

define the start and end angle of the meter arc within the 360 degree polar coordinate system. It also needs to map a physical coordinate system, representing the meter scale, on top of the meter arc. And the origin of the meter coordinate system can be offset in both x- and y-directions with respect to the containing plot area.

RTMeterCoordinates constructors

```
public static newRTMeterCoordinates(startarcangle: number, arcextent: number, startarcscale:
number, endarcscale: number, arcdirection: boolean, arcradius: number): RTMeterCoordinates

public static newRTMeterCoordinatesFull(startarcangle: number, arcextent: number, startarcscale:
number, endarcscale: number, arcdirection: boolean, x: number, y: number, arcradius: number):
RTMeterCoordinates
```

Parameters

startarcangle

Specifies the starting arc angle position of the meter arc in degrees.

arcextent

Specifies the extent of the meter arc in degrees. The default meter arc starts at *startArcAngle* and extends in a negative (clockwise) direction with an extent *arcExtent*.

startarcscale

Specifies the scaling value associated with the *startArcAngle* position of the meter arc.

endarcscale

Specifies the scaling value associated with the ending position of the meter arc.

arcdirection

Specifies the direction of the *arcextent*. The default *arcDirectionPositive* value of false meter arc starts at *startArcAngle* and extends in a negative (clockwise) direction with an extent *arcExtent*. Change to true to have the meter arc extend in a positive (counter-clockwise) direction.

x

Specifies x-position of the center of the meter arc in plot area normalized coordinates.

y

Specifies y-position of the center of the meter arc in plot area normalized coordinates.

arcradius

Specifies radius of the meter arc in plot area normalized coordinates.

Selected Public Instance Properties

ArcCenterX	Get/Set Specifies x-position of the center of the meter arc in plot area normalized coordinates.
ArcCenterY	Get/Set Specifies y-position of the center of the meter arc in plot area normalized

	coordinates.
ArcDirectionPositive	Get/Set the direction of the <i>arcExtent</i> . The default <i>arcDirectionPositive</i> value of false meter arc starts at <i>startArcAngle</i> and extends in a negative (clockwise) direction with an extent <i>arcExtent</i> . Change to true to have the meter arc extend in a positive (counter-clockwise) direction.
ArcExtent	Specifies the extent of the meter arc in degrees. The default meter arc starts at <i>startArcAngle</i> and extends in a negative (clockwise) direction with an extent <i>arcExtent</i> .
ArcRadius	Get/Set radius of the meter arc in plot area normalized coordinates.
EndArcScale	Get/Set the scaling value associated with the ending position of the meter arc.
StartArcAngle	Get/Set Specifies the starting arc angle position of the meter arc in degrees.
StartArcScale	Get/Set the scaling value associated with the <i>startArcAngle</i> position of the meter arc.

A complete listing of **RTMeterCoordinates** properties is found in the `_qcrtrgraphs_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Examples for meter coordinate system

The example below, extracted from the WeatherStation example, method **InitializeHumidity**, creates a meter coordinate system that starts at the arc angle of 225 degrees and has an arc extent of 270 degrees. The meter scale extends from 0.0 to 100.0 in the counterclockwise direction.



[TypeScript]

```

public InitializeHumidity() {
    if (!this.chartVu) return;
    let attrib1: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    // Fahrenheit scale
    let startarcangle: number = 225;
    let arcextent: number = 270;
    let startarcscale: number = 0;
    let endarcscale: number = 100;
    let arcdirection: boolean = false;
    let arcradius: number = 0.6;
    let centery: number = 0.2;
    let centerx: number = 0;
    let meterFont: QCRTGraphTS.ChartFont = this.font12;
    let meterframe1: QCRTGraphTS.RTMeterCoordinates =
QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle, arcextent, startarcscale,
endarcscale, arcdirection, centerx, centery, arcradius);
    .
    .
    .
}

```

[JavaScript]

```

function InitializeHumidity() {
    if (!chartVu) return;
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    // Fahrenheit scale
    let startarcangle = 225;
    let arcextent = 270;
    let startarcscale = 0;
    let endarcscale = 100;
    let arcdirection = false;
    let arcradius = 0.6;
    let centery = 0.2;
    let centerx = 0;
    let meterFont = font12;
    let meterframe1 = QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle,
arcextent, startarcscale, endarcscale, arcdirection, centerx, centery, arcradius);
    .
    .
    .
}

```

The example below, extracted from the Meters example, file ArrowMeters, method **InitializeMeter4**, creates a meter coordinate system that starts at the arc angle of 90 degrees and has an arc extent of 180 degrees. The meter scale extends from 0.0 to 100.0 in the counterclockwise direction.



[TypeScript]

```
private InitializeMeter4() {
    if (!this.chartVu) return;
    let attrib1: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle: number = 90;
    let arcextent: number = 180;
    let startarcscale: number = 0;
    let endarcscale: number = 100;
    let arcdirection: boolean = false;
    let arcradius: number = 0.6;
    let centery: number = 0;
    let centerx: number = 0.25;
    let meterFont: QCRTGraphTS.ChartFont = this.font12;
    let meterframe: QCRTGraphTS.RTMeterCoordinates =
    QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle, arcextent, startarcscale,
    endarcscale, arcdirection, centerx, centery, arcradius);
    meterframe.setGraphBorderDiagonal(0.7, 0, 0.99, 0.45);
}
```

[JavaScript]

```
function InitializeMeter4() {
    if (!chartVu) return;
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle = 90;
    let arcextent = 180;
    let startarcscale = 0;
    let endarcscale = 100;
    let arcdirection = false;
    let arcradius = 0.6;
    let centery = 0;
    let centerx = 0.25;
    let meterFont = font12;
    let meterframe = QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle,
    arcextent, startarcscale, endarcscale, arcdirection, centerx, centery, arcradius);
    meterframe.setGraphBorderDiagonal(0.7, 0, 0.99, 0.45);
}
```

Meter Axis

RTMeterAxis

LinearAxis**RTMeterAxis**

A meter axis extends for the extent of the meter arc and is centered on the origin. Major and minor tick marks are placed at evenly spaced intervals perpendicular to the meter arc. The meter axis also draws meter alarm arcs using the alarm information in the associated **RTProcessVar** object.

RTMeterAxis Constructors

```
public static newRTMeterAxis(frame: RTMeterCoordinates, graphplot: RTMeterIndicator): RTMeterAxis

public static newRTMeterAxisCoords(frame: RTMeterCoordinates): RTMeterAxis

public static newRTMeterAxisCoordsIndicator(frame: RTMeterCoordinates, graphplot:
RTMeterIndicator): RTMeterAxis

public static newRTMeterAxisCoordsIndicatorTicks(frame: RTMeterCoordinates, graphplot:
RTMeterIndicator, tickspace: number, tickspermajor: number): RTMeterAxis

public static newRTMeterAxisCoordsIndicatorTicksLen(frame: RTMeterCoordinates, graphplot:
RTMeterIndicator, tickspace: number, tickspermajor: number, minorticklength: number,
majorticklength: number, tickdir: number): RTMeterAxis
```

Parameters

frame

The **RTMeterCoordinates** object defining the meter properties for the meter axis.

graphplot

The **RTMeterIndicator** object associated with the meter axis.

tickspace

Specifies the spacing between minor tick marks, in degrees.

tickspermajor

Specifies the number of minor tick marks per major tick mark.

Selected Public Instance Properties

[AxisLabels](#) (inherited from **Axis**)

Get/Set the axis labels object associated with this axis.

[AxisLineEnable](#) (inherited from **Axis**)

Set/Get to true draws the axis line connecting the tick marks.

[AxisMajorTickLength](#) (inherited from **Axis**)

Get/Set length of a major tick mark.

[AxisMax](#) (inherited from **Axis**)

Get/Set the axis maximum value.

[AxisMin](#) (inherited from **Axis**)

Get/Set the axis minimum value.

[AxisMinorTickLength](#) (inherited from **Axis**)

Get/Set length of a minor tick mark.

[AxisMinorTicksPerMajor](#) (inherited from **Axis**)

Get/Set the number of minor tick marks per major tick mark.

[AxisTickDir](#) (inherited from **Axis**)

Get/Set the direction of a tick mark. Use one of the tick direction constants:

	AXIS_MIN, AXIS_CENTER, AXIS_MAX.
AxisTickOrigin (inherited from Axis)	Get/Set the starting point for positioning tick marks, in physical coordinates.
AxisTicksEnable (inherited from Axis)	Set/Get to true draws the axis tick marks.
AxisTickSpace (inherited from LinearAxis)	Get/Set the minor tick mark spacing.
ChartObjAttributes (inherited from GraphObj)	Sets the attributes for a chart object using a ChartAttribute object.
InnerAlarmArcNormalized	Get/Set the inner arc of the axis in normalized radius coordinates.
MajorTickLineWidth	Get/Set the major tick line width.
MeterAxisLabels	Get/Set the RTMeterAxisLabels object, if any, associated with this object.
MeterFrame	Get/Set the RTMeterCoordinates coordinate system associated with this object.
MeterIndicator	Get/Set the RTMeterIndicator object, if any, associated with this object.
MinorTickLineWidth	Get/Set the minor tick line width.
OuterAlarmArcNormalized	Get/Set the outer arc of the axis in normalized radius coordinates.
ShowAlarms	Get/Set true to show the alarm arcs.
ZOrder (inherited from GraphObj)	Sets the z-order of the object in the chart. Every object has a z-order value. Each z-order value does NOT have to be unique. If z-order sorting is turned on in the ChartView object, objects are sorted by z-order before they are drawn.

A complete listing of **RTMeterAxis** properties is found in the `_qcrgraphpts_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example for meter axis

The example below, extracted from the `AutoInstrumentPanel` example, method **InitializeTach**, creates a meter coordinate system that starts at the arc angle of 135 degrees and has an arc extent of 230 degrees. The meter scale extends from 0.0 to 8.0 in the counterclockwise direction. Two axes are created. The first is created so that it draws just the major tick marks using a thicker line width. The second uses thin tick marks for the minor tick marks of the meter axis.



[TypeScript]

```
private InitializeTach() {
    if (!this.chartVu) return;
    let attrib1: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle: number = 135;
    let arcextent: number = 230;
    let startarcscale: number = 0;
    let endarcscale: number = 8;
    let arcdirection: boolean = false;
    let arcradius: number = 0.75;
    let centery: number = 0;
    let centerx: number = 0;
    let meterframe: QCRTGraphTS.RTMeterCoordinates =
    QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle, arcextent, startarcscale,
    endarcscale, arcdirection, centerx, centery, arcradius);
    meterframe.setGraphBorderDiagonal(0.45, 0.2, 0.75, 0.9);
    let meterneedle: QCRTGraphTS.RTMeterNeedleIndicator =
    QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe, this.tach);
    meterneedle.setChartObjAttributes(attrib1);
    meterneedle.NeedleLength = 0.75;
    .
    . // Add panel meters
    .

    this.chartVu.addChartObject(meterneedle);
    let meteraxis: QCRTGraphTS.RTMeterAxis = QCRTGraphTS.RTMeterAxis.newRTMeterAxis(meterframe,
    meterneedle);
    meteraxis.setChartObjAttributes(attrib1);
    meteraxis.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MIN);
    meteraxis.LineWidth = 5;
    meteraxis.LineColor = QCRTGraphTS.ChartColor.WHITE;
    meteraxis.setAxisTickSpace(1);
    meteraxis.setAxisMinorTicksPerMajor(1);
    meteraxis.ShowAlarms = true;
    meterneedle.MeterAxis = meteraxis;
    this.chartVu.addChartObject(meteraxis);
    let meterFont: QCRTGraphTS.ChartFont = this.font14Bold;
    let meteraxislabels: QCRTGraphTS.RTMeterAxisLabels =
    QCRTGraphTS.RTMeterAxisLabels.newRTMeterAxisLabels(meteraxis);
    meteraxislabels.setTextFont(meterFont);
    meteraxislabels.LineColor = QCRTGraphTS.ChartColor.WHITE;
    meteraxislabels.setAxisLabelsDir(meteraxis.getAxisTickDir());
    meteraxislabels.OverlapLabelMode = QCRTGraphTS.ChartConstants.OVERLAP_LABEL_DRAW;
    this.chartVu.addChartObject(meteraxislabels);
}
```

```

    let meteraxis2: QCRTGraphTS.RTMeterAxis = QCRTGraphTS.RTMeterAxis.newRTMeterAxis(meterframe,
meterneedle);
    meteraxis2.setChartObjAttributes(attrib1);
    meteraxis2.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MIN);
    meteraxis2.LineWidth = 1;
    meteraxis2.LineColor = QCRTGraphTS.ChartColor.WHITE;
    meteraxis2.setAxisTickSpace(0.1);
    meteraxis2.AxisMinorTickLength = 10;
    meteraxis2.AxisMajorTickLength = 10;
    meteraxis2.setAxisMinorTicksPerMajor(10);
    meteraxis2.ShowAlarms = false;
    this.chartVu.addChartObject(meteraxis2);
}

```

[JavaScript]

```

function InitializeTach() {
    if (!chartVu) return;
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle = 135;
    let arcextent = 230;
    let startarcscale = 0;
    let endarcscale = 8;
    let arcdirection = false;
    let arcradius = 0.75;
    let centery = 0;
    let centerx = 0;
    let meterframe = QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle,
arcextent, startarcscale, endarcscale, arcdirection, centerx, centery, arcradius);
    meterframe.setGraphBorderDiagonal(0.45, 0.2, 0.75, 0.9);
    let meterneedle = QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe,
tach);
    meterneedle.setChartObjAttributes(attrib1);
    meterneedle.NeedleLength = 0.75;
    .
    . // Add panel meters
    .

    chartVu.addChartObject(meterneedle);
    let meteraxis = QCRTGraphTS.RTMeterAxis.newRTMeterAxis(meterframe, meterneedle);
    meteraxis.setChartObjAttributes(attrib1);
    meteraxis.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MIN);
    meteraxis.LineWidth = 5;
    meteraxis.LineColor = QCRTGraphTS.ChartColor.WHITE;
    meteraxis.setAxisTickSpace(1);
    meteraxis.setAxisMinorTicksPerMajor(1);
    meteraxis.ShowAlarms = true;
    meterneedle.MeterAxis = meteraxis;
    chartVu.addChartObject(meteraxis);
    let meterFont = font14Bold;
    let meteraxislabels = QCRTGraphTS.RTMeterAxisLabels.newRTMeterAxisLabels(meteraxis);
    meteraxislabels.setTextFont(meterFont);
    meteraxislabels.LineColor = QCRTGraphTS.ChartColor.WHITE;
    meteraxislabels.setAxisLabelsDir(meteraxis.getAxisTickDir());
    meteraxislabels.OverlapLabelMode = QCRTGraphTS.ChartConstants.OVERLAP_LABEL_DRAW;
    chartVu.addChartObject(meteraxislabels);
    let meteraxis2 = QCRTGraphTS.RTMeterAxis.newRTMeterAxis(meterframe, meterneedle);
    meteraxis2.setChartObjAttributes(attrib1);
    meteraxis2.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MIN);
    meteraxis2.LineWidth = 1;
    meteraxis2.LineColor = QCRTGraphTS.ChartColor.WHITE;
    meteraxis2.setAxisTickSpace(0.1);
    meteraxis2.AxisMinorTickLength = 10;
    meteraxis2.AxisMajorTickLength = 10;
    meteraxis2.setAxisMinorTicksPerMajor(10);
    meteraxis2.ShowAlarms = false;
}

```

```
chartVu.addChartObject(meteraxis2);
}
```

Numeric Meter Axis Labels

Class RTMeterAxisLabels

NumericAxisLabels

RTMeterAxisLabels

This class labels the major tick marks of the **RTMeterAxis** class. The class supports many predefined and user-definable formats, including numeric, exponent, percentage, business and currency formats.

RTMeterAxisLabels constructor

```
public static newRTMeterAxisLabels(baseaxis: RTMeterAxis): RTMeterAxisLabels
public static newRTMeterAxisLabelsBaseAxis(baseaxis: RTMeterAxis): RTMeterAxisLabels
```

Parameters

baseaxis

The **RTMeterAxis** object associated with the labels.

Selected Public Instance Properties

[AxisLabelsDecimalPos](#) (inherited from **NumericAxisLabels**)

Set/Get the number of digits to the right of the decimal point for numeric axis labels.

[AxisLabelsDir](#) (inherited from **AxisLabels**)

Set/Get the justification of the axis labels with respect to the axis tick marks. Use one of the tick direction constants: **AXIS_MIN**, **AXIS_MAX**.

[AxisLabelsEnds](#) (inherited from **AxisLabels**)

Set/Get 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**.

[AxisLabelsFormat](#) (inherited from **AxisLabels**)

Set/Get the numeric format for the axis labels.

[AxisLabelsTickOffsetX](#) (inherited from **AxisLabels**)

Set/Get the x-offset, in window device coordinates, of the label offset from the endpoint of the associated tick mark.

[AxisLabelsTickOffsetY](#) (inherited from

Set/Get the y-offset, in window device

AxisLabels)	coordinates, of the label offset from the endpoint of the associated tick mark.
ChartObjAttributes (inherited from GraphObj)	Sets the attributes for a chart object using a ChartAttribute object.
MeterAxis	Get/Set the RTMeterAxis associated with this object.
OverlapLabelMode (inherited from AxisLabels)	It is possible that axis labels overlap if the window that the axes are placed in is too small, the major tick marks are too close together, or in the case of time axis labels, too large for the current tick mark spacing. A test can be performed in the software to not display labels to overlap.
TextBgColor (inherited from ChartText)	Set/Get the color of the background rectangle under the text, if the textBgMode is true.
TextBgMode (inherited from ChartText)	Set/Get the text background color mode.
TextBoxColor (inherited from ChartText)	Set/Get if the text bounding box is drawn in the text box color.
TextBoxMode (inherited from ChartText)	Set/Get the text bounding box color.
TextFont (inherited from ChartText)	Set/Get the font of the text.
TextNudge (inherited from ChartText)	Set/Get the xy values of the textNudge property. It moves the relative position, using window device coordinates, of the text relative to the specified location of the text.
TextRotation (inherited from ChartText)	Set/Get the rotation of the text in the normal viewing plane.
ZOrder (inherited from GraphObj)	Sets the z-order of the object in the chart. Every object has a z-order value. Each z-order value does NOT have to be unique. If z-order sorting is turned on in the ChartView object, objects are sorted by z-order before they are drawn.

A complete listing of **RTMeterAxisLabels** properties is found in the `_qcrtrgraphs_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example for meter axis labels

The example below, extracted from the WeatherStation example, method **InitializeHumidity**, creates a meter coordinate system that starts at the arc angle of 225 degrees and has an arc extent of 270 degrees. The meter scale extends from 0.0 to 100.0 in the counterclockwise direction. Two axes are created. The first is created so that it draws just the major tick marks using a thicker line width. The second uses thin tick marks for the minor tick marks of the meter axis. Only the first is included below since it is the one labeled.



[TypeScript]

```
public InitializeHumidity() {
    if (!this.chartVu) return;
    let attrib1: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    // Fahrenheit scale
    let startarcangle: number = 225;
    let arcextent: number = 270;
    let startarcscale: number = 0;
    let endarcscale: number = 100;
    let arcdirection: boolean = false;
    let arcradius: number = 0.6;
    let centery: number = 0.2;
    let centerx: number = 0;
    let meterFont: QCRTGraphTS.ChartFont = this.font12;
    let meterframe1: QCRTGraphTS.RTMeterCoordinates =
    QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle, arcextent, startarcscale,
    endarcscale, arcdirection, centerx, centery, arcradius);
    let normrect: QCRTGraphTS.ChartRectangle2D =
    QCRTGraphTS.ChartRectangle2D.newChartRectangle2D(0.67, 0.05, 0.32, 0.53);
    let frame3d: QCRTGraphTS.RT3DFrame = QCRTGraphTS.RT3DFrame.newRT3DFrame(meterframe1, normrect,
    this.facePlateAttrib, QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
    this.chartVu.addChartObject(frame3d);
    meterframe1.setGraphBorderDiagonal(0.67, 0.05, 0.99, 0.58);
    let meterneedle: QCRTGraphTS.RTMeterNeedleIndicator =
    QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe1, this.humidity);
    meterneedle.setChartObjAttributes(attrib1);
    meterneedle.NeedleLength = 0.6;
    .
    . // Add Panel Meters
    .
    this.chartVu.addChartObject(meterneedle);
    let meteraxis1: QCRTGraphTS.RTMeterAxis = QCRTGraphTS.RTMeterAxis.newRTMeterAxis(meterframe1,
    meterneedle);
    meteraxis1.setChartObjAttributes(attrib1);
    meteraxis1.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MIN);
    meteraxis1.LineWidth = 3;
    meteraxis1.LineColor = QCRTGraphTS.ChartColor.BLACK;
    meteraxis1.setAxisTickSpace(20);
    meteraxis1.setAxisMinorTicksPerMajor(1);
    meteraxis1.ShowAlarms = false;
    meterneedle.MeterAxis = meteraxis1;

    this.chartVu.addChartObject(meteraxis1);
    let meteraxislabels1: QCRTGraphTS.RTMeterAxisLabels =
    QCRTGraphTS.RTMeterAxisLabels.newRTMeterAxisLabels(meteraxis1);
    meteraxislabels1.setTextFont(meterFont);
    meteraxislabels1.setAxisLabelsDir(meteraxis1.getAxisTickDir());
    meteraxislabels1.OverlapLabelMode = QCRTGraphTS.ChartConstants.OVERLAP_LABEL_DRAW;
    this.chartVu.addChartObject(meteraxislabels1);
}
```

```

.
.
.
}

```

[JavaScript]

```

function InitializeHumidity() {
    if (!chartVu) return;
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    // Fahrenheit scale
    let startarcangle = 225;
    let arcextent = 270;
    let startarcscale = 0;
    let endarcscale = 100;
    let arcdirection = false;
    let arcradius = 0.6;
    let centery = 0.2;
    let centerx = 0;
    let meterFont = font12;
    let meterframe1 = QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle,
arcextent, startarcscale, endarcscale, arcdirection, centerx, centery, arcradius);
    let normrect = QCRTGraphTS.ChartRectangle2D.newChartRectangle2D(0.67, 0.05, 0.32, 0.53);
    let frame3d = QCRTGraphTS.RT3DFrame.newRT3DFrame(meterframe1, normrect, facePlateAttrib,
QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
    chartVu.addChartObject(frame3d);
    meterframe1.setGraphBorderDiagonal(0.67, 0.05, 0.99, 0.58);
    let meterneedle = QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe1,
humidity);
    meterneedle.setChartObjAttributes(attrib1);
    meterneedle.NeedleLength = 0.6;
    let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmetertagattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
    let panelmeter3 =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(meterframe1,
humidity, panelmetertagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
    panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.RADIUS_CENTER;
    panelmeter3.StringTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
    panelmeter3.StringTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
    panelmeter3.PanelMeterNudge = QCRTGraphTS.ChartPoint2D.newChartPoint2D(0, -8);
    panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
    panelmeter3.StringTemplate.TextFont = font12Bold;
    panelmeter3.Frame3DEnable = false;
    meterneedle.addPanelMeter(panelmeter3);
.
.    // Add Panel Meters
.

    meterneedle.addPanelMeter(panelmeter);
    chartVu.addChartObject(meterneedle);
    let meteraxis1 = QCRTGraphTS.RTMeterAxis.newRTMeterAxis(meterframe1, meterneedle);
    meteraxis1.setChartObjAttributes(attrib1);
    meteraxis1.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MIN);
    meteraxis1.LineWidth = 3;
    meteraxis1.LineColor = QCRTGraphTS.ChartColor.BLACK;
    meteraxis1.setAxisTickSpace(20);
    meteraxis1.setAxisMinorTicksPerMajor(1);
    meteraxis1.ShowAlarms = false;
    meterneedle.MeterAxis = meteraxis1;
    chartVu.addChartObject(meteraxis1);
    let meteraxislabels1 = QCRTGraphTS.RTMeterAxisLabels.newRTMeterAxisLabels(meteraxis1);
    meteraxislabels1.setTextFont(meterFont);
    meteraxislabels1.setAxisLabelsDir(meteraxis1.getAxisTickDir());
    meteraxislabels1.OverlapLabelMode = QCRTGraphTS.ChartConstants.OVERLAP_LABEL_DRAW;

```



```

    chartVu.addChartObject(meteraxislabels1);
    .
    .
    .
}

```

String Meter Axis Labels

Class RTMeterStringAxisLabels

StringAxisLabels

RTMeterStringAxisLabels

This class labels the major tick marks of the **RTMeterAxis** class using user-defined strings

RTMeterStringAxisLabels constructor

```

public static newRTMeterStringAxisLabels(baseaxis: RTMeterAxis): RTMeterStringAxisLabels
public static newRTMeterStringAxisLabelsBaseAxis(baseaxis: RTMeterAxis): RTMeterStringAxisLabels

```

Parameters

baseaxis

The **RTMeterAxis** object associated with the labels.

Parameters

baseaxis

The **RTMeterAxis** object associated with the labels.

Selected Public Instance Properties

The properties of the **RTMeterStringAxisLabels** class are pretty much the same as the **RTMeterAxisLabels** class, with these exceptions.

MeterLabelTextOrient	Get/Set if the text is horizontal (METER_LABEL_HORIZONTAL) at right angles to the tick mark (METER_LABEL_PERPENDICULAR), or radial to the tick mark parallel (METER_LABEL_RADIAL_1, METER_LABEL_RADIAL_2).

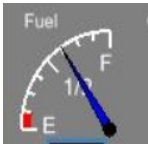
The axis label strings are set using the **setAxisLabelString** method.

SetAxisLabelsStrings (inherited from StringAxisLabels)	Sets the string array used to hold user defined axis label strings. Setting the string array does not automatically turn on the use of string labels. Use enableAxisLabelsStrings to enable axis strings.

A complete listing of **RTMeterStringAxisLabels** properties is found in the `_qcertgraphs_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example for string meter axis labels

The example below, extracted from the `AutoInstrumentPanel` example, method **InitializeFuel**, creates a meter coordinate system that starts at the arc angle of 180 degrees and has an arc extent of 90 degrees. The meter scale extends from 0.0 to 32.0 in the counterclockwise direction. The meter axis is labeled at the major tick marks with the strings `{"E", "1/2", "F"}`.



[TypeScript]

```
private InitializeFuel() {
    if (!this.chartVu) return;
    let attrib1: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle: number = 180;
    let arcextent: number = 90;
    let startarcscale: number = 0;
    let endarcscale: number = 32;
    let arcdirection: boolean = false;
    let arcradius: number = 0.8;
    let centery: number = 0;
    let centerx: number = 0;
    let meterframe: QCRTGraphTS.RTMeterCoordinates =
QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle, arcextent, startarcscale,
endarcscale, arcdirection, centerx, centery, arcradius);
    meterframe.setGraphBorderDiagonal(0.025, 0.25, 0.175, 0.6);
    let meterneedle: QCRTGraphTS.RTMeterNeedleIndicator =
QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe, this.fuel);
    meterneedle.setChartObjAttributes(attrib1);
    meterneedle.NeedleLength = 0.8;
.
.    // Add Panel Meters
.

    this.chartVu.addChartObject(meterneedle);
    let meteraxis: QCRTGraphTS.RTMeterAxis = QCRTGraphTS.RTMeterAxis.newRTMeterAxis(meterframe,
meterneedle);
    meteraxis.setChartObjAttributes(attrib1);
    meteraxis.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MIN);
    meteraxis.LineWidth = 2;
```

```

meteraxis.LineColor = QCRTGraphTS.ChartColor.WHITE;
meteraxis.setAxisTickSpace(4);
meteraxis.setAxisMinorTicksPerMajor(4);
meteraxis.ShowAlarms = true;
meterneedle.MeterAxis = meteraxis;
this.chartVu.addChartObject(meteraxis);
let meterFont: QCRTGraphTS.ChartFont = this.font12Bold;
let meteraxislabels: QCRTGraphTS.RTMeterStringAxisLabels =
QCRTGraphTS.RTMeterStringAxisLabels.newRTMeterStringAxisLabels(meteraxis);
meteraxislabels.setTextFont(meterFont);
meteraxislabels.setAxisLabelsDir(meteraxis.getAxisTickDir());
let labelstrings: string[] = [
    "E",
    "1/2",
    "F"];
meteraxislabels.OverlapLabelMode = QCRTGraphTS.ChartConstants.OVERLAP_LABEL_DRAW;
meteraxislabels.AxisLabelsEnds = QCRTGraphTS.ChartConstants.LABEL_MAX;
meteraxislabels.setAxisLabelsStrings(labelstrings, 3);
meteraxislabels.LineColor = QCRTGraphTS.ChartColor.WHITE;
this.chartVu.addChartObject(meteraxislabels);
}

```

[JavaScript]

```

function InitializeFuel() {
    if (!chartVu) return;
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle = 180;
    let arcextent = 90;
    let startarcscale = 0;
    let endarcscale = 32;
    let arcdirection = false;
    let arcradius = 0.8;
    let centery = 0;
    let centerx = 0;
    let meterframe = QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle,
arcextent, startarcscale, endarcscale, arcdirection, centerx, centery, arcradius);
    meterframe.setGraphBorderDiagonal(0.025, 0.25, 0.175, 0.6);
    let meterneedle = QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe,
fuel);
    meterneedle.setChartObjAttributes(attrib1);
    meterneedle.NeedleLength = 0.8;
    let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    .
    . // Add Panel Meters
    .
    chartVu.addChartObject(meterneedle);
    let meteraxis = QCRTGraphTS.RTMeterAxis.newRTMeterAxis(meterframe, meterneedle);
    meteraxis.setChartObjAttributes(attrib1);
    meteraxis.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MIN);
    meteraxis.LineWidth = 2;
    meteraxis.LineColor = QCRTGraphTS.ChartColor.WHITE;
    meteraxis.setAxisTickSpace(4);
    meteraxis.setAxisMinorTicksPerMajor(4);
    meteraxis.ShowAlarms = true;
    meterneedle.MeterAxis = meteraxis;
    chartVu.addChartObject(meteraxis);
    let meterFont = font12Bold;
    let meteraxislabels =
QCRTGraphTS.RTMeterStringAxisLabels.newRTMeterStringAxisLabels(meteraxis);
    meteraxislabels.setTextFont(meterFont);
    meteraxislabels.setAxisLabelsDir(meteraxis.getAxisTickDir());
    let labelstrings = [
        "E",
        "1/2",
        "F"
    ];
}

```

```
meteraxislabels.OverlapLabelMode = QCRTGraphTS.ChartConstants.OVERLAP_LABEL_DRAW;
meteraxislabels.AxisLabelsEnds = QCRTGraphTS.ChartConstants.LABEL_MAX;
meteraxislabels.setAxisLabelsStrings(labelstrings, 3);
meteraxislabels.LineColor = QCRTGraphTS.ChartColor.WHITE;
chartVu.addChartObject(meteraxislabels);
}
```

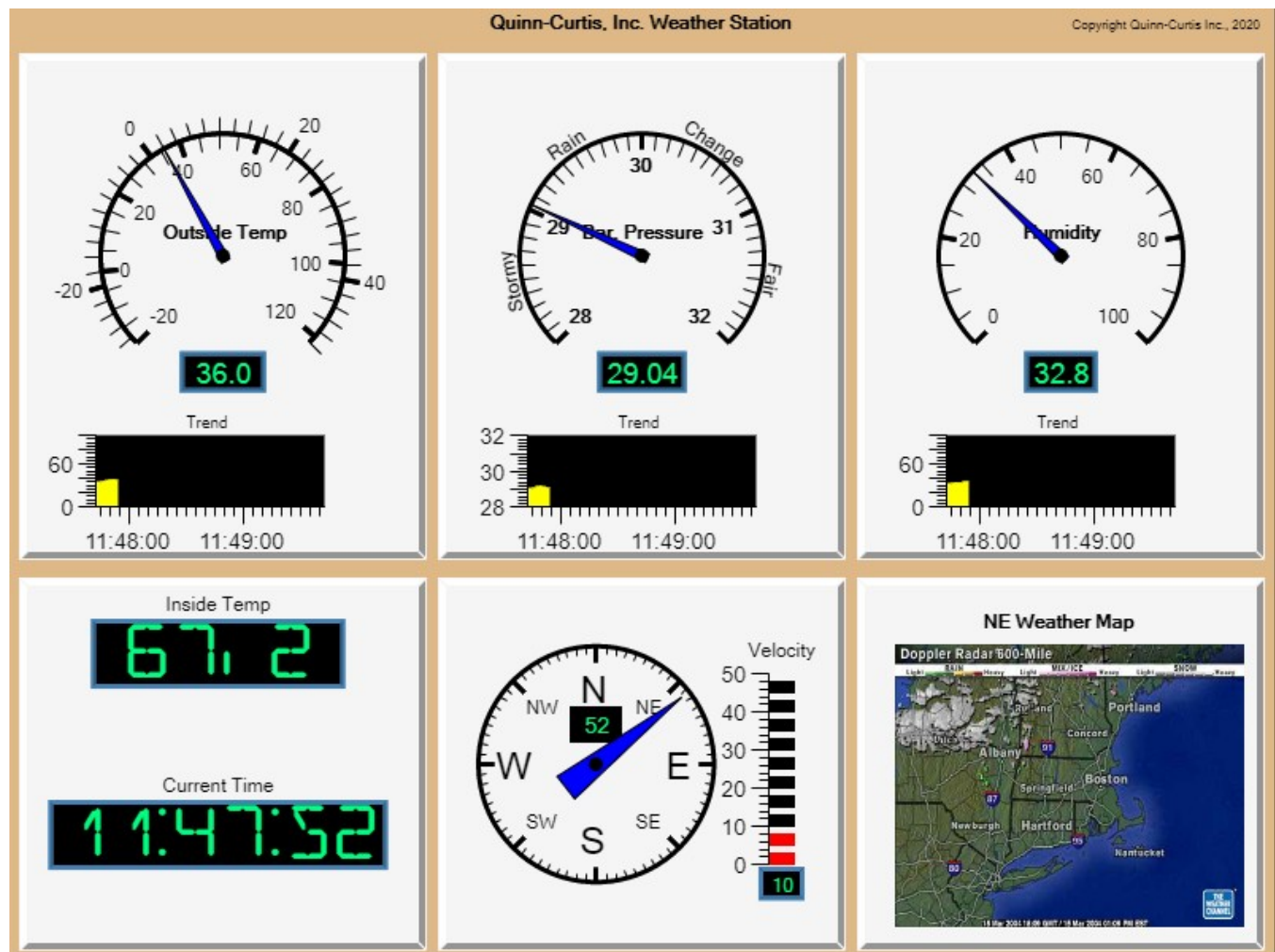
9. Meter Indicators: Needle, Arc and Symbol

RTMeterIndicator

RTMeterArcIndicator

RTMeterNeedleIndicator

RTMeterSymbolIndicator



Familiar examples of analog meter indicators are voltmeters, car speedometers, pressure gauges, compasses and analog clock faces. Three meter indicator types are supported: arc, symbol, and needle meters. An unlimited number of meter indicators can be added to a given meter object. **RTPanelMeter** objects can be attached to an **RTMeterIndicator** object for the display of **RTProcessVar** numeric, alarm and string data in addition to the indicator graphical display. Meter scaling, meter axes, meter axis labels and alarm objects and handle by the meter coordinate system, meter axis and meter axis labels classes described in the preceding chapter.

Meter Indicators

Class RTMeterIndicator

ChartPlot

RTPlot

RTSingleValueIndicator

RTMeterIndicator

The **RTMeterIndicator** class is the abstract base class for all meter indicators. Since it is abstract it does not have a constructor that you can use. It does have properties common to all meter indicator types and these are listed here.

Selected Public Instance Properties

AlarmIndicatorColorMode (inherited from RTSingleValueIndicator)	Get/Set whether the color of the indicator objects changes on an alarm. Use one of the constants: RT_INDICATOR_COLOR_NO_ALARM_CHANGE, RT_INDICATOR_COLOR_CHANGE_ON_ALARM..
ChartObjAttributes (inherited from GraphObj)	Sets the attributes for a chart object using a ChartAttribute object.
ChartObjEnable (inherited from GraphObj)	Enables/Disables the chart object. A chart object is drawn only if it is enabled. A chart object is enabled by default.
ChartObjScale (inherited from GraphObj)	Sets the reference to the PhysicalCoordinates object that the chart object is placed in
CurrentProcessValue (inherited from RTSingleValueIndicator)	Get the current process value of the primary channel.
FillColor (inherited from GraphObj)	Sets the fill color for the chart object.
IndicatorBackground	Get/Set the background attribute of the meter indicator.
IndicatorBackgroundEnable	Set to true to enable the display of the meter indicator background.
IndicatorSubtype	Set/Get the meter indicator subtype. Use one of the meter indicator subtype constants: RT_METER_NEEDLE_SIMPLE_SUBTYPE , RT_METER_NEEDLE_PIEWEDGE_SUBTYPE, RT_METER_NEEDLE_ARROW_SUBTYPE, RT_METER_ARC_BAND_SUBTYPE, RT_METER_SEGMENTED_ARC_SUBTYPE, RT_METER_SINGLE_SEGMENT_ARC_SUBTYPE, RT_METER_SYMBOL_ARC_SUBTYPE, RT_METER_SINGLE_SYMBOL_SUBTYPE.
LineColor (inherited from GraphObj)	Sets the line color for the chart object.
LineStyle (inherited from	Sets the line style for the chart object.

GraphObj)	
LineWidth (inherited from GraphObj)	Sets the line width for the chart object.
MeterAxis	Get/Set the reference meter axis.
NumChannels (inherited from RTPlot)	Get the number of channels in the indicator.
OverRangeNormalizedValue	Get/Set the displayable high end of the indicator range as a normalized value based on the RTMeterCoordinates and RTMeterAxis scale. For example, if the RTMeterAxis scale is 0 to 10, an overRangeNormalizedValue of 0.1 will allow the indicator to display off-scale up to 11.0.
PanelMeterList (inherited from RTPlot)	Set/Get the panel meter list of the RTPlot object.
PrimaryChannel (inherited from RTPlot)	Set/Get the primary channel of the indicator.
RTDataSource (inherited from RTSingleValueIndicator)	Get/Set the array list holding the RTProcessVar variables for the indicator.
UnderRangeNormalizedValue	Get/Set the displayable low end of the indicator range as a normalized value based on the RTMeterCoordinates and RTMeterAxis scale. For example, if the RTMeterAxis scale is 0 to 10, an underRangeNormalizedValue of -0.1 will allow the indicator to display off-scale down to -1.
ZOrder (inherited from GraphObj)	Sets the z-order of the object in the chart. Every object has a z-order value. Each z-order value does NOT have to be unique. If z-order sorting is turned on in the ChartView object, objects are sorted by z-order before they are drawn.

A complete listing of **RTMeterIndicator** properties is found in the `_qcertgraphs_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Arc Meter Indicator

RTMeterArcIndicator

ChartPlot

RTPlot

RTSingleValueIndicator

RTMeterIndicator

RTMeterArcIndicator

This **RTMeterArcIndicator** class displays the current **RTProcessVar** value as an arc. Segmented meter arcs are one of the **RTMeterArcIndicator** subtypes. Varying the thickness of the arc, the segment width and segment spacing, and the segment end caps, will produce a wide variety of meter indicators. One of the advantages of the meter arc indicator is that it can be hollow in the center, allowing for the placement of a numeric panel meter as a digital readout in the center of the meter.

RTMeterArcIndicator constructor

```
public static newRTMeterArcIndicator(frame: RTMeterCoordinates, datasource: RTProcessVar):
RTMeterArcIndicator

public static newRTMeterArcIndicatorCoordsProcessVar(frame: RTMeterCoordinates, datasource:
RTProcessVar): RTMeterArcIndicator

public static newRTMeterArcIndicatorCoordsProcessVarAttrib(frame: RTMeterCoordinates, datasource:
RTProcessVar, innerarc: number, outerarc: number, attrib: ChartAttribute): RTMeterArcIndicator
```

Parameters

frame

The **RTMeterCoordinates** object defining the meter properties for the indicator.

datasource

The process variable associated with the indicator.

innerarc

The inner radius value in radius normalized units (0.0-1.0).

outerarc

The inner radius value in radius normalized units (0.0-1.0).

attrib

The color attributes of the indicator.

Selected Public Instance Properties

[IndicatorSubtype](#)
(inherited from
RTMeterIndicator)

Set/Get the meter indicator subtype. Use one of the arc meter indicator subtype constants:
RT_METER_ARC_BAND_SUBTYPE,
RT_METER_SEGMENTED_ARC_SUBTYPE,
RT_METER_SINGLE_SEGMENT_ARC_SUBTYPE.

[InnerArcCapStyle](#)

Set/Get the inner arc cap style. Use one of the constants:RT_METER_ARC_RADIUS_CAP,
RT_METER_ARC_WEDGE_WIDTH_CAP,
RT_METER_ARC_FLAT_CAP.

[InnerValueArcNormalized](#)

Set/Get the value of the inner arc radius in normalized radius coordinates.

[OuterArcCapStyle](#)

Set/Get the outer arc cap style. Use one of the constants:RT_METER_ARC_RADIUS_CAP,
RT_METER_ARC_WEDGE_WIDTH_CAP,
RT_METER_ARC_FLAT_CAP.

OuterValueArcNormalized	Set/Get the value of the outer arc radius in normalized radius coordinates.
SegmentSpacing	Set/Get the spacing of the arc segments in degrees.
SegmentValueRoundMode	Set/Get how the current process value is rounded up in calculating how many segments to display in RT_METER_SEGMENTED_ARC_SUBTYPE, RT_METER_SINGLE_SEGMENT_ARC_SUBTYPE modes. Use one of the constants: RT_FLOOR_VALUE, RT_CEILING_VALUE.
SegmentWidth	Set/Get the value of the arc segment width in degrees.

A complete listing of **RTMeterArcIndicator** properties is found in the `_qcrgraphts_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

In the single segment arc indicator subtype (**RTMeterArcIndicator.IndicatorSubType** = `METER_SINGLE_SEGMENT_ARC_SUBTYPE`), only the last segment is “on”. The segments up to but not including the final segment are turned “off”,

Examples for arc meter indicators

The examples below are program segments that give the important aspects of configuration an arc meter indicator for the image above it.

Extracted from the example program Meters, file ArcMeters, method **InitializeMeter1**.



[TypeScript]

```
private InitializeMeter1() {
    if (!this.chartVu) return;
    let attrib1: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle: number = 220;
    let arcextent: number = 260;
    let startarcscale: number = 0;
    let endarcscale: number = 100;
```

```

let arcdirection: boolean = false;
let arcradius: number = 0.5;
let centery: number = -0.02;
let centerx: number = 0;
let meterFont: QCRTGraphTS.ChartFont = this.font12;
let meterframe: QCRTGraphTS.RTMeterCoordinates =
QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle, arcextent, startarcscale,
endarcscale, arcdirection, centerx, centery, arcradius);
meterframe.setGraphBorderDiagonal(0, 0, 0.25, 0.45);
let background: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(meterframe,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
this.chartVu.addChartObject(background);
let meterarcindicator: QCRTGraphTS.RTMeterArcIndicator =
QCRTGraphTS.RTMeterArcIndicator.newRTMeterArcIndicator(meterframe, this.processVar1);
meterarcindicator.setChartObjAttributes(attrib1);
meterarcindicator.IndicatorSubtype = QCRTGraphTS.ChartConstants.RT_METER_ARC_BAND_SUBTYPE;
meterarcindicator.InnerValueArcNormalized = 0.65;
meterarcindicator.OuterValueArcNormalized = 0.85;
meterarcindicator.IndicatorBackgroundEnable = true;
meterarcindicator.IndicatorBackground =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 2,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.fromRgb(60, 60, 60));
.
. // Add Panel Meters
.

this.chartVu.addChartObject(meterarcindicator);

```

[JavaScript]

```

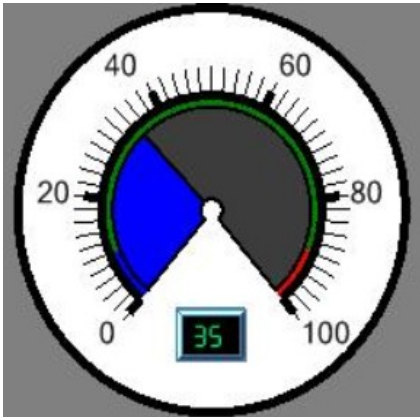
function InitializeMeter1() {
  if (!chartVu) return;
  let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
  let startarcangle = 220;
  let arcextent = 260;
  let startarcscale = 0;
  let endarcscale = 100;
  let arcdirection = false;
  let arcradius = 0.5;
  let centery = -0.02;
  let centerx = 0;
  let meterFont = font12;
  let meterframe = QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle,
arcextent, startarcscale, endarcscale, arcdirection, centerx, centery, arcradius);
  meterframe.setGraphBorderDiagonal(0, 0, 0.25, 0.45);
  let background = QCRTGraphTS.Background.newBackground(meterframe,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
  chartVu.addChartObject(background);
  let meterarcindicator =
QCRTGraphTS.RTMeterArcIndicator.newRTMeterArcIndicator(meterframe, processVar1);
  meterarcindicator.setChartObjAttributes(attrib1);
  meterarcindicator.IndicatorSubtype = QCRTGraphTS.ChartConstants.RT_METER_ARC_BAND_SUBTYPE;
  meterarcindicator.InnerValueArcNormalized = 0.65;
  meterarcindicator.OuterValueArcNormalized = 0.85;
  meterarcindicator.IndicatorBackgroundEnable = true;
  meterarcindicator.IndicatorBackground =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 2,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.fromRgb(60, 60, 60));
.
. // Add Panel Meters
.

chartVu.addChartObject(meterarcindicator);

```

181 Meter Indicators – Needle, Arc and Symbol

Extracted from the example program Meters, file ArcMeters, method **InitializeMeter4**.



[TypeScript]

```
private InitializeMeter4() {
    if (!this.chartVu) return;
    let attrib1: QCRTGraphTS.ChartAttribute =
        QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 2,
        QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle: number = 230;
    let arcextent: number = 280;
    let startarcscale: number = 0;
    let endarcscale: number = 100;
    let arcdirection: boolean = false;
    let arcradius: number = 0.45;
    let centery: number = 0;
    let centerx: number = 0;
    let meterFont: QCRTGraphTS.ChartFont = this.font12;
    let meterframe: QCRTGraphTS.RTMeterCoordinates =
        QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle, arcextent, startarcscale,
        endarcangle, arcdirection, centerx, centery, arcradius);
    meterframe.setGraphBorderDiagonal(0.7, 0, 0.99, 0.45);
    let background: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(meterframe,
        QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
    this.chartVu.addChartObject(background);
    let meterarcindicator: QCRTGraphTS.RTMeterArcIndicator =
        QCRTGraphTS.RTMeterArcIndicator.newRTMeterArcIndicator(meterframe, this.processVar1);
    meterarcindicator.setChartObjAttributes(attrib1);
    meterarcindicator.IndicatorSubtype = QCRTGraphTS.ChartConstants.RT_METER_ARC_BAND_SUBTYPE;
    meterarcindicator.InnerValueArcNormalized = 0.1;
    meterarcindicator.OuterValueArcNormalized = 0.9;
    meterarcindicator.IndicatorBackgroundEnable = true;
    meterarcindicator.IndicatorBackground =
        QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 2,
        QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.fromRgb(60, 60, 60));
    let panelmeterattrib: QCRTGraphTS.ChartAttribute =
        QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
        QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter: QCRTGraphTS.RTNumericPanelMeter =
        QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(meterframe, this.processVar1,
        panelmeterattrib);
    panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.RADIUS_BOTTOM;
    panelmeter.NumericTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
    panelmeter.NumericTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_MAX;
```

```

panelmeter.NumericTemplate.TextFont =this.font12Numeric;
meterarcindicator.addPanelMeter(panelmeter);
this.chartVu.addChartObject(meterarcindicator);

```

[JavaScript]

```

function InitializeMeter4() {
    if (!chartVu) return;
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 2,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle = 230;
    let arcextent = 280;
    let startarcscale = 0;
    let endarcscale = 100;
    let arcdirection = false;
    let arcradius = 0.45;
    let centery = 0;
    let centerx = 0;
    let meterFont =font12;
    let meterframe = QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle,
arcextent, startarcscale, endarcscale, arcdirection, centerx, centery, arcradius);
    meterframe.setGraphBorderDiagonal(0.7, 0, 0.99, 0.45);
    let background = QCRTGraphTS.Background.newBackground(meterframe,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
    chartVu.addChartObject(background);
    let meterarcindicator =
QCRTGraphTS.RTMeterArcIndicator.newRTMeterArcIndicator(meterframe,processVar1);
    meterarcindicator.setChartObjAttributes(attrib1);
    meterarcindicator.IndicatorSubtype = QCRTGraphTS.ChartConstants.RT_METER_ARC_BAND_SUBTYPE;
    meterarcindicator.InnerValueArcNormalized = 0.1;
    meterarcindicator.OuterValueArcNormalized = 0.9;
    meterarcindicator.IndicatorBackgroundEnable = true;
    meterarcindicator.IndicatorBackground =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 2,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.fromRgb(60, 60, 60));
    let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter =
QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(meterframe,processVar1, panelmeterattrib);
    panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.RADIUS_BOTTOM;
    panelmeter.NumericTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
    panelmeter.NumericTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_MAX;
    panelmeter.NumericTemplate.TextFont =font12Numeric;
    meterarcindicator.addPanelMeter(panelmeter);

    chartVu.addChartObject(meterarcindicator);

```

Extracted from the example program Meters, file SegmentedMeters, method **InitializeMeter1**.

183 Meter Indicators – Needle, Arc and Symbol



[TypeScript]

```
private InitializeMeter1() {
    if (!this.chartVu) return;
    let attrib1: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle: number = 220;
    let arcextent: number = 260;
    let startarcscale: number = 0;
    let endarcscale: number = 100;
    let arcdirection: boolean = false;
    let arcradius: number = 0.5;
    let centery: number = -0.02;
    let centerx: number = 0;
    let meterFont: QCRTGraphTS.ChartFont =this.font12;
    let meterframe: QCRTGraphTS.RTMeterCoordinates =
QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle, arcextent, startarcscale,
endarcscale, arcdirection, centerx, centery, arcradius);
    meterframe.setGraphBorderDiagonal(0, 0, 0.25, 0.45);
    let background: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(meterframe,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
    this.chartVu.addChartObject(background);
    let meterarcindicator: QCRTGraphTS.RTMeterArcIndicator =
QCRTGraphTS.RTMeterArcIndicator.newRTMeterArcIndicator(meterframe, this.processVar1);
    meterarcindicator.setChartObjAttributes(attrib1);
    meterarcindicator.IndicatorSubtype =
QCRTGraphTS.ChartConstants.RT_METER_SEGMENTED_ARC_SUBTYPE;
    meterarcindicator.InnerValueArcNormalized = 0.65;
    meterarcindicator.OuterValueArcNormalized = 0.85;
    meterarcindicator.IndicatorBackgroundEnable = true;
    meterarcindicator.IndicatorBackground =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 2,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.fromRgb(60, 60, 60));

    // Add Panel Meters
    this.chartVu.addChartObject(meterarcindicator);
}
```

[JavaScript]

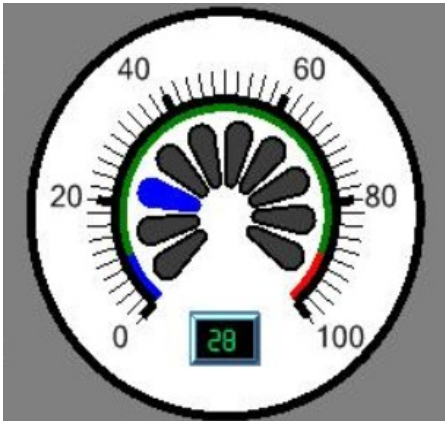
```
function InitializeMeter1() {
    if (!chartVu) return;
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle = 220;
    let arcextent = 260;
```

```

let startarcscale = 0;
let endarcscale = 100;
let arcdirection = false;
let arcradius = 0.5;
let centery = -0.02;
let centerx = 0;
let meterFont = font12;
let meterframe = QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle,
arcentent, startarcscale, endarcscale, arcdirection, centerx, centery, arcradius);
meterframe.setGraphBorderDiagonal(0, 0, 0.25, 0.45);
let background = QCRTGraphTS.Background.newBackground(meterframe,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
chartVu.addChartObject(background);
let meterarcindicator = QCRTGraphTS.RTMeterArcIndicator.newRTMeterArcIndicator(meterframe,
processVar1);
meterarcindicator.setChartObjAttributes(attrib1);
meterarcindicator.IndicatorSubtype =
QCRTGraphTS.ChartConstants.RT_METER_SEGMENTED_ARC_SUBTYPE;
meterarcindicator.InnerValueArcNormalized = 0.65;
meterarcindicator.OuterValueArcNormalized = 0.85;
meterarcindicator.IndicatorBackgroundEnable = true;
meterarcindicator.IndicatorBackground =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 2,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.fromRgb(60, 60, 60));
.
. // Add Panel Meters
.
chartVu.addChartObject(meterarcindicator);

```

Extracted from the example program Meters, file SegmentedMeters.ts, method **InitializeMeter4**.



[TypeScript]

```

private InitializeMeter4() {
    if (!this.chartVu) return;
    let attrib1: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 2,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle: number = 230;
    let arcentent: number = 280;
    let startarcscale: number = 0;
    let endarcscale: number = 100;
    let arcdirection: boolean = false;
    let arcradius: number = 0.45;
    let centery: number = 0;
    let centerx: number = 0;
    let meterFont: QCRTGraphTS.ChartFont =this.font12;

```

185 Meter Indicators – Needle, Arc and Symbol

```
let meterframe: QCRTGraphTS.RTMeterCoordinates =
QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle, arcextent, startarcscale,
endarcscale, arcdirection, centerx, centery, arcradius);
meterframe.setGraphBorderDiagonal(0.7, 0, 0.99, 0.45);
let background: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(meterframe,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
this.chartVu.addChartObject(background);
let meterarcindicator: QCRTGraphTS.RTMeterArcIndicator =
QCRTGraphTS.RTMeterArcIndicator.newRTMeterArcIndicator(meterframe, this.processVar1);
meterarcindicator.setChartObjAttributes(attrib1);
meterarcindicator.IndicatorSubtype =
QCRTGraphTS.ChartConstants.RT_METER_SEGMENTED_ARC_SUBTYPE;
meterarcindicator.InnerValueArcNormalized = 0.1;
meterarcindicator.OuterValueArcNormalized = 0.9;
meterarcindicator.IndicatorBackgroundEnable = true;
meterarcindicator.IndicatorBackground =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 2,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.fromRgb(60, 60, 60));
let panelmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let panelmeter: QCRTGraphTS.RTNumericPanelMeter =
QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(meterframe, this.processVar1,
panelmeterattrib);
panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.RADIUS_BOTTOM;
panelmeter.NumericTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
panelmeter.NumericTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_MAX;
panelmeter.NumericTemplate.TextFont = this.font12Numeric;
meterarcindicator.addPanelMeter(panelmeter);
this.chartVu.addChartObject(meterarcindicator);
```

[JavaScript]

```
function InitializeMeter4() {
  if (!chartVu) return;
  let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 2,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
  let startarcangle = 230;
  let arcextent = 280;
  let startarcscale = 0;
  let endarcscale = 100;
  let arcdirection = false;
  let arcradius = 0.45;
  let centery = 0;
  let centerx = 0;
  let meterFont = font12;
  let meterframe = QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle,
arcextent, startarcscale, endarcscale, arcdirection, centerx, centery, arcradius);
  meterframe.setGraphBorderDiagonal(0.7, 0, 0.99, 0.45);
  let background = QCRTGraphTS.Background.newBackground(meterframe,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
  chartVu.addChartObject(background);
  let meterarcindicator = QCRTGraphTS.RTMeterArcIndicator.newRTMeterArcIndicator(meterframe,
processVar1);
  meterarcindicator.setChartObjAttributes(attrib1);
  meterarcindicator.IndicatorSubtype =
QCRTGraphTS.ChartConstants.RT_METER_SEGMENTED_ARC_SUBTYPE;
  meterarcindicator.InnerValueArcNormalized = 0.1;
  meterarcindicator.OuterValueArcNormalized = 0.9;
  meterarcindicator.IndicatorBackgroundEnable = true;
  meterarcindicator.IndicatorBackground =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 2,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.fromRgb(60, 60, 60));
  let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
```

```
let panelmeter = QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(meterframe,
processVar1, panelmeterattrib);
panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.RADIUS_BOTTOM;
panelmeter.NumericTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
panelmeter.NumericTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_MAX;
panelmeter.NumericTemplate.TextFont = font12Numeric;
meterarcindicator.addPanelMeter(panelmeter);
chartVu.addChartObject(meterarcindicator);
```

Needle Meter Indicator

RTMeterNeedleIndicator

ChartPlot

RTPlot

RTSingleValueIndicator

RTMeterIndicator

RTMeterNeedleIndicator

This **RTMeterNeedleIndicator** class displays the current **RTProcessVar** value as a needle. Subtypes of the **RTMeterNeedleIndicator** are simple needles, pie wedge shaped needles (the fat end of the pie wedge is at the radius center) and arrow needles.

RTMeterNeedleIndicator Constructor

```
public static newRTMeterNeedleIndicator(frame: RTMeterCoordinates, datasource: RTProcessVar):
RTMeterNeedleIndicator

public static newRTMeterNeedleIndicatorCoordsProcessVar(frame: RTMeterCoordinates, datasource:
RTProcessVar): RTMeterNeedleIndicator

public static newRTMeterNeedleIndicatorCoordsProcessVarAttrib(frame: RTMeterCoordinates,
datasource: RTProcessVar, needlelength: number, needleoverhang: number, needlewidth: number,
attrib: ChartAttribute): RTMeterNeedleIndicator
```

Parameters

frame

The **RTMeterCoordinates** object defining the meter coordinate system.

datasource

The process variable associated with the meter indicator.

needlelength

Specifies length of the needle in normalized plot coordinates.

needleoverhang

Specifies the overhang of the back end of the needle indicator specified in needle radius normalized coordinates.

needlewidth

The color attributes of the meter indicator.

attrib

The color attributes of the meter indicator.

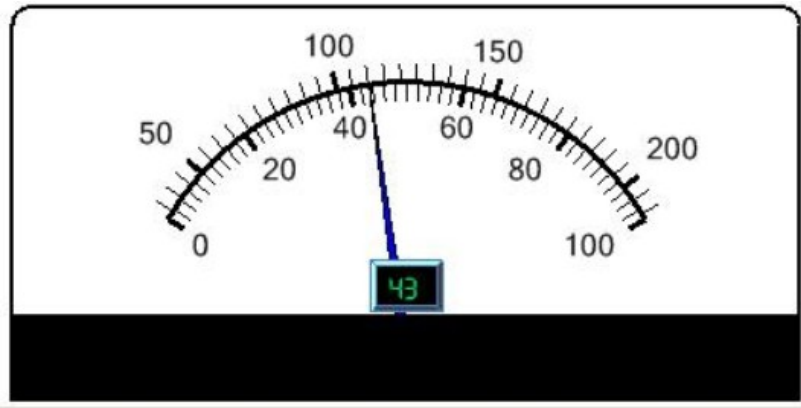
Selected Public Instance Properties

IndicatorSubtype (inherited from RTMeterIndicator)	Set/Get the meter indicator subtype. Use one of the meter needle indicator subtype constants: RT_METER_NEEDLE_SIMPLE_SUBTYPE , RT_METER_NEEDLE_PIEWEDGE_SUBTYPE, RT_METER_NEEDLE_ARROW_SUBTYPE,
NeedleBaseWidth	Set/Get the width of the base end of the needle for the RT_METER_NEEDLE_SIMPLE_SUBTYPE needle type, in device coordinates.
NeedleHeadLengthMultiplier	Set/Get the head length multiplier for the RT_METER_NEEDLE_ARROW_SUBTYPE needle type, in device coordinates.
NeedleHeadWidthMultiplier	Set/Get the head width multiplier for the RT_METER_NEEDLE_ARROW_SUBTYPE needle type, in device coordinates.
NeedleLength	Set/Get the length of the needle in normalized plot coordinates. length.
NeedleOverhang	Set/Get the overhang of the back end of the needle indicator specified as a fraction of the needle length.
PieWedgeDegrees	Set/Get Specifies the arc width of the needle for the RT_METER_NEEDLE_PIEWEDGE_SUBTYPE needle type, in degrees coordinates.
PivotColor	Set/Get the color of the needle pivot.
PivotDrawFlag	Set to true to draw the needle pivot.
PivotRadius	Set/Get in device coordinates the radius of the pivot point of the needle, analogous to the bearing or axle supporting the meter needle.

A complete listing of **RTMeterNeedleIndicator** properties is found in the `_qcertgraphs_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Examples for needle meter indicators

The examples below are program segments that give the important aspects of configuration a needle meter indicator for the image above it.



Extracted from the example program Meters, file NeedleMeters, method **InitializeMeter2**.

[TypeScript]

```
private InitializeMeter2() {
    if (!this.chartVu) return;
    let attrib1: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle: number = 150;
    let arcextent: number = 120;
    let startarcscale: number = 0;
    let endarcscale: number = 100;
    let arcdirection: boolean = false;
    let arcradius: number = 0.7;
    let centery: number = -0.75;
    let centerx: number = 0;
    let meterFont: QCRTGraphTS.ChartFont = this.font12;
    let meterframe1: QCRTGraphTS.RTMeterCoordinates =
    QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle, arcextent, startarcscale,
    endarcscale, arcdirection, centerx, centery, arcradius);
    meterframe1.setGraphBorderDiagonal(0.25, 0, 0.7, 0.45);
    let background: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(meterframe1,
    QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.WHITE);
    this.chartVu.addChartObject(background);
    let meterneedle: QCRTGraphTS.RTMeterNeedleIndicator =
    QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe1, this.processVar2);
    meterneedle.setChartObjAttributes(attrib1);
    meterneedle.NeedleLength = 0.7;
    meterneedle.ZOrder = 55;
    let panelmeterattrib: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter: QCRTGraphTS.RTNumericPanelMeter =
    QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(meterframe1, this.processVar1,
    panelmeterattrib);
    panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
    panelmeter.NumericTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
    panelmeter.NumericTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
    panelmeter.setLocation(0, 0.2);
    panelmeter.NumericTemplate.TextFont = this.font12Numeric;
    meterneedle.addPanelMeter(panelmeter);
    let panelmeterattrib3: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
    let panelmeter3: QCRTGraphTS.RTStringPanelMeter =
    QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(meterframe1,
    this.processVar1, panelmeterattrib3, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
    panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
    panelmeter3.setPositionReference(panelmeter);
}
```

189 Meter Indicators – Needle, Arc and Symbol

```
panelmeter3.PanelMeterNudge.Y = 0;
panelmeter3.StringTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
panelmeter3.StringTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_MAX;
panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
panelmeter3.StringTemplate.TextFont = this.font12Bold;
meterneedle.addPanelMeter(panelmeter3);
this.chartVu.addChartObject(meterneedle);
```

[JavaScript]

```
function InitializeMeter2() {
    if (!chartVu) return;
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle = 150;
    let arcextent = 120;
    let startarcscale = 0;
    let endarcscale = 100;
    let arcdirection = false;
    let arcradius = 0.7;
    let centery = -0.75;
    let centerx = 0;
    let meterFont = font12;
    let meterframe1 = QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle,
arcextent, startarcscale, endarcscale, arcdirection, centerx, centery, arcradius);
    meterframe1.setGraphBorderDiagonal(0.25, 0, 0.7, 0.45);
    let background = QCRTGraphTS.Background.newBackground(meterframe1,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.WHITE);
    chartVu.addChartObject(background);
    let meterneedle = QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe1,
processVar2);
    meterneedle.setChartObjAttributes(attrib1);
    meterneedle.NeedleLength = 0.7;
    meterneedle.ZOrder = 55;
    let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter = QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(meterframe1,
processVar1, panelmeterattrib);
    panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
    panelmeter.NumericTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
    panelmeter.NumericTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
    panelmeter.setLocation(0, 0.2);
    panelmeter.NumericTemplate.TextFont = font12Numeric;
    meterneedle.addPanelMeter(panelmeter);
    let panelmeter2tagattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
    let panelmeter3 =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(meterframe1,
processVar1, panelmeter2tagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
    panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
    panelmeter3.setPositionReference(panelmeter);
    panelmeter3.PanelMeterNudge.Y = 0;
    panelmeter3.StringTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
    panelmeter3.StringTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_MAX;
    panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
    panelmeter3.StringTemplate.TextFont = font12Bold;
    meterneedle.addPanelMeter(panelmeter3);
    chartVu.addChartObject(meterneedle);
```

Extracted from the example program Meters, file ArrowMeters.ts, method **InitializeMeter8**.



[TypeScript]

```
private InitializeMeter8() {
    if (!this.chartVu) return;
    let attrib1: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle: number = 90;
    let arcextent: number = 165;
    let startarcscale: number = -10;
    let endarcscale: number = 110;
    let arcdirection: boolean = false;
    let arcradius: number = 0.75;
    let centery: number = -0.1;
    let centerx: number = 0;
    let meterFont: QCRTGraphTS.ChartFont = this.font12Bold;
    let meterframe: QCRTGraphTS.RTMeterCoordinates =
    QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle, arcextent, startarcscale,
    endarcscale, arcdirection, centerx, centery, arcradius);
    meterframe.setGraphBorderDiagonal(0.5, 0.5, 0.8, 1);
    let background: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(meterframe,
    QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.WHITE);
    this.chartVu.addChartObject(background);
    let meterneedle: QCRTGraphTS.RTMeterNeedleIndicator =
    QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe, this.processVar1);
    meterneedle.IndicatorSubtype = QCRTGraphTS.ChartConstants.RT_METER_NEEDLE_ARROW_SUBTYPE;
    meterneedle.NeedleLength = 0.55;
    meterneedle.setChartObjAttributes(attrib1);
    let panelmeterattrib: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter: QCRTGraphTS.RTNumericPanelMeter =
    QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(meterframe, this.processVar1,
    panelmeterattrib);
    panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.RADIUS_TOP;
    panelmeter.NumericTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_MAX;
    panelmeter.NumericTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
    panelmeter.PanelMeterNudge.X = -16;

    panelmeter.NumericTemplate.TextFont = this.font12Numeric;
    meterneedle.addPanelMeter(panelmeter);
    this.chartVu.addChartObject(meterneedle);\
}
```

[JavaScript]

```
function InitializeMeter8() {
    if (!chartVu) return;
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
```

191 Meter Indicators – Needle, Arc and Symbol

```
let startarcangle = 90;
let arcextent = 165;
let startarcscale = -10;
let endarcscale = 110;
let arcdirection = false;
let arcradius = 0.75;
let centery = -0.1;
let centerx = 0;
let meterFont = font12Bold;
let meterframe = QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle,
arcextent, startarcscale, endarcscale, arcdirection, centerx, centery, arcradius);
meterframe.setGraphBorderDiagonal(0.5, 0.5, 0.8, 1);
let background = QCRTGraphTS.Background.newBackground(meterframe,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.WHITE);
chartVu.addChartObject(background);
let meterneedle =
QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe,processVar1);
meterneedle.IndicatorSubtype = QCRTGraphTS.ChartConstants.RT_METER_NEEDLE_ARROW_SUBTYPE;
meterneedle.NeedleLength = 0.55;
meterneedle.setChartObjAttributes(attrib1);
let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let panelmeter =
QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(meterframe,processVar1, panelmeterattrib);
panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.RADIUS_TOP;
panelmeter.NumericTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_MAX;
panelmeter.NumericTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
panelmeter.PanelMeterNudge.X = -16;
panelmeter.NumericTemplate.TextFont = font12Numeric;
meterneedle.addPanelMeter(panelmeter);
chartVu.addChartObject(meterneedle);
```

Symbol Meter Indicators

Class RTMeterSymbolIndicator

ChartPlot

RTPlot

RTSingleValueIndicator

RTMeterIndicator

RTMeterSymbolIndicator

This **RTMeterSymbolIndicator** class displays the current **RTProcessVar** value as a symbol moving around in the meter arc. Symbols include all of the scatter plot symbols: SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, BAR3D, and CIRCLE.

RTMeterSymbolIndicator constructor

```
public static newRTMeterSymbolIndicator(frame: RTMeterCoordinates, datasource: RTProcessVar):
RTMeterSymbolIndicator

public static newRTMeterSymbolIndicatorCoordsProcessVar(frame: RTMeterCoordinates, datasource:
RTProcessVar): RTMeterSymbolIndicator

public newRTMeterSymbolIndicatorCoordsProcessVarSymbol(frame: RTMeterCoordinates, datasource:
RTProcessVar, symbolnum: number, symbolsize: number, attrib: ChartAttribute):
RTMeterSymbolIndicator
```

Parameters

frame

The **RTMeterCoordinates** object defining the meter properties for the indicator.

datasource

The process variable associated with the indicator.

symbolnum

Specifies what symbol to use in the indicator. Use one of the scatter plot symbol constants: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, BAR3D, CIRCLE.

symbolsize

The size of the symbol in points.

attrib

The color attributes of the indicator.

Selected Public Instance Properties

[IndicatorSubtype](#) (inherited from **RTMeterIndicator**)

Set/Get the meter indicator subtype. Use one of the meter symbol indicator subtype constants:
RT_METER_SYMBOL_ARC_SUBTYPE,
RT_METER_SINGLE_SYMBOL_SUBTYPE.

[SegmentValueRoundMode](#)

Get/Set that the current process value is rounded up in calculating how many symbols to display in
RT_METER_SYMBOL_ARC_SUBTYPE,
RT_METER_SINGLE_SYMBOL_SUBTYPE
modes. Use one of the constants:
RT_FLOOR_VALUE, RT_CEILING_VALUE.

[SymbolNum](#)

Set/Get the symbol used as the indicator symbol.
Use one of the scatter plot symbol constants:
NOSYMBOL, SQUARE, TRIANGLE, DIAMOND,
CROSS, PLUS, STAR, LINE, HBAR, VBAR,
BAR3D, CIRCLE.

[SymbolPosPercent](#)

Set/Get the radial position of the symbol indicator.

[SymbolSize](#)

Set/Get the size of the symbol indicator in points.

[SymbolSpacing](#)

Get/Set the space, in degrees, between adjacent symbols.

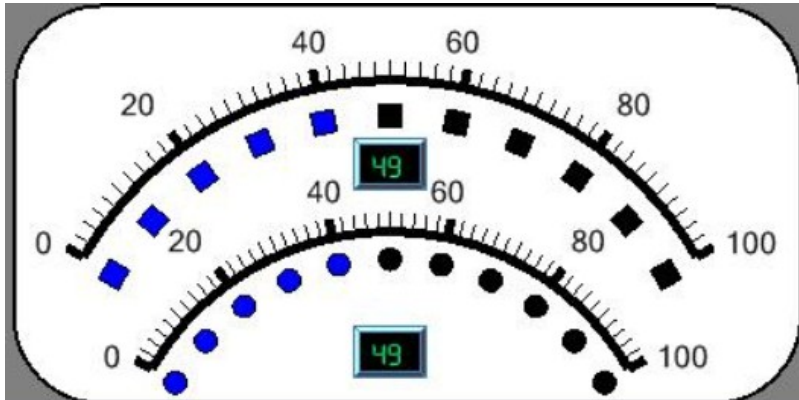
A complete listing of **RTMeterSymbolIndicator** properties is found in the `_qcrtrgraphs_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

In the single symbol indicator subtype (**RTMeterSymbolIndicator.IndicatorSubType** = `RT_METER_SINGLE_SYMBOL_SUBTYPE`), only the last symbol is “on”. The symbols up to but not including the final symbol are turned “off”,

Examples for symbol meter indicators

The examples below are program segments that give the important aspects of configuration a needle meter indicator for the image above it.

The top meter indicator, extracted from the example program Meters, file SymbolMeters, method **InitializeMeter2**.



[TypeScript]

```
private InitializeMeter2() {
    if (!this.chartVu) return;
    let attrib1: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle: number = 150;
    let arcextent: number = 120;
    let startarcscale: number = 0;
    let endarcscale: number = 100;
    let arcdirection: boolean = false;
    let arcradius: number = 0.9;
    let centery: number = -1.0;
    let centerx: number = 0;
    let meterFont: QCRTGraphTS.ChartFont = this.font12;
    let meterframe: QCRTGraphTS.RTMeterCoordinates =
    QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle, arcextent, startarcscale,
    endarcscale, arcdirection, centerx, centery, arcradius);
    meterframe.setGraphBorderDiagonal(0.25, 0, 0.7, 0.45);
    let background: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(meterframe,
    QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
    this.chartVu.addChartObject(background);

    let meterarcindicator: QCRTGraphTS.RTMeterSymbolIndicator =
    QCRTGraphTS.RTMeterSymbolIndicator.newRTMeterSymbolIndicator(meterframe, this.processVar2);
    meterarcindicator.setChartObjAttributes(attrib1);
    meterarcindicator.IndicatorSubtype = QCRTGraphTS.ChartConstants.RT_METER_SYMBOL_ARC_SUBTYPE;
    meterarcindicator.IndicatorBackgroundEnable = true;
    meterarcindicator.SymbolNum = QCRTGraphTS.ChartConstants.SQUARE;
    meterarcindicator.SymbolPosPercent = 0.9;
    meterarcindicator.SymbolSpacing = 10;

    let panelmeterattrib: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
```

```

let panelmeter: QCRTGraphTS.RTNumericPanelMeter =
QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(meterframe, this.processVar1,
panelmeterattrib);
panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
panelmeter.setLocation(0, 1.05);
panelmeter.NumericTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
panelmeter.NumericTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
panelmeter.NumericTemplate.TextFont = this.font12Numeric;
meterarcindicator.addPanelMeter(panelmeter);
this.chartVu.addChartObject(meterarcindicator);

```

[JavaScript]

```

function InitializeMeter2() {
    if (!chartVu) return;
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let startarcangle = 150;
    let arcextent = 120;
    let startarcscale = 0;
    let endarcscale = 100;
    let arcdirection = false;
    let arcradius = 0.9;
    let centery = -1.0;
    let centerx = 0;
    let meterFont = font12;
    let meterframe = QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle,
arcextent, startarcscale, endarcscale, arcdirection, centerx, centery, arcradius);
    meterframe.setGraphBorderDiagonal(0.25, 0, 0.7, 0.45);
    let background = QCRTGraphTS.Background.newBackground(meterframe,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
    chartVu.addChartObject(background);

    let meterarcindicator =
QCRTGraphTS.RTMeterSymbolIndicator.newRTMeterSymbolIndicator(meterframe, processVar2);
    meterarcindicator.setChartObjAttributes(attrib1);
    meterarcindicator.IndicatorSubtype = QCRTGraphTS.ChartConstants.RT_METER_SYMBOL_ARC_SUBTYPE;
    meterarcindicator.IndicatorBackgroundEnable = true;
    meterarcindicator.SymbolNum = QCRTGraphTS.ChartConstants.SQUARE;
    meterarcindicator.SymbolPosPercent = 0.9;
    meterarcindicator.SymbolSpacing = 10;
    let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter = QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(meterframe,
processVar1, panelmeterattrib);
    panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
    panelmeter.setLocation(0, 1.05);
    panelmeter.NumericTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
    panelmeter.NumericTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
    panelmeter.NumericTemplate.TextFont = font12Numeric;
    meterarcindicator.addPanelMeter(panelmeter);
    chartVu.addChartObject(meterarcindicator);
}

```

The bottom meter indicator, extracted from the example program Meters, file SymbolMeters.ts, method **InitializeMeter3**.

[TypeScript]

```

private InitializeMeter3() {
    if (!this.chartVu) return;

```


195 Meter Indicators – Needle, Arc and Symbol

```
let attrib1: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
let startarcangle: number = 150;
let arcextent: number = 120;
let startarcscale: number = 0;
let endarcscale: number = 100;
let arcdirection: boolean = false;
let arcradius: number = 0.7;
let centery: number = -1.35;
let centerx: number = 0;
let meterFont: QCRTGraphTS.ChartFont =this.font12;
let meterframe: QCRTGraphTS.RTMeterCoordinates =
QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle, arcextent, startarcscale,
endarcscale, arcdirection, centerx, centery, arcradius);
meterframe.setGraphBorderDiagonal(0.25, 0, 0.7, 0.45);
let background: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(meterframe,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
this.chartVu.addChartObject(background);

let meterarcindicator: QCRTGraphTS.RTMeterSymbolIndicator =
QCRTGraphTS.RTMeterSymbolIndicator.newRTMeterSymbolIndicator(meterframe, this.processVar2);
meterarcindicator.setChartObjAttributes(attrib1);
meterarcindicator.IndicatorSubtype = QCRTGraphTS.ChartConstants.RT_METER_SYMBOL_ARC_SUBTYPE;
meterarcindicator.IndicatorBackgroundEnable = true;
meterarcindicator.SymbolNum = QCRTGraphTS.ChartConstants.SQUARE;
meterarcindicator.SymbolPosPercent = 0.9;
meterarcindicator.SymbolSpacing = 10;

let panelmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let panelmeter: QCRTGraphTS.RTNumericPanelMeter =
QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(meterframe, this.processVar1,
panelmeterattrib);
panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
panelmeter.setLocation(0, 0.6);
panelmeter.NumericTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
panelmeter.NumericTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
panelmeter.NumericTemplate.TextFont =this.font12Numeric;
meterarcindicator.addPanelMeter(panelmeter);
this.chartVu.addChartObject(meterarcindicator);
.
```

[JavaScript]

```
function InitializeMeter3() {
if (!chartVu) return;
let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
let startarcangle = 150;
let arcextent = 120;
let startarcscale = 0;
let endarcscale = 100;
let arcdirection = false;
let arcradius = 0.7;
let centery = -1.35;
let centerx = 0;
let meterFont = font12;
let meterframe = QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle,
arcextent, startarcscale, endarcscale, arcdirection, centerx, centery, arcradius);
meterframe.setGraphBorderDiagonal(0.25, 0, 0.7, 0.45);
let background = QCRTGraphTS.Background.newBackground(meterframe,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
chartVu.addChartObject(background);

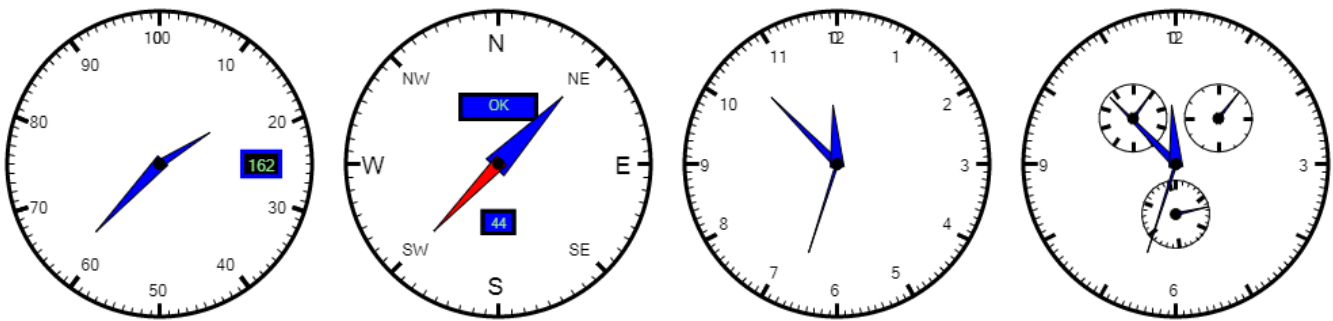
let meterarcindicator =
QCRTGraphTS.RTMeterSymbolIndicator.newRTMeterSymbolIndicator(meterframe, processVar2);
meterarcindicator.setChartObjAttributes(attrib1);
```

```
meterarcindicator.IndicatorSubtype = QCRTGraphTS.ChartConstants.RT_METER_SYMBOL_ARC_SUBTYPE;
meterarcindicator.IndicatorBackgroundEnable = true;
meterarcindicator.SymbolNum = QCRTGraphTS.ChartConstants.SQUARE;
meterarcindicator.SymbolPosPercent = 0.9;
meterarcindicator.SymbolSpacing = 10;
let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let panelmeter = QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(meterframe,
processVar1, panelmeterattrib);
panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
panelmeter.setLocation(0, 0.6);
panelmeter.NumericTemplate.XJust = QCRTGraphTS.ChartConstants.JUSTIFY_CENTER;
panelmeter.NumericTemplate.YJust = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
panelmeter.NumericTemplate.TextFont = font12Numeric;
meterarcindicator.addPanelMeter(panelmeter);
chartVu.addChartObject(meterarcindicator);
```

10. Dials and Clocks

RTComboProcessVar

RTMeterNeedleIndicator



Clocks and dials use the same meter components as described in the previous chapter:

RTMeterCoordinates, **RTMeterAxis**, **RTMeterAxisLabels**, **RTMeterStringAxisLabels**, **RTMeterIndicator**, **RTMeterArcIndicator**, **RTMeterNeedleIndicator**, and

RTMeterSymbolIndicator. For the purposes of this discussion, a dial and a clock is a meter that has an arc extent of 360 degrees, i.e. a full circle. Also, the current value of the dial or clock is characterized by a single value. In the case of a clock it is a date/time value and in the case of a dial it is a simple numeric value. While they are characterized by a single value, dials and clocks have multiple meter indicators representing that value using varying degrees of precision. Everyone is familiar with the hour, minute and second hands of clocks so we don't need to describe that further. As for a dial, the aircraft altimeter gauge is a perfect example. It has a small hand representing thousands of feet and a large hand representing hundreds. Clocks and dials must be able to take a single value, either time or some floating point value, and translate that information into two or more meter indicator values. In the case of a clock, the current time must be converted into values for the hour, minute and second hands. The class responsible for this conversion is the **RTComboProcessVar** class. It converts a single **RTProcessVar** value into multiple **RTProcessVar** objects, one representing the current value of each indicator of the clock or dial.

Converting Dial and Clock Data using RTComboProcessVar

Class RTComboProcessVar

RTProcessVar

RTComboProcessVar

The **RTComboProcessVar** class has an internal collection of **RTProcessVar** objects. The current value assigned to the **RTComboProcessVar** object is simultaneously converted to current values for each of the **RTProcessVar** objects in the collection. For each **RTProcessVar** object, the conversion is defined by a divisor and a modulo N value. Each **RTProcessVar** object will have unique combination of divisors and modulo N values as defining characteristics.

```
for (let i: number = 0; (i < this.processVarsList.length); i++) {
    processvalue = (<RTProcessVar>(this.processVarsList[i]));
    currentvalue = ChartSupport.intMod((pv / this.divisorList[i]), this.moduloList[i]);
    processvalue.setCurrentValue(timestamp, currentvalue);
}
```

where

comboProcessVar

The main **RTComboProcessVar** object that is updated by the application program

processVarList

The collection of **RTProcessVar** objects internal to the **RTComboProcessVar**. These items are updated automatically by the master **RTComboProcessVar** whenever an update is made to the master class.

Note that the divisor operation takes place first, followed by the modulo operation.

RTComboProcessVar constructors

```
public static newRTComboProcessVar(tagname: string, defaultattribute: ChartAttribute):
RTComboProcessVar

public static newRTComboProcessVarNameAttrib(tagname: string, defaultattribute: ChartAttribute):
RTComboProcessVar

public static newRTComboProcessVarDatasetAttrib(dataset: TimeSimpleDataset, defaultattribute:
ChartAttribute): RTComboProcessVar
```

Parameters

dataset

A dataset that will be used to store historical values.

defaultattribute

Specifies the default attribute for this process variable.

tagname

The string representing the tag name of the process variable.

Selected Public Instance Properties

Item	Get the RTProcessVar item at the specified index in the process variable list.

Selected Public Instance Methods

addProcessVar	Adds a new process variable to the process variable list.
resetProcessVarsList	Clears the process variable list.
setCurrentValue	Overloaded. Updates the current value and the dataset of the underlying RTProcessVar . It also updates the process variable list with the calculated process values.
setDivisorItem	Sets the divisor factor at the specified index.
setModuloItem	Sets the modulo factor at the specified index.

A complete listing of **RTComboProcessVar** properties is found in the `_qcrgraphts_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Examples for using RTComboProcessVar in a clock application

The example, extracted from the example program `AutoInstrumentPanel`, methods **InitializeGraph** and **InitializeClock**, show the important aspects of using an **RTComboProcessVar** object to supply data for the three meter needle indicators used as the hands of a clock.



[TypeScript]

```
clockdata: QCRTGraphTS.RTProcessVar[] = new Array<QCRTGraphTS.RTProcessVar>(3);
clock12Hour: QCRTGraphTS.RTComboProcessVar = new QCRTGraphTS.RTComboProcessVar();

.
.
.

this.clockdata[0] = QCRTGraphTS.RTProcessVar.newRTProcessVar("Seconds", defaultattrib);
this.clockdata[1] = QCRTGraphTS.RTProcessVar.newRTProcessVar("Minutes", defaultattrib);
this.clockdata[2] = QCRTGraphTS.RTProcessVar.newRTProcessVar("Hours", defaultattrib);
this.clock12Hour = QCRTGraphTS.RTComboProcessVar.newRTComboProcessVarNameAttrib("12-Hour Clock",
defaultattrib);
this.clock12Hour.addProcessVar(this.clockdata[0]);
// seconds
this.clock12Hour.addProcessVar(this.clockdata[1]);
// minutes
```

```

this.clock12Hour.addProcessVar(this.clockdata[2]);
// hours
// Clock/Meter coordinates is going to be scaled from 0-12,
// All values must be converted to this range
this.clock12Hour.setDivisorItem(0, 5);
// seconds/5 give seconds position on 0-12 scale
this.clock12Hour.setDivisorItem(1, (5 * 60));
// seconds/300 give minutes on 0-12 scale
this.clock12Hour.setDivisorItem(2, (60 * 60));
// seconds / 3600 give hours on 0-12 scale
this.clock12Hour.setModuloItem(0, 12);
// apply modulo 12 base
this.clock12Hour.setModuloItem(1, 12);
// apply modulo 12 base
this.clock12Hour.setModuloItem(2, 12);
.
.
.

private InitializeClock() {
    if (!this.chartVu) return;
    let startarcangle: number = 90;
    let arcextent: number = 360;
    let startarcscale: number = 0;
    let endarcscale: number = 12;
    let arcdirection: boolean = false;
    let arcradius: number = 0.5;
    let centery: number = 0;
    let centerx: number = 0;
    let meterFont: QCRTGraphTS.ChartFont = this.font12;
    let meterframe: QCRTGraphTS.RTMeterCoordinates =
QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle, arcextent, startarcscale,
endarcscale, arcdirection, centerx, centery, arcradius);
    meterframe.setGraphBorderDiagonal(0.8, 0.0, 0.99, 0.3);
    let frameattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    // Seconds
    let needleattrib1: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let meterneedle1: QCRTGraphTS.RTMeterNeedleIndicator =
QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe, this.clockdata[0]);
    meterneedle1.NeedleBaseWidth = 1;
    meterneedle1.setChartObjAttributes(needleattrib1);
    meterneedle1.NeedleLength = 0.5;
    this.chartVu.addChartObject(meterneedle1);
    // Minutes
    let needleattrib2: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let meterneedle2: QCRTGraphTS.RTMeterNeedleIndicator =
QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe, this.clockdata[1]);
    meterneedle2.NeedleBaseWidth = 3;
    meterneedle2.setChartObjAttributes(needleattrib2);
    meterneedle2.NeedleLength = 0.45;
    this.chartVu.addChartObject(meterneedle2);
    // Hours
    let needleattrib3: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let meterneedle3: QCRTGraphTS.RTMeterNeedleIndicator =
QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe, this.clockdata[2]);
    meterneedle3.NeedleBaseWidth = 5;
    meterneedle3.setChartObjAttributes(needleattrib3);
    meterneedle3.NeedleLength = 0.3;
    this.chartVu.addChartObject(meterneedle3);
    let meteraxisattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID);

```

201 Dials and Clocks

```
let meteraxis: QCRTGraphTS.RTMeterAxis = QCRTGraphTS.RTMeterAxis.newRTMeterAxis(meterframe,
meterneedle2);
meteraxis.setChartObjAttributes(meteraxisattrib);
meteraxis.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MIN);
meteraxis.LineWidth = 1;
meteraxis.LineColor = QCRTGraphTS.ChartColor.BLACK;
meteraxis.setAxisTickSpace(0.5);
meteraxis.setAxisMinorTicksPerMajor(6);
meteraxis.ShowAlarms = false;
meterneedle2.MeterAxis = meteraxis;

this.chartVu.addChartObject(meteraxis);
let meteraxislabels: QCRTGraphTS.RTMeterAxisLabels =
QCRTGraphTS.RTMeterAxisLabels.newRTMeterAxisLabels(meteraxis);
meteraxislabels.setTextFont(meterFont);
meteraxislabels.setAxisLabelsDir(meteraxis.getAxisTickDir());
// meteraxislabels.TextBoxMode = true;
meteraxislabels.OverlapLabelMode = QCRTGraphTS.ChartConstants.OVERLAP_LABEL_DRAW;
meteraxislabels.AxisLabelsEnds = QCRTGraphTS.ChartConstants.LABEL_MAX;
this.chartVu.addChartObject(meteraxislabels);
let meteraxis2: QCRTGraphTS.RTMeterAxis = QCRTGraphTS.RTMeterAxis.newRTMeterAxis(meterframe,
meterneedle2);
meteraxis2.setChartObjAttributes(meteraxisattrib);
meteraxis2.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MIN);
meteraxis2.LineWidth = 1;
meteraxis2.LineColor = QCRTGraphTS.ChartColor.BLACK;
meteraxis2.setAxisTickSpace(1);
meteraxis2.setAxisMinorTicksPerMajor(1);
meteraxis2.ShowAlarms = false;
this.chartVu.addChartObject(meteraxis2);
let attrib2: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GRAY, 5,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
let shaperect: QCRTGraphTS.ChartRectangle2D =
QCRTGraphTS.ChartRectangle2D.newChartRectangle2D(0.8, 0.025, 0.19, 0.25);
let genshape: QCRTGraphTS.RTGenShape = QCRTGraphTS.RTGenShape.newRTGenShapeRect(meterframe,
shaperect, 0, QCRTGraphTS.ChartConstants.RT_SHAPE_ELLIPSE,
QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
genshape.setChartObjAttributes(attrib2);
this.chartVu.addChartObject(genshape);
}
```

[JavaScript]

```
var clockdata = new Array(3);
var clock12Hour = new QCRTGraphTS.RTComboProcessVar();
.
.
.
clockdata[0] = QCRTGraphTS.RTProcessVar.newRTProcessVar("Seconds", defaultattrib);
clockdata[1] = QCRTGraphTS.RTProcessVar.newRTProcessVar("Minutes", defaultattrib);
clockdata[2] = QCRTGraphTS.RTProcessVar.newRTProcessVar("Hours", defaultattrib);
clock12Hour = QCRTGraphTS.RTComboProcessVar.newRTComboProcessVarNameAttrib("12-Hour Clock",
defaultattrib);
clock12Hour.addProcessVar(clockdata[0]);
// seconds
clock12Hour.addProcessVar(clockdata[1]);
// minutes
clock12Hour.addProcessVar(clockdata[2]);
// hours
// Clock/Meter coordinates is going to be scaled from 0-12,
// All values must be converted to this range
clock12Hour.setDivisorItem(0, 5);
// seconds/5 give seconds position on 0-12 scale
clock12Hour.setDivisorItem(1, (5 * 60));
// seconds/300 give minutes on 0-12 scale
clock12Hour.setDivisorItem(2, (60 * 60));
// seconds / 3600 give hours on 0-12 scale
```

```

clock12Hour.setModuloItem(0, 12);
// apply modulo 12 base
clock12Hour.setModuloItem(1, 12);
// apply modulo 12 base
clock12Hour.setModuloItem(2, 12);

async function InitializeClock() {
  if (!chartVu) return;
  let startarcangle = 90;
  let arcextent = 360;
  let startarcscale = 0;
  let endarcscale = 12;
  let arcdirection = false;
  let arcradius = 0.5;
  let centery = 0;
  let centerx = 0;
  let meterFont = font12;
  let meterframe = QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle,
arcextent, startarcscale, endarcscale, arcdirection, centerx, centery, arcradius);
  meterframe.setGraphBorderDiagonal(0.8, 0.0, 0.99, 0.3);
  let frameattrib = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK,
3, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
  // Seconds
  let needleattrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK,
1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
  let meterneedle1 = QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe,
clockdata[0]);
  meterneedle1.NeedleBaseWidth = 1;
  meterneedle1.setChartObjAttributes(needleattrib1);
  meterneedle1.NeedleLength = 0.5;
  chartVu.addChartObject(meterneedle1);
  // Minutes
  let needleattrib2 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK,
1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
  let meterneedle2 = QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe,
clockdata[1]);
  meterneedle2.NeedleBaseWidth = 3;
  meterneedle2.setChartObjAttributes(needleattrib2);
  meterneedle2.NeedleLength = 0.45;
  chartVu.addChartObject(meterneedle2);
  // Hours
  let needleattrib3 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK,
1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
  let meterneedle3 = QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe,
clockdata[2]);
  meterneedle3.NeedleBaseWidth = 5;
  meterneedle3.setChartObjAttributes(needleattrib3);
  meterneedle3.NeedleLength = 0.3;
  chartVu.addChartObject(meterneedle3);
  let meteraxisattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID);
  let meteraxis = QCRTGraphTS.RTMeterAxis.newRTMeterAxis(meterframe, meterneedle2);
  meteraxis.setChartObjAttributes(meteraxisattrib);
  meteraxis.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MIN);
  meteraxis.LineWidth = 1;
  meteraxis.LineColor = QCRTGraphTS.ChartColor.BLACK;
  meteraxis.setAxisTickSpace(0.5);
  meteraxis.setAxisMinorTicksPerMajor(6);
  meteraxis.ShowAlarms = false;
  meterneedle2.MeterAxis = meteraxis;
  chartVu.addChartObject(meteraxis);
  let meteraxislabels = QCRTGraphTS.RTMeterAxisLabels.newRTMeterAxisLabels(meteraxis);
  meteraxislabels.setTextFont(meterFont);
  meteraxislabels.setAxisLabelsDir(meteraxis.getAxisTickDir());
  // meteraxislabels.TextEditMode = true;
  meteraxislabels.OverlapLabelMode = QCRTGraphTS.ChartConstants.OVERLAP_LABEL_DRAW;
  meteraxislabels.AxisLabelsEnds = QCRTGraphTS.ChartConstants.LABEL_MAX;
  chartVu.addChartObject(meteraxislabels);
  let meteraxis2 = QCRTGraphTS.RTMeterAxis.newRTMeterAxis(meterframe, meterneedle2);

```


203 Dials and Clocks

```
meteraxis2.setChartObjAttributes(meteraxisattrib);
meteraxis2.setAxisTickDir(QCARTGraphTS.ChartConstants.AXIS_MIN);
meteraxis2.LineWidth = 1;
meteraxis2.LineColor = QCARTGraphTS.ChartColor.BLACK;
meteraxis2.setAxisTickSpace(1);
meteraxis2.setAxisMinorTicksPerMajor(1);
meteraxis2.ShowAlarms = false;
chartVu.addChartObject(meteraxis2);
let attrib2 = QCARTGraphTS.ChartAttribute.newChartAttribute(QCARTGraphTS.ChartColor.GRAY, 5,
QCARTGraphTS.ChartConstants.LS_SOLID, QCARTGraphTS.ChartColor.WHITE);
let shaperect = QCARTGraphTS.ChartRectangle2D.newChartRectangle2D(0.8, 0.025, 0.19, 0.25);
let genshape = QCARTGraphTS.RTGenShape.newRTGenShapeRect(meterframe, shaperect, 0,
QCARTGraphTS.ChartConstants.RT_SHAPE_ELLIPSE, QCARTGraphTS.ChartConstants.NORM_GRAPH_POS);
genshape.setChartObjAttributes(attrib2);
chartVu.addChartObject(genshape);
}
```

Examples for using RTComboProcessVar in an altimeter application

The example, extracted from the example program Meters, file DialsAndClocks, methods **InitializeGraph** and **InitializeDial1**, show the important aspects of using an **RTComboProcessVar** object to supply data for the two meter needle indicators used as the hands of a clock.



[TypeScript]

```
processVar1: QCARTGraphTS.RTProcessVar = new QCARTGraphTS.RTProcessVar();

dialComboProcessVar1: QCARTGraphTS.RTComboProcessVar = new QCARTGraphTS.RTComboProcessVar();

bigProcessVarArray: QCARTGraphTS.RTProcessVar[] = new Array(11);

this.dialComboProcessVar1 = QCARTGraphTS.RTComboProcessVar.newRTComboProcessVar("Altimeter",
this.processVar1.DefaultAttribute);
this.dialComboProcessVar1.addProcessVar(this.bigProcessVarArray[0]);
this.dialComboProcessVar1.addProcessVar(this.bigProcessVarArray[1]);
this.dialComboProcessVar1.setDivisorItem(0, 1);
this.dialComboProcessVar1.setDivisorItem(1, 10);
this.dialComboProcessVar1.setModuloItem(0, 100);
this.dialComboProcessVar1.setModuloItem(1, 100);

private InitializeDial1() {
    if (!this.chartVu) return;
    let startarcangle: number = 90;
    let arcextent: number = 360;
    let startarcscale: number = 0;
```

```

let endarcscale: number = 100;
let arcdirection: boolean = false;
let arcradius: number = 0.9;
let centery: number = 0;
let centerx: number = 0;
let meterFont: QCRTGraphTS.ChartFont = this.font12;
let meterframe: QCRTGraphTS.RTMeterCoordinates =
QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle, arcextent, startarcscale,
endarcscale, arcdirection, centerx, centery, arcradius);
meterframe.setGraphBorderDiagonal(0, 0, 0.25, 0.45);
let gbackground: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(meterframe,
QCRTGraphTS.ChartConstants.GRAPH_BACKGROUND, QCRTGraphTS.ChartColor.WHITE);
this.chartVu.addChartObject(gbackground);
let frameattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK,
3, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
let needleattrib1: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK,
1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
let meterneedle1: QCRTGraphTS.RTMeterNeedleIndicator =
QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe,
this.bigProcessVarArray[0]);
meterneedle1.NeedleBaseWidth = 5;
meterneedle1.setChartObjAttributes(needleattrib1);
meterneedle1.NeedleLength = 0.55;
this.chartVu.addChartObject(meterneedle1);
let needleattrib2: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK,
1, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
let meterneedle2: QCRTGraphTS.RTMeterNeedleIndicator =
QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe,
this.bigProcessVarArray[1]);
meterneedle2.NeedleBaseWidth = 3;
meterneedle2.setChartObjAttributes(needleattrib2);
meterneedle2.NeedleLength = 0.35;
this.chartVu.addChartObject(meterneedle2);
let meteraxisattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.BLACK,
1, QCRTGraphTS.ChartConstants.LS_SOLID);
let meteraxis: QCRTGraphTS.RTMeterAxis = QCRTGraphTS.RTMeterAxis.newRTMeterAxis(meterframe,
meterneedle2);
meteraxis.setChartObjAttributes(meteraxisattrib);
meteraxis.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MIN);
meteraxis.LineWidth = 1;
meteraxis.LineColor = QCRTGraphTS.ChartColor.BLACK;
meteraxis.setAxisTickSpace(1);
meteraxis.setAxisMinorTicksPerMajor(10);
meteraxis.ShowAlarms = false;
meterneedle2.MeterAxis = meteraxis;

this.chartVu.addChartObject(meteraxis);
let meteraxislabels: QCRTGraphTS.RTMeterAxisLabels =
QCRTGraphTS.RTMeterAxisLabels.newRTMeterAxisLabels(meteraxis);
meteraxislabels.setTextFont(meterFont);
meteraxislabels.setAxisLabelsDir(meteraxis.getAxisTickDir());
meteraxislabels.OverlapLabelMode = QCRTGraphTS.ChartConstants.OVERLAP_LABEL_DRAW;
meteraxislabels.AxisLabelsEnds = QCRTGraphTS.ChartConstants.LABEL_MAX;
this.chartVu.addChartObject(meteraxislabels);
let meteraxis2: QCRTGraphTS.RTMeterAxis = QCRTGraphTS.RTMeterAxis.newRTMeterAxis(meterframe,
meterneedle2);
meteraxis2.setChartObjAttributes(meteraxisattrib);
meteraxis2.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MIN);
meteraxis2.LineWidth = 3;
meteraxis2.LineColor = QCRTGraphTS.ChartColor.BLACK;
meteraxis2.setAxisTickSpace(10);
meteraxis2.setAxisMinorTicksPerMajor(1);
meteraxis2.ShowAlarms = false;
this.chartVu.addChartObject(meteraxis2);
let panelmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLUE,
3, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);

```

205 Dials and Clocks

```
let panelmeter: QCRTGraphTS.RTNumericPanelMeter =
QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(meterframe, this.dialComboProcessVar1,
panelmeterattrib);
panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
panelmeter.setLocation(0.6, 0);
panelmeter.NumericTemplate.setXJust(QCRTGraphTS.ChartConstants.JUSTIFY_CENTER);
panelmeter.NumericTemplate.setYJust(QCRTGraphTS.ChartConstants.JUSTIFY_CENTER);
panelmeter.TextColor = QCRTGraphTS.ChartColor.LIGHTGREEN;
panelmeter.NumericTemplate.TextFont = this.font12Numeric;
panelmeter.NumericTemplate.DecimalPos = 0;
this.chartVu.addChartObject(panelmeter);
}
```

[JavaScript]

```
var processVar1 = new QCRTGraphTS.RTProcessVar();
var dialComboProcessVar1 = new QCRTGraphTS.RTComboProcessVar();
// altimeter
var dialComboProcessVar2 = new QCRTGraphTS.RTComboProcessVar();
// 12-hour clock
var dialComboProcessVar3 = new QCRTGraphTS.RTComboProcessVar();
var bigProcessVarArray = new Array(11);
.
.
.
    dialComboProcessVar1 = QCRTGraphTS.RTComboProcessVar.newRTComboProcessVar("Altimeter",
processVar1.DefaultAttribute);
    dialComboProcessVar1.addProcessVar(bigProcessVarArray[0]);
    dialComboProcessVar1.addProcessVar(bigProcessVarArray[1]);
    dialComboProcessVar1.setDivisorItem(0, 1);
    dialComboProcessVar1.setDivisorItem(1, 10);
    dialComboProcessVar1.setModuloItem(0, 100);
    dialComboProcessVar1.setModuloItem(1, 100);
.
.
.
function InitializeDial1() {
    if (!chartVu) return;
    let startarcangle = 90;
    let arcextent = 360;
    let startarcscale = 0;
    let endarcscale = 100;
    let arcdirection = false;
    let arcradius = 0.9;
    let centery = 0;
    let centerx = 0;
    let meterFont = font12;
    let meterframe = QCRTGraphTS.RTMeterCoordinates.newRTMeterCoordinatesFull(startarcangle,
arcextent, startarcscale, endarcscale, arcdirection, centerx, centery, arcradius);
    meterframe.setGraphBorderDiagonal(0, 0, 0.25, 0.45);
    let gbackground = QCRTGraphTS.Background.newBackground(meterframe,
QCRTGraphTS.ChartConstants.GRAPH_BACKGROUND, QCRTGraphTS.ChartColor.WHITE);
    chartVu.addChartObject(gbackground);
    let frameattrib = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK,
3, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let needleattrib1 =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
    let meterneedle1 = QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe,
bigProcessVarArray[0]);
    meterneedle1.NeedleBaseWidth = 5;
    meterneedle1.setChartObjAttributes(needleattrib1);
    meterneedle1.NeedleLength = 0.55;
    chartVu.addChartObject(meterneedle1);
    let needleattrib2 =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLUE);
```

```

    let meterneedle2 = QCRTGraphTS.RTMeterNeedleIndicator.newRTMeterNeedleIndicator(meterframe,
bigProcessVarArray[1]);
    meterneedle2.NeedleBaseWidth = 3;
    meterneedle2.setChartObjAttributes(needleattrib2);
    meterneedle2.NeedleLength = 0.35;
    chartVu.addChartObject(meterneedle2);
    let meteraxisattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.BLACK, 1,
QCRTGraphTS.ChartConstants.LS_SOLID);
    let meteraxis = QCRTGraphTS.RTMeterAxis.newRTMeterAxis(meterframe, meterneedle2);
    meteraxis.setChartObjAttributes(meteraxisattrib);
    meteraxis.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MIN);
    meteraxis.LineWidth = 1;
    meteraxis.LineColor = QCRTGraphTS.ChartColor.BLACK;
    meteraxis.setAxisTickSpace(1);
    meteraxis.setAxisMinorTicksPerMajor(10);
    meteraxis.ShowAlarms = false;
    meterneedle2.MeterAxis = meteraxis;
    chartVu.addChartObject(meteraxis);
    let meteraxislabels = QCRTGraphTS.RTMeterAxisLabels.newRTMeterAxisLabels(meteraxis);
    meteraxislabels.setTextFont(meterFont);
    meteraxislabels.setAxisLabelsDir(meteraxis.getAxisTickDir());
    meteraxislabels.OverlapLabelMode = QCRTGraphTS.ChartConstants.OVERLAP_LABEL_DRAW;
    meteraxislabels.AxisLabelsEnds = QCRTGraphTS.ChartConstants.LABEL_MAX;
    chartVu.addChartObject(meteraxislabels);
    let meteraxis2 = QCRTGraphTS.RTMeterAxis.newRTMeterAxis(meterframe, meterneedle2);
    meteraxis2.setChartObjAttributes(meteraxisattrib);
    meteraxis2.setAxisTickDir(QCRTGraphTS.ChartConstants.AXIS_MIN);
    meteraxis2.LineWidth = 3;
    meteraxis2.LineColor = QCRTGraphTS.ChartColor.BLACK;
    meteraxis2.setAxisTickSpace(10);
    meteraxis2.setAxisMinorTicksPerMajor(1);
    meteraxis2.ShowAlarms = false;
    chartVu.addChartObject(meteraxis2);
    let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter = QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(meterframe,
dialComboProcessVar1, panelmeterattrib);
    panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
    panelmeter.setLocation(0.6, 0);
    panelmeter.NumericTemplate.setXJust(QCRTGraphTS.ChartConstants.JUSTIFY_CENTER);
    panelmeter.NumericTemplate.setYJust(QCRTGraphTS.ChartConstants.JUSTIFY_CENTER);
    panelmeter.TextColor = QCRTGraphTS.ChartColor.LIGHTGREEN;
    panelmeter.NumericTemplate.TextFont = font12Numeric;
    panelmeter.NumericTemplate.DecimalPos = 0;
    chartVu.addChartObject(panelmeter);
}

```

11. Single and Multiple Channel Annunciators

RTAnnunciator
RTMultiValueAnnunciator



An annunciator is used to display the current values and alarm states of real-time data. Each channel of data corresponds to a rectangular cell where each cell can contain the tag name, units, current value, and alarm status message. Any of these items may be excluded. If a channel is in alarm, the background of the corresponding cell changes its color, giving a strong visual indication that an alarm has occurred.

Single Channel Annunciator

Class RTAnnunciator

ChartPlot

RTPlot

RTMultiValueIndicator

RTAnnunciator

An **RTAnnunciator** is used to display the current values and alarm states of a single channel real-time data.

RTAnnunciator constructor

```
public static newRTAnnunciator(transform: PhysicalCoordinates, datasource: RTPProcessVar,
annunpos: ChartRectangle2D, attrib: ChartAttribute)

public static newRTAnnunciatorCoords(transform: PhysicalCoordinates): RTAnnunciator

public static newRTAnnunciatorCoordsProcessVar(transform: PhysicalCoordinates, datasource:
RTPProcessVar, annunpos: ChartRectangle2D, attrib: ChartAttribute)
```

Parameters

transform

The coordinate system for the new **RTAnnunciator** object.

datasource

The process variable associated with the annunciator.

annunpos

The position and size of the annunciator.

attrib

The color attributes of the annunciator.

The annunciator resides in coordinate system scaled for physical coordinates of (0.0,0.0) – (1.0, 1.0). The annunciator rectangle size and position is defined using the **RTAnnunciator.AnnunciatorRect** property. The default annunciator consists of a simple rectangle that changes color in response to the alarm state of the **RTPProcessVar** object attached to the annunciator. The annunciator can be customized with tag names, numeric readouts and alarm messages by adding **RTPanelMeter** objects to the **RTAnnunciator** object. See the examples below.

Public Instance Properties

AnnunciatorRect	Get/Set the annunciator rectangle.

A complete listing of **RTAnnunciator** properties is found in the `_qcrgraphTS_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example for single channel annunciator

The example below, extracted from the Annunciators example, file `Annunciators`, method **InitializeAnnunciator1**, creates a single channel annunciator with a tag name, numeric readout and alarm.



[TypeScript]

```
private InitializeAnnunciator1() {
    if (!this.chartVu) return;
    let pTransform1: QCRTGraphTS.CartesianCoordinates =
        QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
    pTransform1.setGraphBorderDiagonal(0.01, .05, 0.15, 0.35);
    let background: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(pTransform1,
        QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
    this.chartVu.addChartObject(background);
    let annunrect: QCRTGraphTS.ChartRectangle2D =
        QCRTGraphTS.ChartRectangle2D.newChartRectangle2D(0.05, 0.05, 0.9, 0.9);
    let attrib1: QCRTGraphTS.ChartAttribute =
        QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.DARKGRAY, 5,
        QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.DARKGRAY);
    let annunciator: QCRTGraphTS.RTAnnunciator =
        QCRTGraphTS.RTAnnunciator.newRTAnnunciator(pTransform1, this.processVar1, annunrect, attrib1);
    let panelmeterattrib: QCRTGraphTS.ChartAttribute =
        QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
        QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter: QCRTGraphTS.RTNumericPanelMeter =
        QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(pTransform1, this.processVar2,
        panelmeterattrib);
    panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.PLOTAREA_CENTER;
    panelmeter.NumericTemplate.TextFont = this.font18Numeric;
    panelmeter.NumericTemplate.PostfixString = String.fromCharCode(176) + "F";
    annunciator.addPanelMeter(panelmeter);
    let panelmeter2: QCRTGraphTS.RTAlarmPanelMeter =
        QCRTGraphTS.RTAlarmPanelMeter.newRTAlarmPanelMeter(pTransform1, this.processVar1,
        panelmeterattrib);
    panelmeter2.PanelMeterPosition = QCRTGraphTS.ChartConstants.INSIDE_BARBASE;
    panelmeter2.AlarmTemplate.TextFont = this.font12;
    panelmeter2.setPositionReference(panelmeter);
    annunciator.addPanelMeter(panelmeter2);
    let panelmetertagattrib: QCRTGraphTS.ChartAttribute =
        QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
        QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
    let panelmeter3: QCRTGraphTS.RTStringPanelMeter =
        QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(pTransform1,
        this.processVar1, panelmetertagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
    panelmeter3.setPositionReference(panelmeter);
    panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.INSIDE_BAR;
    panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
    annunciator.addPanelMeter(panelmeter3);
}
```

[JavaScript]

```

function InitializeAnnunciator1() {
    if (!chartVu) return;
    let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
    pTransform1.setGraphBorderDiagonal(0.01, .05, 0.15, 0.35);
    let background = QCRTGraphTS.Background.newBackground(pTransform1,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
    chartVu.addChartObject(background);
    let annunrect = QCRTGraphTS.ChartRectangle2D.newChartRectangle2D(0.05, 0.05, 0.9, 0.9);
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.DARKGRAY, 5,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.DARKGRAY);
    let annunciator = QCRTGraphTS.RTAnnunciator.newRTAnnunciator(pTransform1, processVar1,
annunrect, attrib1);
    let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter = QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(pTransform1,
processVar2, panelmeterattrib);
    panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.PLOTAREA_CENTER;
    panelmeter.NumericTemplate.TextFont = font18Numeric;
    panelmeter.NumericTemplate.PostfixString = String.fromCharCode(176) + "F";
    annunciator.addPanelMeter(panelmeter);
    let panelmeter2 = QCRTGraphTS.RTAlarmPanelMeter.newRTAlarmPanelMeter(pTransform1, processVar1,
panelmeterattrib);
    panelmeter2.PanelMeterPosition = QCRTGraphTS.ChartConstants.INSIDE_BARBASE;
    panelmeter2.AlarmTemplate.TextFont = font12;
    panelmeter2.setPositionReference(panelmeter);
    annunciator.addPanelMeter(panelmeter2);
    let panelmetertagattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
    let panelmeter3 =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(pTransform1,
processVar1, panelmetertagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
    panelmeter3.setPositionReference(panelmeter);
    panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.INSIDE_BAR;
    panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
    annunciator.addPanelMeter(panelmeter3);
    chartVu.addChartObject(annunciator);
}

```

Multi-Channel Annunciators**Class RTMultiValueAnnunciator****ChartPlot****RTPlot****RTMultiValueIndicator****RTMultiValueAnnunciator**

An **RTMultiValueAnnunciator** is used to display the current values and alarm states of a collection of **RTProcessVar** objects. It consists of a rectangular grid with individual channels represented by the rows and columns in of the grid. Each grid cell can contain the tag name, units, current value, and alarm status message for a single **RTProcessVar** object. Any of these items may be excluded. If a channel is in alarm, the background of the corresponding cell changes its color, giving a strong visual indication that an alarm has occurred.

RTMultiValueAnnunciator constructors

```

public static newRTMultiValueAnnunciatorCoords(transform: PhysicalCoordinates):
RTMultiValueAnnunciator

public static newRTMultiValueAnnunciator(transform: PhysicalCoordinates, datasource:
RTProcessVar[], attrib: ChartAttribute): RTMultiValueAnnunciator

public static newRTMultiValueAnnunciatorCoordsProcessVarAttrib(transform: PhysicalCoordinates,
datasource: RTProcessVar[], attrib: ChartAttribute): RTMultiValueAnnunciator

public static newRTMultiValueAnnunciatorCoordsProcessVarsAttribs(transform: PhysicalCoordinates,
datasource: RTProcessVar[], attribs: ChartAttribute[]): RTMultiValueAnnunciator

public static newRTMultiValueAnnunciatorCoordsProcessVarsRCAttribs(transform:
PhysicalCoordinates, datasource: RTProcessVar[], numcols: number, numrows: number, attribs:
ChartAttribute[]) : RTMultiValueAnnunciator

public static newRTMultiValueAnnunciatorCoordsMultiProcessVarsRCAttribs(transform:
PhysicalCoordinates, datasource: RTMultiProcessVar, numcols: number, numrows: number, attribs:
ChartAttribute[]): RTMultiValueAnnunciator

```

Parameters*transform*The coordinate system for the new **RTMultiValueAnnunciator** object.*datasource*An array of **RTProcessVar** objects, one for each annunciator cell.*numcols*

The number of columns in the annunciator display.

numrows

The number of rows in the annunciator display.

attribs

An array of the color attributes one for each annunciator cell.

Public Instance Properties[CellColumnMargin](#)

Get/Set the extra space between columns of the annunciator, specified in normalized NORM_PLOT_POS coordinates.

[CellRowMargin](#)

Get/Set the extra space between rows of the annunciator, specified in normalized NORM_PLOT_POS coordinates.

[NumberColumns](#)

Get the number of rows in the annunciator.

[NumberRows](#)

Get the number of rows in the annunciator.

A complete listing of **RTMultiValueAnnunciator** properties is found in the `_qcrgraphpts_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

The **CellColumnMargin** and the **CellRowMargin** values represent the total amount of spacing used for the cell column and cell row margins respectively. A value of 0.2 implies that 20% of the row or column space will be used as margin, and 80% will be used for the annunciator cells. The 20% margin value is

divided up between the cells in the row or column. If the multi-channel annunciator has 4 annunciator cells in a row, there are 5 border areas between the cells (3 at the interior of the annunciator cell grid and 2 on either end). The total margin of 20% is therefore divided 5 times, resulting in a 4% margin between the column of each grid cell.

Example for a simple multi-channel annunciator

The example below, extracted from the `AutoInstrumentPanel` example, method **InitializeAnnunciator**, creates a multi-channel annunciator that shows only the tag name of the associated **RTProcessVar** object.



[TypeScript]

```
private InitializeAnnunciator() {
    if (!this.chartVu) return;
    let pTransform1: QCRTGraphTS.CartesianCoordinates =
QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
    pTransform1.setGraphBorderDiagonal(0.05, 0.1, 0.725, 0.175);
    let attrib: QCRTGraphTS.ChartAttribute;
    let attribArray: QCRTGraphTS.ChartAttribute[] = new Array(this.annunciator1.length);
    for (let i: number = 0; (i < this.annunciator1.length); i++) {
        attrib = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.DARKRED, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.DARKRED);
        attribArray[i] = attrib;
    }

    let numRows: number = 1;
    let numcols: number = 8;
    let annunciator: QCRTGraphTS.RTMultiValueAnnunciator =
QCRTGraphTS.RTMultiValueAnnunciator.newRTMultiValueAnnunciatorCoordsProcessVarsRCAttribs(pTransform1, this.annunciator1, numcols, numRows, attribArray);
    annunciator.CellColumnMargin = 0.05;
    annunciator.CellRowMargin = 0.1;
    let panelmetertagattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
    let panelmeter: QCRTGraphTS.RTStringPanelMeter =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(pTransform1,
this.annunciator1[0], panelmetertagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
    panelmeter.StringTemplate.TextFont = this.font12Bold;
    panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.INSIDE_BAR;
    panelmeter.Frame3DEnable = false;
    panelmeter.TextColor = QCRTGraphTS.ChartColor.WHITE;
    panelmeter.StringTemplate.TextBgMode = false;
    panelmeter.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_INDICATOR_COLOR_NO_ALARM_CHANGE;
    annunciator.addPanelMeter(panelmeter);
    this.chartVu.addChartObject(annunciator);
}
```

[JavaScript]

```
function InitializeAnnunciator() {
    if (!chartVu) return;
    let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
    pTransform1.setGraphBorderDiagonal(0.05, 0.1, 0.725, 0.175);
    let attrib;
    let attribArray = new Array(annunciator1.length);
```

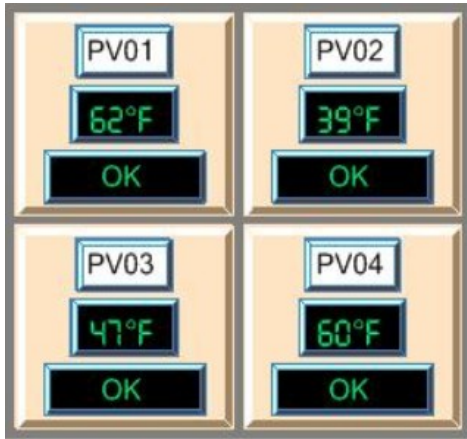
```

for (let i = 0; (i < annunciator1.length); i++) {
    attrib = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.DARKRED, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.DARKRED);
    attribArray[i] = attrib;
}
let numRows = 1;
let numcols = 8;
let annunciator =
QCRTGraphTS.RTMultiValueAnnunciator.newRTMultiValueAnnunciatorCoordsProcessVarsRCAttribs(pTransfo
rml, annunciator1, numcols, numRows, attribArray);
    annunciator.CellColumnMargin = 0.05;
    annunciator.CellRowMargin = 0.1;
    let panelmetertagattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
    let panelmeter =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(pTransform1,
annunciator1[0], panelmetertagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
    panelmeter.StringTemplate.TextFont = font12Bold;
    panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.INSIDE_BAR;
    panelmeter.Frame3DEnable = false;
    panelmeter.TextColor = QCRTGraphTS.ChartColor.WHITE;
    panelmeter.StringTemplate.TextBgMode = false;
    panelmeter.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_INDICATOR_COLOR_NO_ALARM_CHANGE;
    annunciator.addPanelMeter(panelmeter);
    chartVu.addChartObject(annunciator);
}

```

Example for a simple multi-channel annunciator

The example below, extracted from the Annunciators example, file `Annunciators`, method **InitializeAnnunciator2**, creates a multi-channel annunciator that shows the tag name, current value and alarm state of the associated **RTProcessVar** object.



[TypeScript]

```

private InitializeAnnunciator2() {
    let processVarArray: QCRTGraphTS.RTProcessVar[] = [
        this.processVar1,
        this.processVar2,
        this.processVar3,
        this.processVar4];
    if (!this.chartVu) return;
}

```

```

let pTransform1: QCRTGraphTS.CartesianCoordinates =
QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
pTransform1.setGraphBorderDiagonal(0.175, .05, 0.45, 0.475);
let background: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(pTransform1,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
this.chartVu.addChartObject(background);
let attrib1: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BISQUE, 5,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BISQUE);
let attrib2: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BISQUE, 5,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BISQUE);
let attrib3: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BISQUE, 5,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BISQUE);
let attrib4: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BISQUE, 5,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BISQUE);
let attribArray: QCRTGraphTS.ChartAttribute[] = [
    attrib1,
    attrib2,
    attrib3,
    attrib4];
let numRows: number = 2;
let numcols: number = 2;
let annunciator: QCRTGraphTS.RTMultiValueAnnunciator =
QCRTGraphTS.RTMultiValueAnnunciator.newRTMultiValueAnnunciatorCoordsProcessVarsRCAttribs(pTransform1, processVarArray, numcols, numRows, attribArray);
let panelmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let panelmeter: QCRTGraphTS.RTNumericPanelMeter =
QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(pTransform1, this.processVar2,
panelmeterattrib);
panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.CENTERED_BAR;
panelmeter.NumericTemplate.TextFont = this.font14Numeric;
panelmeter.NumericTemplate.PostfixString = String.fromCharCode(176) + "F";
annunciator.addPanelMeter(panelmeter);
let panelmeter2: QCRTGraphTS.RTAlarmPanelMeter =
QCRTGraphTS.RTAlarmPanelMeter.newRTAlarmPanelMeter(pTransform1, this.processVar1,
panelmeterattrib);
panelmeter2.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
panelmeter2.AlarmTemplate.TextFont = this.font12;
panelmeter2.setPositionReference(panelmeter);
annunciator.addPanelMeter(panelmeter2);
let panelmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
let panelmeter3: QCRTGraphTS.RTStringPanelMeter =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(pTransform1,
this.processVar1, panelmeterattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
panelmeter3.setPositionReference(panelmeter);
panelmeter3.StringTemplate.TextFont = this.font12;
panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.ABOVE_REFERENCED_TEXT;
panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
annunciator.addPanelMeter(panelmeter3);
this.chartVu.addChartObject(annunciator);
}

```

[JavaScript]

```

function InitializeAnnunciator2() {
    let processVarArray = [
        processVar1,
        processVar2,
        processVar3,

```

```

        processVar4
    ];
    if (!chartVu) return;
    let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
    pTransform1.setGraphBorderDiagonal(0.175, .05, 0.45, 0.475);
    let background = QCRTGraphTS.Background.newBackground(pTransform1,
    QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
    chartVu.addChartObject(background);
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BISQUE, 5,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BISQUE);
    let attrib2 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BISQUE, 5,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BISQUE);
    let attrib3 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BISQUE, 5,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BISQUE);
    let attrib4 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BISQUE, 5,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BISQUE);
    let attribArray = [
        attrib1,
        attrib2,
        attrib3,
        attrib4
    ];
    let numRows = 2;
    let numcols = 2;
    let annunciator =
    QCRTGraphTS.RTMultiValueAnnunciator.newRTMultiValueAnnunciatorCoordsProcessVarsRCAttribs(pTransform1,
    processVarArray, numcols, numRows, attribArray);
    let panelmeterattrib =
    QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter = QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(pTransform1,
    processVar2, panelmeterattrib);
    panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.CENTERED_BAR;
    panelmeter.NumericTemplate.TextFont = font14Numeric;
    panelmeter.NumericTemplate.PostfixString = String.fromCharCode(176) + "F";
    annunciator.addPanelMeter(panelmeter);
    let panelmeter2 = QCRTGraphTS.RTAlarmPanelMeter.newRTAlarmPanelMeter(pTransform1, processVar1,
    panelmeterattrib);
    panelmeter2.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
    panelmeter2.AlarmTemplate.TextFont = font12;
    panelmeter2.setPositionReference(panelmeter);
    annunciator.addPanelMeter(panelmeter2);
    let panelmetertagattrib =
    QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
    let panelmeter3 =
    QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(pTransform1,
    processVar1, panelmetertagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
    panelmeter3.setPositionReference(panelmeter);
    panelmeter3.StringTemplate.TextFont = font12;
    panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.ABOVE_REFERENCED_TEXT;
    panelmeter3.TextColor = QCRTGraphTS.ChartColor.BLACK;
    annunciator.addPanelMeter(panelmeter3);
    chartVu.addChartObject(annunciator);
}

```

Example for a large multi-channel annunciator

The example below, extracted from the Annunciators, file Annunciators, method **InitializeAnnunciator4** example, creates a multi-channel annunciator that shows the tag name and current value of the associated **RTProcessVar** object. The alarm state is implicit in the annunciator background color. See the example program for the code listing.

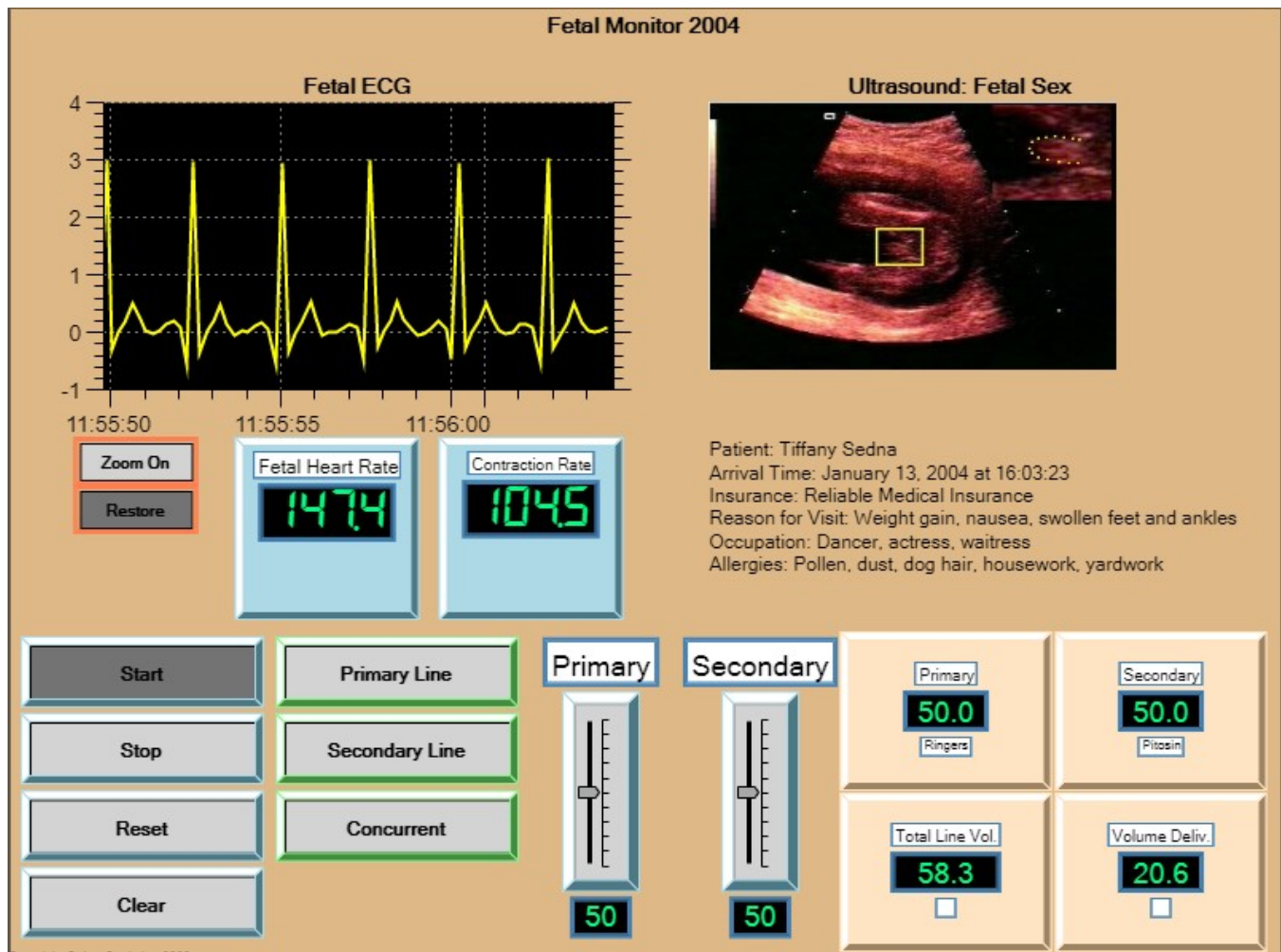
0°F PV0	7°F PV1	12°F PV2	3°F PV3	20°F PV4	61°F PV5	47°F PV6	80°F PV7	0°F PV8	3°F PV9	1°F PV10	11°F PV11	36°F PV12	14°F PV13
65°F PV14	90°F PV15	0°F PV16	10°F PV17	7°F PV18	17°F PV19	23°F PV20	17°F PV21	56°F PV22	60°F PV23	0°F PV24	1°F PV25	12°F PV26	33°F PV27
48°F PV28	3°F PV29	10°F PV30	58°F PV31	0°F PV32	12°F PV33	25°F PV34	19°F PV35	9°F PV36	16°F PV37	53°F PV38	88°F PV39	0°F PV40	2°F PV41
10°F PV42	16°F PV43	27°F PV44	30°F PV45	44°F PV46	48°F PV47	0°F PV48	3°F PV49	15°F PV50	6°F PV51	22°F PV52	24°F PV53	7°F PV54	32°F PV55

12. The Scroll Frame and Single Channel Scrolling Plots

RTScrollFrame

RTVerticalScrollFrame

RTSimpleSingleValuePlot



Scrolling graphs are built using three main classes. The first is the **RTScrollFrame** class that manages the constant rescaling of the coordinate system of the scrolling graph. The second and third are **RTSimpleSingleValuePlot** and **RTGroupMultiValuePlot** classes that encapsulate the actual line plot, bar plot, scatter plot or group plot that is plotted in the scrolling graph. The **RTScrollFrame** class and the **RTSimpleSingleValuePlot** classes are described in this chapter and the **RTGroupMultiValuePlot** class is described in the next.

The original **RTScrollFrame** manages scrolling of numeric, time/date and elapsed time coordinate systems in the horizontal direction. The **RTVerticalScrollFrame**, manages scrolling in the vertical direction.

Scroll Frame

Class RTScrollFrame

ChartPlot

RTPlot

RTMultiValueIndicator

RTScrollFrame

The scrolling algorithm used in this software is different that in earlier Quinn-Curtis real-time graphics products. Scrolling plots are no longer updated incrementally whenever the underlying data is updated. Instead, the underlying **RTProcessVar** data objects are updated as fast as you want. Scrolling graphs (all graphs for that matter) are only updated with the **ChartView.updateDraw()** method is called. What makes scrolling graphs appear to scroll is the scroll frame (**RTScrollFrame**). When a scroll frame is updated as a result of the **ChartView.updateDraw()** event, it analyzes the **RTSimpleSingleValuePlot** and **RTGroupMultiValuePlot** objects that have been attached to it and creates a coordinate system that matches the current and historical data associated with the plot objects. The plot objects in the scroll frame are drawn into this coordinate system. As data progresses forward in time the coordinate system is constantly being rescaled to include the most recent time values as part of the x-coordinate system. You can control whether or not the starting point of the scroll frame coordinate system remains fixed, whether it advances in sync with the constantly changing end of the scroll frame. Other options allow the y-scale to be constantly rescaled to reflect the current dynamic range of the y-values in the scroll frame. The long term goal is that as computers get faster, and JavaScript/TypeScript more efficient, you will never need to update the display faster than 30-60 times a second, since this will result smooth scrolling even if the underlying data is updated 10,000 times a second.

RTScrollFrame constructors

```
public static newRTScrollFrame(component: ChartView, processvar: RTProcessVar, initialscale: PhysicalCoordinates, scrollmode: number): RTScrollFrame
```

```
public static newRTScrollFrameChartViewProcessVarCoordsScrollMode(component: ChartView, processvar: RTProcessVar, initialscale: PhysicalCoordinates, scrollmode: number): RTScrollFrame
```

```
public static newRTScrollFrameChartViewProcessVarCoordsScrollModeXAutoScrollY(component: ChartView, processvar: RTProcessVar, initialscale: PhysicalCoordinates, scrollxmode: number, autoscaleymode: number): RTScrollFrame
```

```
public static newRTScrollFrameChartViewCoordsScrollMode(component: ChartView, initialscale: PhysicalCoordinates, scrollmode: number): RTScrollFrame
```

```
public static newRTScrollFrameChartViewCoordsScrollModeXAutoScrollY(component: ChartView, initialscale: PhysicalCoordinates, scrollxmode: number, autoscaleymode: number): RTScrollFrame
```


Parameters*component*

This **ChartView** component the scroll frame is placed in.

processvar

The source process variable.

initialscale

A coordinate system that serves as the initial scale for the scroll frame.

scrollxmode

Specifies x-axis auto-scale mode of the scroll frame. Use one of the x-axis scroll frame constants:

RT_NO_AUTOSCALE_X - no auto-scale for the x-axis, use in non-scrolling graphs.

RT_AUTOSCALE_X_CURRENT_SCALE - auto-scale based on current scale, use in non-scrolling graphs.

RT_AUTOSCALE_X_MIN - autoscale x-axis minimum only, use in non-scrolling graphs.

RT_AUTOSCALE_X_MAX - autoscale x-axis maximum only, use in non-scrolling graphs.

RT_AUTOSCALE_X_MINMAX - autoscale x-axis minimum and maximum, use in non-scrolling graphs.

RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL - autoscale the x-axis for a fixed range, with moving maximum and minimum values, use in scrolling graphs.

RT_MAXEXTENT_FIXEDSTART_AUTOSCROLL - autoscale the x-axis with the start of the x-axis fixed, and the end of the x-axis moving, use in scrolling graphs.

RT_FIXEDNUMPOINT_AUTOSCROLL.- autoscale the x-axis for a fixed number of points, with moving maximum and minimum values, use in scrolling graphs.

autoscaleymode

Specifies y-axis auto-scale mode of the scroll frame. Use one of the y-axis scroll frame constants:

RT_NO_AUTOSCALE_Y - no auto-scale for the y-axis

RT_AUTOSCALE_Y_MIN - autoscale y-axis minimum only

RT_AUTOSCALE_Y_MAX - autoscale y-axis maximum only

RT_AUTOSCALE_Y_MINMAX. - - autoscale y-axis minimum and maximum

Selected Public Instance Properties[AutoScaleRoundXMode](#)

Get/Set the auto-scale round mode for the x-coordinate.

Use one of the AUTOAXES round mode constants:

AUTOAXES_EXACT, AUTOAXES_NEAR,
AUTOAXES_FAR.

[AutoScaleRoundYMode](#)

Get/Set the auto-scale mode for the y-coordinate. Use

one of the AUTOAXES round mode constants:

AUTOAXES_EXACT, AUTOAXES_NEAR,
AUTOAXES_FAR.

[MaxDisplayHistory](#)

Get/Set the maximum number of points displayed.

[MinSamplesForAutoScale](#)

Get/Set the minimum number of samples that need to be in the dataset before an auto-scale operation is carried

ScrollRescaleMargin

out. This prevents the first datapoints from generating an arbitrarily small range.

Get/Set the scroll rescale margin. When the limits of the scale needs to be increased, the ScrollRescaleMargin * (current range of the x-axis) is added to the upper and lower limits of the current scale.

ScrollScaleModeX

Get/Set the scrolling mode for the x-coordinate. Use one of the x-axis scroll frame constants:

RT_NO_AUTOSCALE_X,
RT_AUTOSCALE_X_CURRENT_SCALE,
RT_AUTOSCALE_X_MIN,
RT_AUTOSCALE_X_MAX,
RT_AUTOSCALE_X_MINMAX,
RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL,
RT_MAXEXTENT_FIXEDSTART_AUTOSCROLL,
RT_FIXEDNUMPOINT_AUTOSCROLL

ScrollScaleModeY

Get/Set the scrolling mode for the y-coordinate. Use one of the y-axis scroll frame constants:

RT_NO_AUTOSCALE_Y,
RT_AUTOSCALE_Y_MIN,
RT_AUTOSCALE_Y_MAX,
RT_AUTOSCALE_Y_MINMAX

TimeStampMode

Get/Set the time stamp mode for the time values in the process variables. Use one of the time stamp mode constants:

RT_NOT_MONOTONIC_X_MODE - not monotonic means that the x-values do not have to increase with increasing time. A real-time xy plot that plots x-values against y-values might have this characteristic.
RT_MONOTONIC_X_MODE – The default value. Monotonic means that the x-values always increase with increasing time. If the scroll frame routines know that the x-values will never “backtrack” it speeds up the search algorithm for minimum and maximum x-values to use in auto-scaling the x-axis.

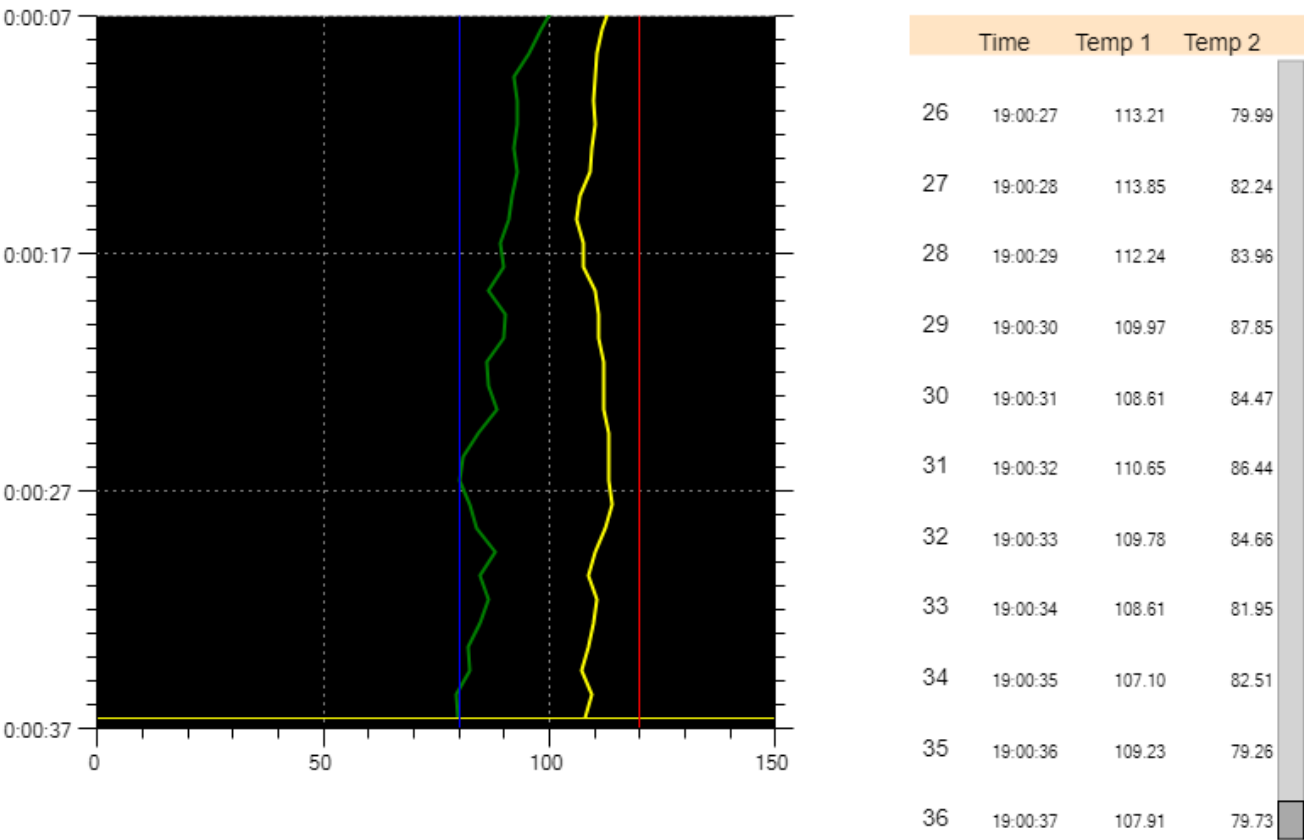
A complete listing of **RTScrollFrame** properties is found in the `_qcertgraphs_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

```
this.scrollframe1 = QCRTGraphTS.RTScrollFrame.newRTScrollFrame(this.chartVu, this.bigProcessVarArray[0], pTransform1, QCRTGraphTS.ChartConstants.RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL);
this.scrollframe1.ScrollScaleModeY = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_Y_MINMAX;
this.scrollframe1.addProcessVar(this.bigProcessVarArray[1]);
this.scrollframe1.addProcessVar(this.bigProcessVarArray[2]);
this.scrollframe1.ScrollRescaleMargin = 0.05;
this.scrollframe1.MinSamplesForAutoScale = 50;
this.chartVu.addChartObject(this.scrollframe1);
```

See the examples for the **RTSimpleSingleValuePlot** and **RTGroupMultiValuePlot** for example of the use of the **RTScrollFrame** class.

Class RTVerticalScrollFrame

ChartPlot
 RTPlot
 RTMultiValueIndicator
 RTVerticalScrollFrame



The **RTVerticalScrollFrame** is basically the same as the original **RTScrollFrame**, except it controls scrolling along the vertical axis. When you use a vertical scroll frame, typically you would have the y-scale setup as an elapsed time, or time/date based scale. It can also be setup as a numeric base scale. Otherwise it works much the same as the **RTScrollFrame**.

RTVerticalScrollFrame constructors

```
public static newRTVerticalScrollFrameChartViewProcessVarCoordsScrollMode(component: ChartView,
processvar: RTPProcessVar, initialscale: PhysicalCoordinates, scrollmode: number):
RTVerticalScrollFrame
```

```
public static newRTVerticalScrollFrameChartViewProcessVarCoordsScrollAutoScaleX(component:
ChartView, processvar: RTPProcessVar, initialscale: PhysicalCoordinates, autoscalexmode: number,
scrollymode: number): RTVerticalScrollFrame
```

```
public static newRTVerticalScrollFrameChartViewCoordsScrollScrollMode(component: ChartView,
initialscale: PhysicalCoordinates, scrollmode: number): RTVerticalScrollFrame
```

```
public static newRTVerticalScrollFrameChartViewCoordsScrollAutoScaleXScrollModeY(component:
ChartView, initialscale: PhysicalCoordinates, autoscalexmode: number, scrollymode: number) :
RTVerticalScrollFrame
```

Parameters

component

This **ChartView** component the scroll frame is placed in.

processvar

The source process variable.

initialscale

A coordinate system that serves as the initial scale for the scroll frame.

scrollymode

Specifies y-axis auto-scale mode of the scroll frame. Use one of the x-axis scroll frame constants:

RT_NO_AUTOSCALE_X - no auto-scale for the x-axis, use in non-scrolling graphs.

RT_AUTOSCALE_Y_CURRENT_SCALE - auto-scale based on current scale, use in non-scrolling graphs.

RT_AUTOSCALE_Y_MIN - autoscale x-axis minimum only, use in non-scrolling graphs.

RT_AUTOSCALE_Y_MAX - autoscale x-axis maximum only, use in non-scrolling graphs.

RT_AUTOSCALE_Y_MINMAX - autoscale x-axis minimum and maximum, use in non-scrolling graphs.

RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL - autoscale the x-axis for a fixed range, with moving maximum and minimum values, use in scrolling graphs.

RT_MAXEXTENT_FIXEDSTART_AUTOSCROLL - autoscale the x-axis with the start of the x-axis fixed, and the end of the x-axis moving, use in scrolling graphs.

RT_FIXEDNUMPOINT_AUTOSCROLL.- autoscale the x-axis for a fixed number of points, with moving maximum and minimum values, use in scrolling graphs.

autoscalexmode

Specifies x-axis auto-scale mode of the scroll frame. Use one of the y-axis scroll frame constants:

RT_NO_AUTOSCALE_X - no auto-scale for the y-axis

RT_AUTOSCALE_X_MIN - autoscale y-axis minimum only

RT_AUTOSCALE_X_MAX - autoscale y-axis maximum only

RT_AUTOSCALE_X_MINMAX. - - autoscale y-axis minimum and maximum

Selected Public Instance Properties

[AutoScaleRoundXMode](#)

Get/Set the auto-scale round mode for the x-coordinate.

Use one of the AUTOAXES round mode constants:

	AUTOAXES_EXACT, AUTOAXES_NEAR, AUTOAXES_FAR.
AutoScaleRoundYMode	Get/Set the auto-scale mode for the y-coordinate. Use one of the AUTOAXES round mode constants: AUTOAXES_EXACT, AUTOAXES_NEAR, AUTOAXES_FAR.
MaxDisplayHistory	Get/Set the maximum number of points displayed.
MinSamplesForAutoScale	Get/Set the minimum number of samples that need to be in the dataset before an auto-scale operation is carried out. This prevents the first datapoints from generating an arbitrarily small range.
ScrollRescaleMargin	Get/Set the scroll rescale margin. When the limits of the scale needs to be increased, the ScrollRescaleMargin * (current range of the x-axis) is added to the upper and lower limits of the current scale.
ScrollScaleModeX	Get/Set the scrolling mode for the x-coordinate. Use one of the x-axis scroll frame constants: RT_NO_AUTOSCALE_X, RT_AUTOSCALE_X_MIN, RT_AUTOSCALE_X_MAX, RT_AUTOSCALE_X_MINMAX
ScrollScaleModeY	Get/Set the scrolling mode for the y-coordinate. Use one of the y-axis scroll frame constants: RT_NO_AUTOSCALE_Y, RT_AUTOSCALE_Y_CURRENT_SCALE, RT_AUTOSCALE_Y_MIN, RT_AUTOSCALE_Y_MAX, RT_AUTOSCALE_Y_MINMAX, RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL, RT_MAXEXTENT_FIXEDSTART_AUTOSCROLL, RT_FIXEDNUMPOINT_AUTOSCROLL
TimeStampMode	Get/Set the time stamp mode for the time values in the process variables. Use one of the time stamp mode constants: RT_NOT_MONOTONIC_Y_MODE - not monotonic means that the x-values do not have to increase with increasing time. A real-time xy plot that plots x-values against y-values might have this characteristic. RT_MONOTONIC_Y_MODE – The default value. Monotonic means that the x-values always increase with increasing time. If the scroll frame routines know that the x-values will never “backtrack” it speeds up the search algorithm for minimum and maximum x-values to use in auto-scaling the x-axis.

A complete listing of **RTVerticalScrollFrame** properties is found in the `_qcertgraphs_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Single Channel Scrolling Graphs

Class RTSimpleSingleValuePlot

ChartPlot

RTPlot

RTSingleValueIndicator

RTSimpleSingleValuePlot

The **RTSimpleSingleValuePlot** plot class uses a template based on the **SimplePlot** class to create a real-time plot that displays **RTProcessVar** current and historical real-time data in a scrolling line, scrolling bar, or scrolling scatter plot format. Any plot object derived from the **SimplePlot** can be plotted as a scrolling graph.

RTSimpleSingleValuePlot constructors

```
public static newRTSimpleSingleValuePlotCoords(transform: PhysicalCoordinates):
RTSimpleSingleValuePlot

public static newRTSimpleSingleValuePlot(transform: PhysicalCoordinates, plottemplate:
SimplePlot, datasource: RTProcessVar): RTSimpleSingleValuePlot

public static newRTSimpleSingleValuePlotSimplePlotRTProcessVar(plottemplate: SimplePlot,
datasource: RTProcessVar): RTSimpleSingleValuePlot
```

Parameters

transform

The coordinate system for the new **RTSimpleSingleValuePlot** object.

plottemplate

This **SimplePlot** object is used as a template for the scrolling plot object.

datasource

The source process variable.

Selected Public Instance Properties

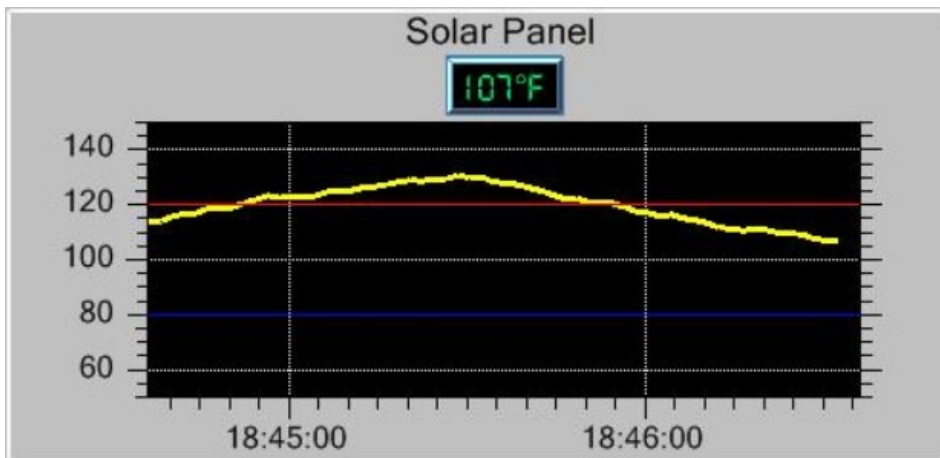
EndOfPlotLineMarker	Get/Set The end of plot marker type. Use one of the Marker marker type constants: MARKER_NULL , MARKER_VLINE , MARKER_HLINE , MARKER_HVLINE .
PlotTemplate	Get/Set the simple plot template.

A complete listing of **RTSimpleSingleValuePlot** properties is found in the `_qcrgraphts_.html` documentation file, located in the `QCRGraphTS\docs\modules` subdirectory.

Example for a simple single-channel scrolling line plot

The example below, extracted from the HomeAutomation example, file HomeAutomation, class SolarPanelUserController1, method **InitializeScrollGraph**, creates a single-channel scrolling graph.

Note: Both the **RTScrollFrame** and the **RTSimpleSingleValuePlot** objects are added to the **ChartView**. When the **ChartView.updateDraw** method is called, the **RTScrollFrame** object in the **ChartView** object list causes the scroll graph coordinate system to be re-scaled to reflect the current data values. The **RTSimpleSingleValue** object in the **ChartView** list redraws the line plot in the new re-scaled coordinate system.



[TypeScript]

```
let startTime: Date = new Date();
let endTime: Date = new Date();
QCRTGraphTS.ChartCalendar.add(endTime, QCRTGraphTS.ChartConstants.SECOND, 120);
let pTransform1: QCRTGraphTS.TimeCoordinates =
QCRTGraphTS.TimeCoordinates.newTimeCoordinates(startTime,
this.currentTemperature1.DefaultMinimumDisplayValue, endTime,
this.currentTemperature1.DefaultMaximumDisplayValue);
.
.
.

this.scrollFrame = QCRTGraphTS.RTScrollFrame.newRTScrollFrame(this.chartVu,
this.currentTemperature1, pTransform1,
QCRTGraphTS.ChartConstants.RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL);
this.scrollFrame.ScrollScaleModeY = QCRTGraphTS.ChartConstants.RT_NO_AUTOSCALE_Y;
this.scrollFrame.ScrollRescaleMargin = 0.05;
this.chartVu.addChartObject(this.scrollFrame);

let lineplot: QCRTGraphTS.SimpleLinePlot =
QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(pTransform1, attrib1);
lineplot.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
let solarPanelLinePlot: QCRTGraphTS.RTSimpleSingleValuePlot =
QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform1, lineplot,
this.currentTemperature1);
this.chartVu.addChartObject(solarPanelLinePlot);
```

[JavaScript]

```
let startTime = new Date();
let endTime = new Date();
```

```

    QCRTGraphTS.ChartCalendar.add(endTime, QCRTGraphTS.ChartConstants.SECOND, 120);
    let pTransform1 = QCRTGraphTS.TimeCoordinates.newTimeCoordinates(startTime,
this.currentTemperature1.DefaultMinimumDisplayValue, endTime,
this.currentTemperature1.DefaultMaximumDisplayValue);
    pTransform1.setGraphBorderDiagonal(x1, y1, x2, y2);
    .
    .
    .

    this.scrollFrame = QCRTGraphTS.RTScrollFrame.newRTScrollFrame(this.chartVu,
this.currentTemperature1, pTransform1,
QCRTGraphTS.ChartConstants.RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL);
    this.scrollFrame.ScrollScaleModeY = QCRTGraphTS.ChartConstants.RT_NO_AUTOSCALE_Y;
    this.scrollFrame.ScrollRescaleMargin = 0.05;
    this.chartVu.addChartObject(this.scrollFrame);

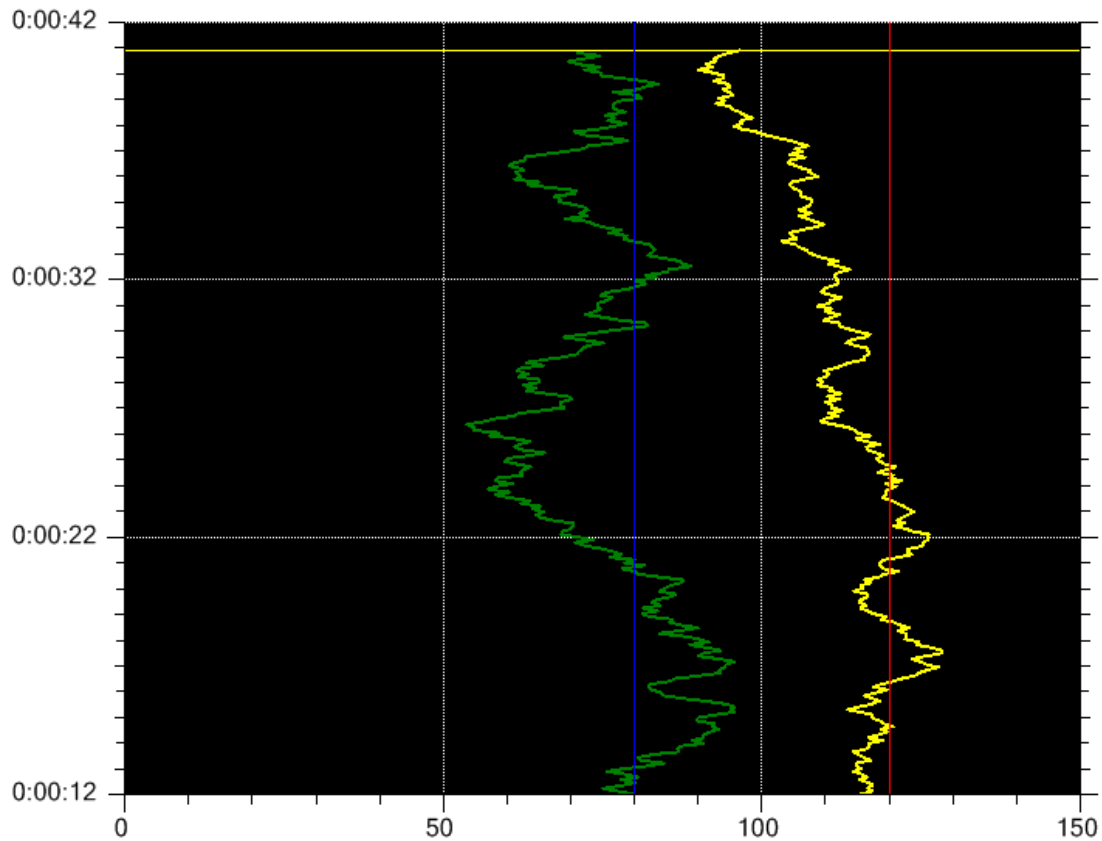
    let lineplot = QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(pTransform1, attrib1);
    lineplot.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
    let solarPanelLinePlot =
QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform1, lineplot,
this.currentTemperature1);
    this.chartVu.addChartObject(solarPanelLinePlot);

```

Example for a simple two-channel vertical scrolling line plot

The example below, extracted from the `ProcessVarDataTables.VerticalProcessVarDataTable`, method **InitializeVerticalScrollGraph**, creates a vertical, elapsed time based, two-channel scrolling graph.

Note: Both the **RTVerticalScrollFrame** and the **RTSimpleSingleValuePlot** objects are added to the **ChartView**. When the **ChartView.updateDraw** method is called, the **RTScrollFrame** object in the **ChartView** object list causes the scroll graph coordinate system to be re-scaled to reflect the current data values. The **RTSimpleSingleValue** object in the **ChartView** list redraws the line plot in the new re-scaled coordinate system.



[TypeScript]

```

this.scrollFrame =
  QCRTGraphTS.RTVerticalScrollFrame.newRTVerticalScrollFrameChartViewProcessVarCoordsScrollMode(this
    s.chartVu, this.currentTemperature1, pTransform1,
    QCRTGraphTS.ChartConstants.RT_FIXEDEXTEXT_MOVINGSTART_AUTOSCROLL);
this.scrollFrame.addProcessVar(this.currentTemperature2);
this.scrollFrame.ScrollScaleModeX = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_X_MINMAX;
// Allow 100 samples to accumulate before autoscaling y-axis. This prevents rapid
// changes of the y-scale for the first few samples
this.scrollFrame.MinSamplesForAutoScale = 100;
this.scrollFrame.ScrollRescaleMargin = 0.05;
this.chartVu.addChartObject(this.scrollFrame);
let attrib1: QCRTGraphTS.ChartAttribute =
  QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.YELLOW, 2,
    QCRTGraphTS.ChartConstants.LS_SOLID);
let lineplot1: QCRTGraphTS.SimpleLinePlot =
  QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(pTransform1, attrib1);
lineplot1.setCoordinateSwap(true);
lineplot1.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
let solarPanelLinePlot1: QCRTGraphTS.RTSimpleSingleValuePlot =
  QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform1, lineplot1,
    this.currentTemperature1);
solarPanelLinePlot1.LineColor = QCRTGraphTS.ChartColor.YELLOW;
this.chartVu.addChartObject(solarPanelLinePlot1);
let attrib2: QCRTGraphTS.ChartAttribute =
  QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.GREEN, 2,
    QCRTGraphTS.ChartConstants.LS_SOLID);
let lineplot2: QCRTGraphTS.SimpleLinePlot =
  QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(pTransform1, attrib2);
lineplot2.setCoordinateSwap(true);
lineplot2.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);

```

```
let solarPanelLinePlot2: QCRTGraphTS.RTSimpleSingleValuePlot =
QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform1, lineplot2,
this.currentTemperature2);
solarPanelLinePlot2.LineColor = QCRTGraphTS.ChartColor.YELLOW;
solarPanelLinePlot2.EndOfPlotLineMarker = QCRTGraphTS.ChartConstants.MARKER_HLINE;
this.chartVu.addChartObject(solarPanelLinePlot2);
```

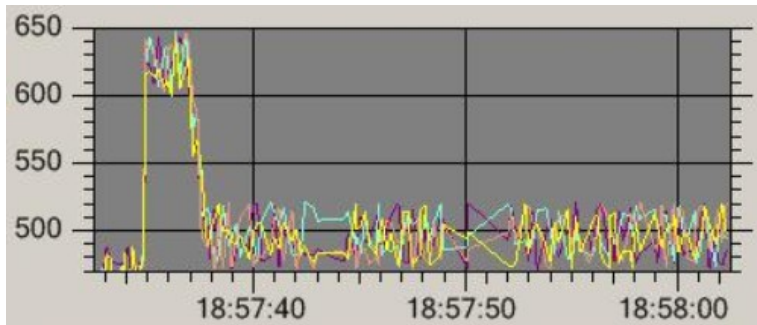
[JavaScript]

```
this.scrollFrame =
QCRTGraphTS.RTVerticalScrollFrame.newRTVerticalScrollFrameChartViewProcessVarCoordsScrollMode(thi
s.chartVu, this.currentTemperature1, pTransform1,
QCRTGraphTS.ChartConstants.RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL);
this.scrollFrame.addProcessVar(this.currentTemperature2);
this.scrollFrame.ScrollScaleModeX = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_X_MINMAX;
// Allow 100 samples to accumulate before autoscaling y-axis. This prevents rapid
// changes of the y-scale for the first few samples
this.scrollFrame.MinSamplesForAutoScale = 100;
this.scrollFrame.ScrollRescaleMargin = 0.05;
this.chartVu.addChartObject(this.scrollFrame);
let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.YELLOW, 2,
QCRTGraphTS.ChartConstants.LS_SOLID);
let lineplot1 = QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(pTransform1, attrib1);
lineplot1.setCoordinateSwap(true);
lineplot1.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
let solarPanelLinePlot1 =
QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform1, lineplot1,
this.currentTemperature1);
solarPanelLinePlot1.LineColor = QCRTGraphTS.ChartColor.YELLOW;
this.chartVu.addChartObject(solarPanelLinePlot1);
let attrib2 = QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.GREEN, 2,
QCRTGraphTS.ChartConstants.LS_SOLID);
let lineplot2 = QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(pTransform1, attrib2);
lineplot2.setCoordinateSwap(true);
lineplot2.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
let solarPanelLinePlot2 =
QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform1, lineplot2,
this.currentTemperature2);
solarPanelLinePlot2.LineColor = QCRTGraphTS.ChartColor.YELLOW;
solarPanelLinePlot2.EndOfPlotLineMarker = QCRTGraphTS.ChartConstants.MARKER_HLINE;
this.chartVu.addChartObject(solarPanelLinePlot2);
```

Example for a multi-channel scrolling line plot

The example below, extracted from the Dynamometer example, file `Dynamometers`, method `InitializeEngine1ScrollGraph`, creates a multi-channel scrolling graph.

Note: You do not have to use an `RTGroupMultiValuePlot` to plot multi-channel data in a scrolling graph. You can just use multiple `RTSimpleSingleValuePlot` objects as in the example below. You can also mix object types, including line plots, bar plots and scatter plot in the same scrolling graph.



[TypeScript]

```

    this.scrollFrame1 = QCRTGraphTS.RTScrollFrame.newRTScrollFrame(this.chartVu,
this.EngineCylinderTemp1[0], pTransform1,
    QCRTGraphTS.ChartConstants.RT_FIXEEXTENT_MOVINGSTART_AUTOSCROLL);
    this.scrollFrame1.addProcessVar(this.EngineCylinderTemp1[1]);
    this.scrollFrame1.addProcessVar(this.EngineCylinderTemp1[2]);
    this.scrollFrame1.addProcessVar(this.EngineCylinderTemp1[3]);
    this.scrollFrame1.ScrollScaleModeY = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_Y_MINMAX;
    this.scrollFrame1.ScrollRescaleMargin = 0.05;
    this.chartVu.addChartObject(this.scrollFrame1);

    for (let i: number = 0; (i < 4); i++) {
        let nulldataset: QCRTGraphTS.SimpleDataset = new QCRTGraphTS.SimpleDataset();
        nulldataset.DataName = "NULL DATASET";
        let lineplot: QCRTGraphTS.SimpleLinePlot =
QCRTGraphTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, nulldataset, attribarray[i]);
        lineplot.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
        this.rtLinePlotArray1[i] =
QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform1, lineplot,
this.EngineCylinderTemp1[i]);
        this.chartVu.addChartObject(this.rtLinePlotArray1[i]);
    }

```

[JavaScript]

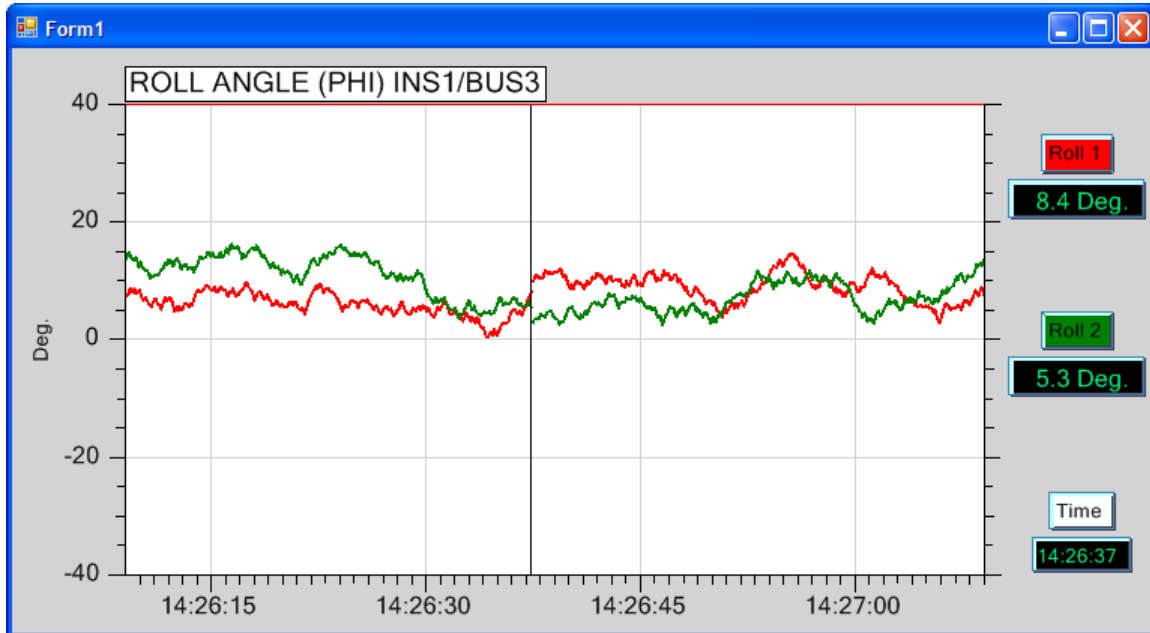
```

    scrollFrame1 = QCRTGraphTS.RTScrollFrame.newRTScrollFrame(chartVu, EngineCylinderTemp1[0],
pTransform1, QCRTGraphTS.ChartConstants.RT_FIXEEXTENT_MOVINGSTART_AUTOSCROLL);
    scrollFrame1.addProcessVar(EngineCylinderTemp1[1]);
    scrollFrame1.addProcessVar(EngineCylinderTemp1[2]);
    scrollFrame1.addProcessVar(EngineCylinderTemp1[3]);
    scrollFrame1.ScrollScaleModeY = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_Y_MINMAX;
    scrollFrame1.ScrollRescaleMargin = 0.05;
    chartVu.addChartObject(scrollFrame1);
    for (let i = 0; (i < 4); i++) {
        let nulldataset = new QCRTGraphTS.SimpleDataset();
        nulldataset.DataName = "NULL DATASET";
        let lineplot = QCRTGraphTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, nulldataset,
attribarray[i]);
        lineplot.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
        rtLinePlotArray1[i] =
QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform1, lineplot,
EngineCylinderTemp1[i]);
        chartVu.addChartObject(rtLinePlotArray1[i]);
    }

```

Example for a simple two-channel sweep line plot

The example below, extracted from the SweepGraphExample1 example, file SweepGraphExample1, method InitializeSweepGraph, creates a two-channel sweep graph. A sweep graph is defined as one which uses a sweep line at the most recent update point. Data behind (to the left of) the sweep line is the most recent data and data in front of the sweep line (to the right of) is older data. As the sweep line moves from the left to the right, new data gradually replaces the old data.



There are several special things you must do in order to create a sweep graph of this type. First, set the RescaleMargin property of the RTScrollFrame to 1.0. This places the scroll frame in the proper mode for a sweep style graph.

```
this.scrollFrame = QCRTGraphTS.RTScrollFrame.newRTScrollFrame(this.chartVu,
this.currentRollAngle1, this.pTransform1,
QCRTGraphTS.ChartConstants.RT_FIXEEXTENT_MOVINGSTART_AUTOSCROLL);
this.scrollFrame.ScrollRescaleMargin = 1;
if (this.useAutoScaleY) {
    this.scrollFrame.ScrollScaleModeY = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_Y;
    this.scrollFrame.TimeStampMode = QCRTGraphTS.ChartConstants.RT_NOT_MONOTONIC_X_MODE;
    this.scrollFrame.MinSamplesForAutoScale = 300;
    this.scrollFrame.MaxDisplayHistory = 600;
}

this.chartVu.addChartObject(this.scrollFrame);
```

Second, when creating the **RTSimpleSingleValuePlot** objects, set the SweepMode property to true for each object. Set the EndOfPlotLineMarker to QCRTGraphTS.ChartConstants.MARKER_VLINE. You only need to set this for the last RTSimpleSingleValuePlot you create, since there is no need to draw the sweep line more than once for each update.

```

let lineplot1: QCRTGraphTS.SimpleLinePlot =
QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(this.pTransform1, this.attrib1);
lineplot1.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
let sweepLinePlot1: QCRTGraphTS.RTSimpleSingleValuePlot =
QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(this.pTransform1, lineplot1,
this.currentRollAngle1);
sweepLinePlot1.SweepMode = true;

sweepLinePlot1.EndOfPlotLineMarker = QCRTGraphTS.ChartConstants.MARKER_VLINE;
this.chartVu.addChartObject(sweepLinePlot1);

```

If you want more than one line plot in the sweep graph, create an additional **SimpleLinePlot**, and related **RTSimpleSingleValuePlot**, and add it to the chart. The example program does this.

Note that in the picture above, the x-axis label values are always increasing. Even though the data in front of the sweep line is older than the data behind the sweep line, the x-axis label values do not reflect the time stamp values of the older data. Instead, x-axis label values show a time in the future, with respect to the current update time.

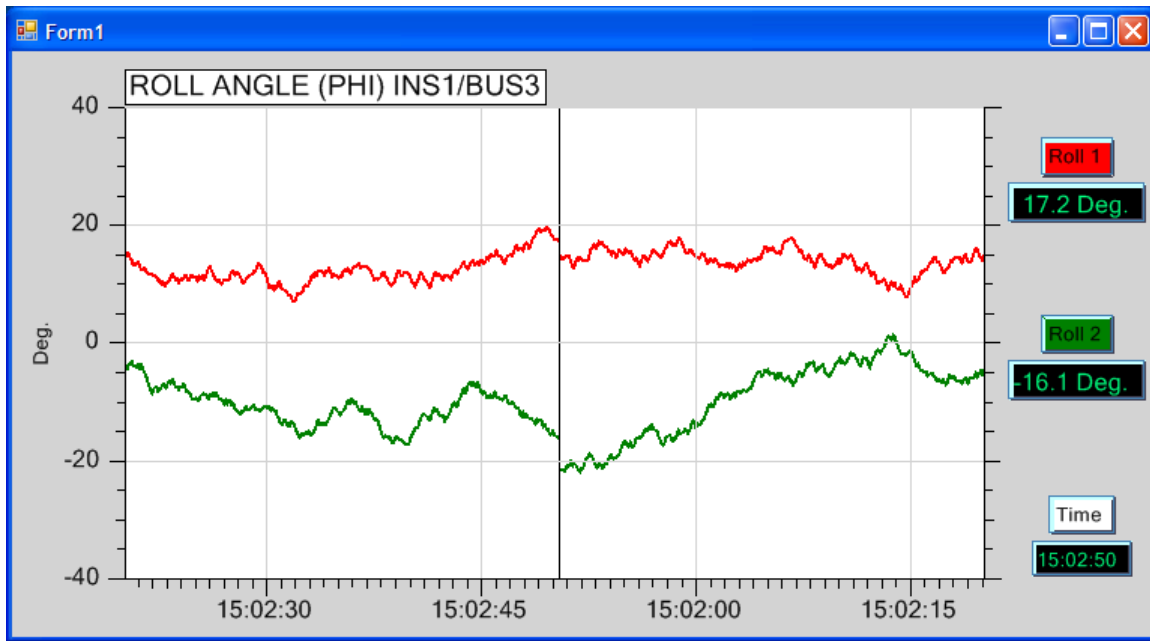
If you want to have the x-axis labels behind the sweep line display the time stamps of the most recent data values, but have the x-axis labels in front of the sweep line display the older time stamps associated with the sweep plots older data values, you need to use **RTDualScaleSweepGraphAxis**, a special x-axis class we created specifically for this type of sweep graph.

```

this.dualSweep5 = QCRTGraphTS.RTDualScaleSweepGraphAxis.newRTDualScaleSweepGraphAxisXYAxisGrids(
this.xaxis, yaxis, xAxisLab, xAxisGrid, yAxisGrid);
this.chartVu.addChartObject(this.dualSweep5);

```

In this case, create your x- and y-axes, x- and y-axis labels, and x- and y-axis grids exactly as if you were creating a normal (non-sweep style) scrolling graph. When you create the **RTDualScaleSweepGraphAxis** object, you must pass in the axis, axis labels and grid objects you have already created and added to the chart. The **RTDualScaleSweepGraphAxis** uses these objects as templates to create additional internal objects needed. The result is seen below.



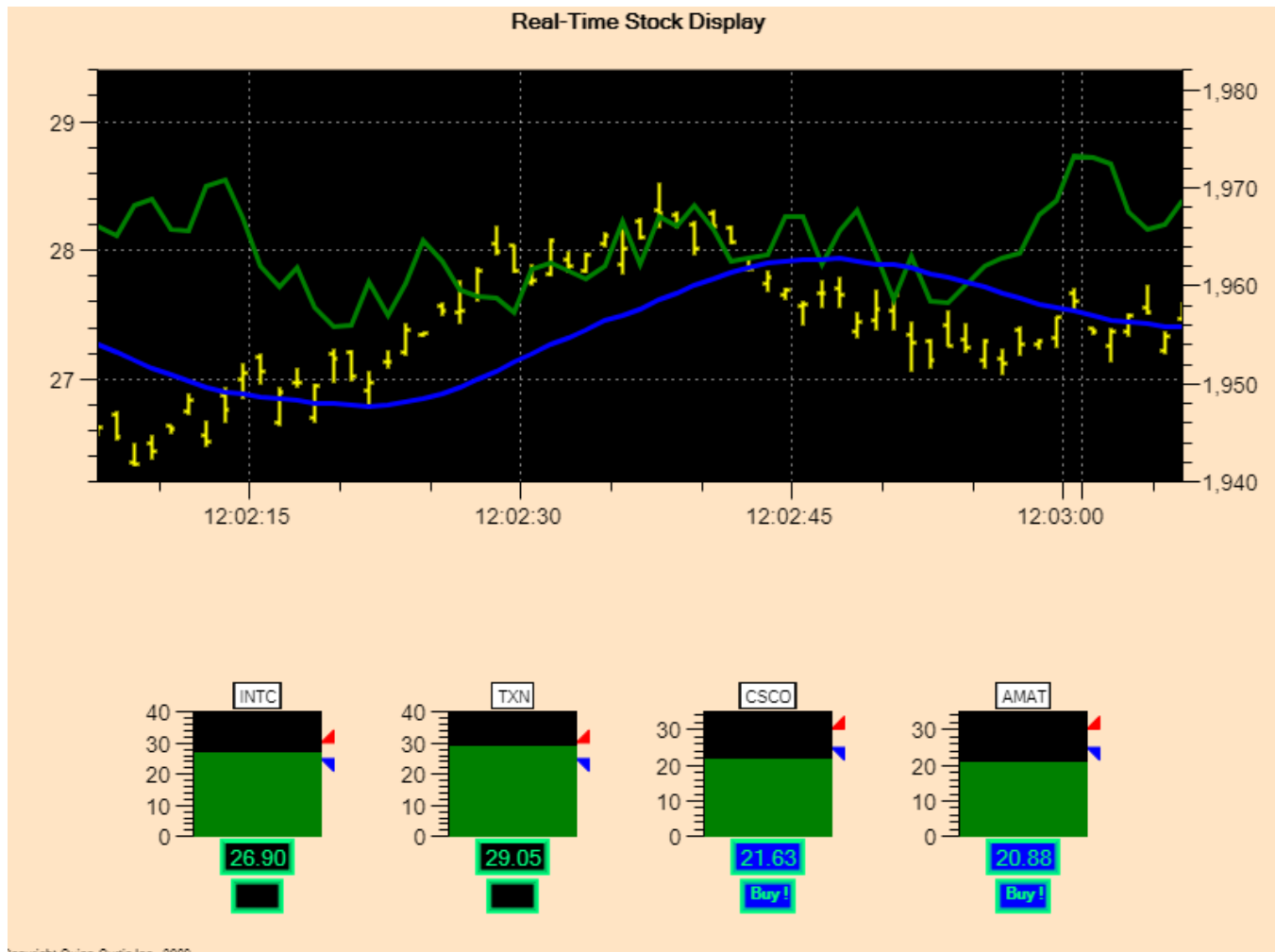
Note that in this example, the x-axis labels in front of (to the right of) the sweep line are older than the x-axis labels behind (to the left of) the sweep line. Finally, you must call the **RTDualScaleSweepGraphAxis.setMaxXTimeValue** method, with each update of the **RTProcessVar** objects, to force the axes to update properly, passing in the current time stamp.

```
if (this.dualSweep5) {
    this.dualSweep5.setMaxXTimeValue(ts);
}
```

where **ts** is a **Date** object representing the time stamp of the most recent update. You will find this call in the example program **timer1_Tick** method.

13. Multi-Channel Scrolling Plots

RTGroupMultiValuePlot



The **RTGroupMultiValuePlot** class can be used to plot multi-channel scrolling plot data. It uses the **GroupPlot** class as a template to define the attributes of the multi-channel plot. It is not the only technique, since the previous chapter had an example that plotted multiple line plots using the **RTSimpleSingleValue** plot class. If you need to plot multiple channel data, and each channel is a different plot type (i.e. one channel is a line plot, the next channel is a bar plot and the third channel is a scatter plot), you must use the technique that uses **RTSimpleSingleValue** objects.

There are two basic types of **GroupPlot** objects. The first type is a multi-channel plot. Plot objects of this type include **MultiLinePlot**, **StackedLinePlot**, **GroupBarPlot**, **StackedBarPlot**. These objects are characterized as having unique **ChartAttribute** objects defining the colors and fill characteristic of each channel. The second type is the multi-variable plot. These are objects that require two or more y-values

to characterize the plot at a given instance in time. These include the **HistogramPlot**, **BubblePlot**, **FloatingBarPlot**, **CellPlot** and **OHLCPLOT** classes. Usually one instance of one of these multi-variable objects is characterized by a single **ChartAttribute**, similar to the **SimplePlot** objects. Both types of **GroupPlot** objects can be used in scrolling graphs.

Multi-Channel Scrolling Graphs

Class RTGroupMultiValuePlot

ChartPlot
 RTPlot
 RTMultiValueIndicator
 RTGroupMultiValuePlot

The **RTGroupMultiValuePlot** plot class uses a template based on the **GroupPlot** class to create a real-time plot that displays a collection of **RTProcessVar** objects as a group plot in a scrolling graph format.

RTGroupMultiValuePlot constructors

```
[public static newRTGroupMultiValuePlot(transform: PhysicalCoordinates): RTGroupMultiValuePlot

public static newRTGroupMultiValuePlotCoordsGroupPlotProcessVars(transform: PhysicalCoordinates,
plottemplate: GroupPlot, datasource: RTProcessVar[]): RTGroupMultiValuePlot

public static newRTGroupMultiValuePlotCoordsGroupPlotMultiProcessVar(transform:
PhysicalCoordinates, plottemplate: GroupPlot, datasource: RTMultiProcessVar):
RTGroupMultiValuePlot
```

Parameters

- transform*
The coordinate system for the new **RTGroupMultiPlot** object.
- plottemplate*
A template defining the group plot object.
- datasource*
An array of **RTProcessVar** objects, one for each group in the group plot template.

Selected Public Instance Properties

EndOfPlotLineMarker	Get/Set The end of plot marker type. Use one of the Marker marker type constants: MARKER_NULL, MARKER_VLINE, MARKER_HLINE,

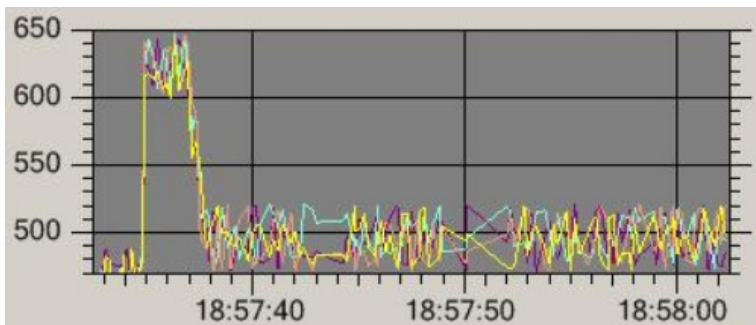
	MARKER HVLINE.
MarkerGroupName	Get/Set the group number that is used for the end of plot line marker.
PlotTemplate	Get/Set the group plot template.

A complete listing of **RTGroupMultiValuePlot** properties is found in the `_qcrgraphts_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example for a multi-channel scrolling line plot

The example below, extracted from the Dynamometer example, file `Dynamometer`, method **InitializeEngine2ScrollGraph**, creates a multi-channel scrolling graph.

Note: You do not have to use an **RTGroupMultiValuePlot** to plot multi-channel data in a scrolling graph. You can just use multiple **RTSimpleSingleValuePlot** objects as in the `InitializeEngine1ScrollGraph` method.



[TypeScript]

```
this.scrollFrame2 = QCRTGraphTS.RTScrollFrame.newRTScrollFrame(this.chartVu,
this.EngineCylinderTemp2[0], pTransform1,
QCRTGraphTS.ChartConstants.RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL);
this.scrollFrame2.addProcessVar(this.EngineCylinderTemp2[1]);
this.scrollFrame2.addProcessVar(this.EngineCylinderTemp2[2]);
this.scrollFrame2.addProcessVar(this.EngineCylinderTemp2[3]);
this.scrollFrame2.ScrollScaleModeY = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_Y_MINMAX;
this.scrollFrame2.ScrollRescaleMargin = 0.05;

let multilineplot: QCRTGraphTS.MultiLinePlot =
QCRTGraphTS.MultiLinePlot.newMultiLinePlotCoordsAttrib(pTransform1, attribarray);
multilineplot.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
this.rtMultiLinePlot =
QCRTGraphTS.RTGroupMultiValuePlot.newRTGroupMultiValuePlotCoordsGroupPlotProcessVars(pTransform1,
multilineplot, this.EngineCylinderTemp2);
this.chartVu.addChartObject(this.rtMultiLinePlot);

this.chartVu.addChartObject(this.scrollFrame2);
```

[JavaScript]

```

scrollFrame2 = QCRTGraphTS.RTScrollFrame.newRTScrollFrame(chartVu, EngineCylinderTemp2[0],
pTransform1, QCRTGraphTS.ChartConstants.RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL);
scrollFrame2.addProcessVar(EngineCylinderTemp2[1]);
scrollFrame2.addProcessVar(EngineCylinderTemp2[2]);
scrollFrame2.addProcessVar(EngineCylinderTemp2[3]);
scrollFrame2.ScrollScaleModeY = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_Y_MINMAX;
scrollFrame2.ScrollRescaleMargin = 0.05;

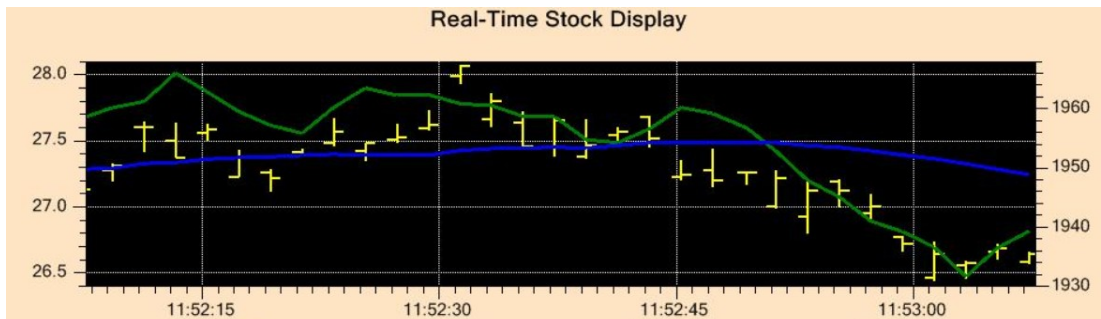
let multilineplot = QCRTGraphTS.MultiLinePlot.newMultiLinePlotCoordsAttrib(pTransform1,
attribarray);
multilineplot.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
rtMultiLinePlot =
QCRTGraphTS.RTGroupMultiValuePlot.newRTGroupMultiValuePlotCoordsGroupPlotProcessVars(pTransform1,
multilineplot, EngineCylinderTemp2);
chartVu.addChartObject(rtMultiLinePlot);

chartVu.addChartObject(scrollFrame2);

```

Example for multi-scale, multi-axis scrolling graph combining stock open-high-low-close plots with line plots.

The example below, extracted from the RTStockDisplay example, method **InitializeScrollGraph**, creates a multi-channel scrolling graph the combines an open-high-low-close plots with line plots using two different scales.



[TypeScript]

```

this.scrollFrame1 = QCRTGraphTS.RTScrollFrame.newRTScrollFrame(this.chartVu, this.stockOpen1,
pTransform1, QCRTGraphTS.ChartConstants.RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL);
this.scrollFrame1.ScrollScaleModeY = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_Y_MINMAX;
// Need to add this ProcessVar s to have auto-scale work for all values of OHLC plot
this.scrollFrame1.addProcessVar(this.stockHigh1);
this.scrollFrame1.addProcessVar(this.stockLow1);
this.scrollFrame1.addProcessVar(this.stockClose1);
this.scrollFrame1.ScrollRescaleMargin = 0.01;
this.chartVu.addChartObject(this.scrollFrame1);
this.scrollFrame2 = QCRTGraphTS.RTScrollFrame.newRTScrollFrame(this.chartVu, this.NASDAQChannel,
pTransform2, QCRTGraphTS.ChartConstants.RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL);
this.scrollFrame2.ScrollScaleModeY = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_Y_MINMAX;
this.scrollFrame2.ScrollRescaleMargin = 0.01;
this.chartVu.addChartObject(this.scrollFrame2);
let attrib1: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.YELLOW, 2,
QCRTGraphTS.ChartConstants.LS_SOLID);
let ohlcplot1: QCRTGraphTS.OHLCPlot =
QCRTGraphTS.OHLCPlot.newOHLCPlotCoordsWidthAttrib(pTransform1,
QCRTGraphTS.ChartCalendar.getCalendarWidthValue(QCRTGraphTS.ChartConstants.SECOND, 0.5),
attrib1);
ohlcplot1.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
let stockvars: QCRTGraphTS.RTProcessVar[] = [
    this.stockOpen1,

```

```

    this.stockHigh1,
    this.stockLow1,
    this.stockClose1];
this.rtPlot1 =
QCRTGraphTS.RTGroupMultiValuePlot.newRTGroupMultiValuePlotCoordsGroupPlotProcessVars(pTransform1,
ohlplot1, stockvars);
this.chartVu.addChartObject(this.rtPlot1);
let attrib2: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.GREEN, 3,
QCRTGraphTS.ChartConstants.LS_SOLID);

let lineplot2: QCRTGraphTS.SimpleLinePlot =
QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(pTransform2, attrib2);
lineplot2.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
this.rtPlot2 = QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform2,
lineplot2, this.NASDAQChannel);
this.chartVu.addChartObject(this.rtPlot2);
let attrib3: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.BLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID);
let lineplot3: QCRTGraphTS.SimpleLinePlot =
QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(pTransform2, attrib3);
lineplot3.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
this.rtPlot3 = QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform1,
lineplot3, this.movingAverageStock);
this.chartVu.addChartObject(this.rtPlot3);

```

[JavaScript]

```

this.scrollFrame1 = QCRTGraphTS.RTScrollFrame.newRTScrollFrame(this.chartVu, this.stockOpen1,
pTransform1, QCRTGraphTS.ChartConstants.RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL);
this.scrollFrame1.ScrollScaleModeY = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_Y_MINMAX;
// Need to add this ProcessVar s to have auto-scale work for all values of OHLC plot
this.scrollFrame1.addProcessVar(this.stockHigh1);
this.scrollFrame1.addProcessVar(this.stockLow1);
this.scrollFrame1.addProcessVar(this.stockClose1);
this.scrollFrame1.ScrollRescaleMargin = 0.01;
this.chartVu.addChartObject(this.scrollFrame1);
this.scrollFrame2 = QCRTGraphTS.RTScrollFrame.newRTScrollFrame(this.chartVu, this.NASDAQChannel,
pTransform2, QCRTGraphTS.ChartConstants.RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL);
this.scrollFrame2.ScrollScaleModeY = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_Y_MINMAX;
this.scrollFrame2.ScrollRescaleMargin = 0.01;
this.chartVu.addChartObject(this.scrollFrame2);
let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.YELLOW, 2,
QCRTGraphTS.ChartConstants.LS_SOLID);
let ohlplot1 = QCRTGraphTS.OHLCPlot.newOHLCPlotCoordsWidthAttrib(pTransform1,
QCRTGraphTS.ChartCalendar.getCalendarWidthValue(QCRTGraphTS.ChartConstants.SECOND, 0.5),
attrib1);
ohlplot1.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
let stockvars = [
    this.stockOpen1,
    this.stockHigh1,
    this.stockLow1,
    this.stockClose1
];
this.rtPlot1 =
QCRTGraphTS.RTGroupMultiValuePlot.newRTGroupMultiValuePlotCoordsGroupPlotProcessVars(pTransform1,
ohlplot1, stockvars);
this.chartVu.addChartObject(this.rtPlot1);
let attrib2 = QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.GREEN, 3,
QCRTGraphTS.ChartConstants.LS_SOLID);

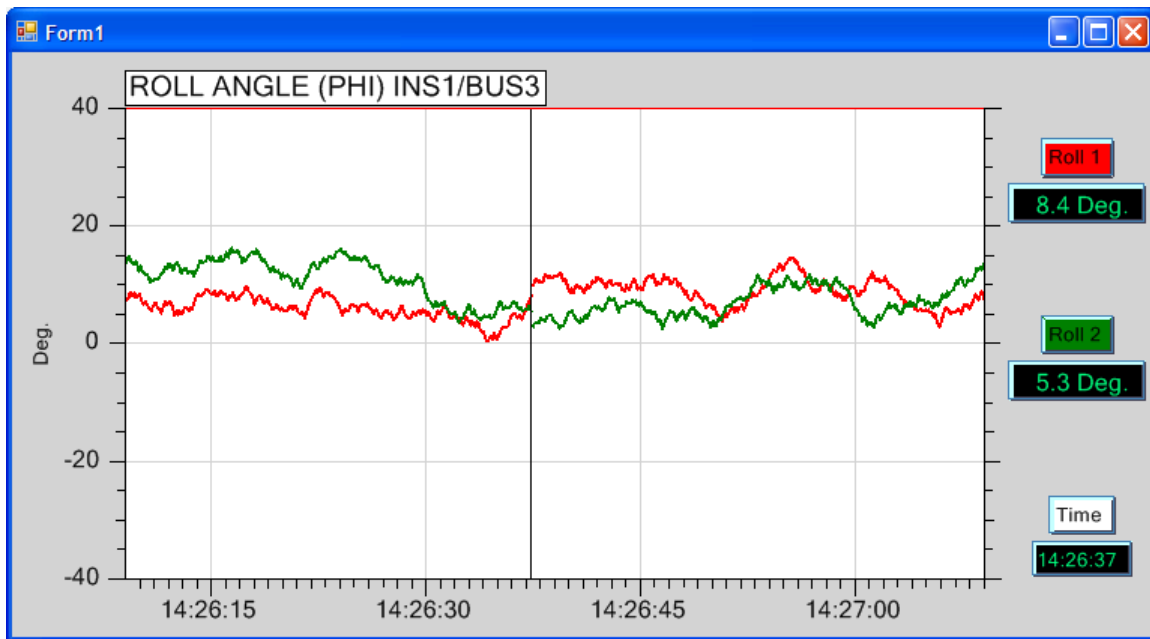
let lineplot2 = QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(pTransform2, attrib2);
lineplot2.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
this.rtPlot2 = QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform2,
lineplot2, this.NASDAQChannel);
this.chartVu.addChartObject(this.rtPlot2);
let attrib3 = QCRTGraphTS.ChartAttribute.newChartAttribute3(QCRTGraphTS.ChartColor.BLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID);

```

```
let lineplot3 = QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(pTransform2, attrib3);
lineplot3.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
this.rtPlot3 = QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(pTransform1,
lineplot3, this.movingAverageStock);
this.chartVu.addChartObject(this.rtPlot3);
```

Example for a simple two-channel sweep line plot

The example below, extracted from the SweepGraphExample1 example, file SweepGraphUserControl1, method InitializeSweepGraph, creates a two-channel sweep graph. A sweep graph is defined as one which uses a sweep line at the most recent update point. Data behind (to the left of) the sweep line is the most recent data and data in front of the sweep line (to the right of) is older data. As the sweep line moves from the left to the right, new data gradually replaces the old data.



There are several special things you must do in order to create a sweep graph of this type. First, set the RescaleMargin property of the RTScrollFrame to 1.0. This places the scroll frame in the proper mode for a sweep style graph.

```
this.scrollFrame = QCRTGraphTS.RTScrollFrame.newRTScrollFrame(this.chartVu,
this.currentRollAngle1, this.pTransform1,
QCRTGraphTS.ChartConstants.RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL);
this.scrollFrame.ScrollRescaleMargin = 1;
if (this.useAutoScaleY) {
    this.scrollFrame.ScrollScaleModeY = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_Y;
    this.scrollFrame.TimeStampMode = QCRTGraphTS.ChartConstants.RT_NOT_MONOTONIC_X_MODE;
    this.scrollFrame.MinSamplesForAutoScale = 300;
    this.scrollFrame.MaxDisplayHistory = 600;
}

this.scrollFrame.addProcessVar(this.currentRollAngle2);
this.chartVu.addChartObject(this.scrollFrame);
```

Second, when creating the **RTGroupMultiValuePlot** objects, set the `SweepMode` property to true for each object. Set the `EndOfPlotLineMarker` to `QCRTGraphTS.ChartConstants.MARKER_VLINE`. You only need to set this for the last **RTGroupMultiValuePlot** you create, since there is no need to draw the sweep line more than once for each update.

```
let lineplot1: QCRTGraphTS.SimpleLinePlot =
  QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(this.pTransform1, this.attrib1);
lineplot1.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
let sweepLinePlot1: QCRTGraphTS.RTSimpleSingleValuePlot =
  QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(this.pTransform1, lineplot1,
    this.currentRollAngle1);
sweepLinePlot1.SweepMode = true;

this.chartVu.addChartObject(sweepLinePlot1);
let lineplot2: QCRTGraphTS.SimpleLinePlot =
  QCRTGraphTS.SimpleLinePlot.newSimpleLinePlotCoordsAttrib(this.pTransform1, this.attrib2);
lineplot2.setFastClipMode(QCRTGraphTS.ChartConstants.FASTCLIP_X);
let sweepLinePlot2: QCRTGraphTS.RTSimpleSingleValuePlot =
  QCRTGraphTS.RTSimpleSingleValuePlot.newRTSimpleSingleValuePlot(this.pTransform1, lineplot2,
    this.currentRollAngle2);
sweepLinePlot2.SweepMode = true;
sweepLinePlot2.EndOfPlotLineMarker = QCRTGraphTS.ChartConstants.MARKER_VLINE;
this.chartVu.addChartObject(sweepLinePlot2);
```

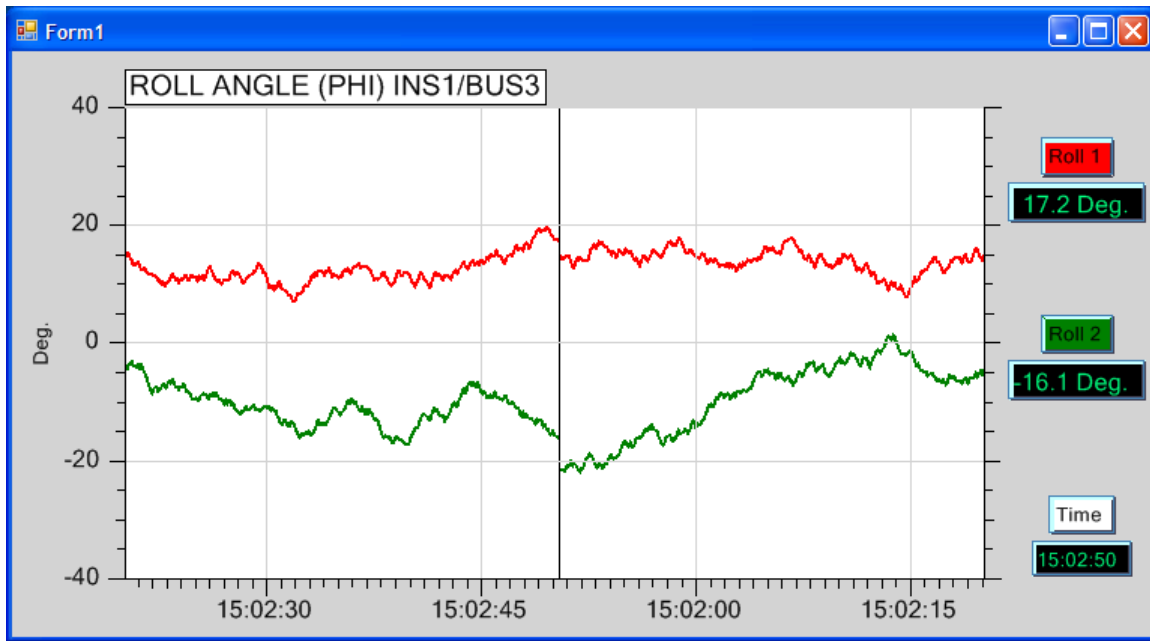
You can use this technique to add as many channels of data to sweep graph as you want, using a single **RTGroupMultiValuePlot**.

Note that in the picture above, the x-axis label values are always increasing. Even though the data in front of the sweep line is older than the data behind the sweep line, the x-axis label values do not reflect the time stamp values of the older data. Instead, x-axis label values show a time in the future, with respect to the current update time.

If you want to have the x-axis labels behind the sweep line display the time stamps of the most recent data values, but have the x-axis labels in front of the sweep line display the older time stamps associated with the sweep plots older data values, you need to use **RTDualScaleSweepGraphAxis**, a special x-axis class we created specifically for this type of sweep graph.

```
this.dualSweep5 = QCRTGraphTS.RTDualScaleSweepGraphAxis.newRTDualScaleSweepGraphAxisXYAxisGrids(
  this.xaxis, yaxis, xAxisLab, xAxisGrid, yAxisGrid);
this.chartVu.addChartObject(this.dualSweep5);
```

In this case, create your x- and y-axes, x- and y-axis labels, and x- and y-axis grids exactly as if you were creating a normal (non-sweep style) scrolling graph. When you create the **RTDualScaleSweepGraphAxis** object, you must pass in the axis, axis labels and grid objects you have already created and added to the chart. The **RTDualScaleSweepGraphAxis** uses these objects as templates to create additional internal objects needed. The result is seen below.



Note that in this example, the x-axis labels in front of (to the right of) the sweep line are older than the x-axis labels behind (to the left of) the sweep line. Finally, you must call the **RTDualScaleSweepGraphAxis.setMaxXTimeValue** method, with each update of the **RTProcessVar** objects, to force the axes to update properly, passing in the current time stamp.

```
if (this.dualSweep5) {
    this.dualSweep5.setMaxXTimeValue(ts);
}
```

where **ts** is a **Date** object representing the time stamp of the most recent update. You will find this call in the example program **timer1_Tick** method.

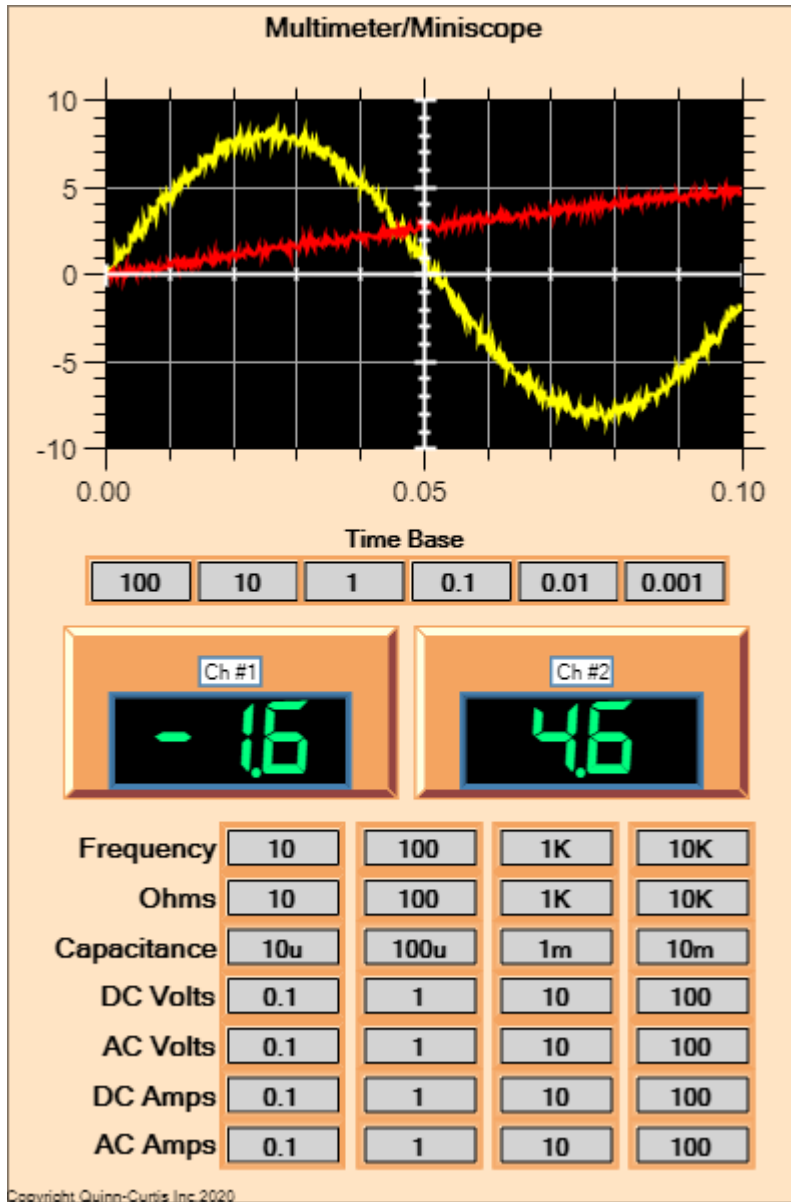
14. Buttons, Track Bars and Other Form Control Classes

RTControlButton

RTControlTrackBar

RTFormControlPanelMeter

RTFormControlGrid



Real-time displays often require user interface features such as buttons and track 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.

We created Canvas versions of a simple scrollbar and button controls. These have the class names **ScrollbarArea** and **ButtonArea**, and are described in the QCChart2D documentation. They are compatible both mouse and touch based systems. The QCRTGraph software contains two additional controls (**RTControlTrackBar**, and **RTControlButton**), subclassed from ScrollbarArea and ButtonArea. The **RTControlTrackBar** control adds a trackbar style pointer thumb, axis and tickmarks to the scrollbar.. The **RTControlButton** is pretty much the same as the ButtonArea control. Typical examples of each type of control are seen below.

ButtonArea



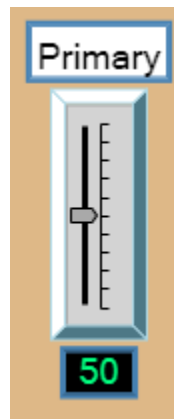
ScrollbarArea



RTControlButton



RTControlTrackBar



Control Buttons

Class RTControlButton

ButtonArea

RTControlButton

The **RTControlButton** class is subclassed from **ButtonArea** class, which is in turn 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.

RTControlButton constructor

```

public static newRTControlButton(buttontype: number): RTControlButton

public static newRTControlButtonComponentType(component: ChartView, buttontype: number):
RTControlButton

public static newRTControlButtonPosTypeString(component: ChartView, pos: ChartRectangle2D,
buttontype: number, text: string): RTControlButton

public static newRTControlButtonPosTypeStringAttrib(component: ChartView, pos: ChartRectangle2D,
buttontype: number, text: string, buttonattrib: ChartAttribute): RTControlButton

```

Parameters

<i>component</i>	A reference to the ChartView object that the button is placed in.
<i>buttontype</i>	The button type. Use one of the button type constants: MOMENTARYBUTTON, TOGGLEBUTTON, RADIOBUTTON
<i>pos</i>	A reference to the positioning rectangle, specifying the button position and size using chart normalized coordinates.
<i>buttontype</i>	The button type. Use one of the button type constants: MOMENTARYBUTTON, TOGGLEBUTTON, RADIOBUTTON
<i>text</i>	The text string used as the button label.
<i>buttonattrib</i>	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

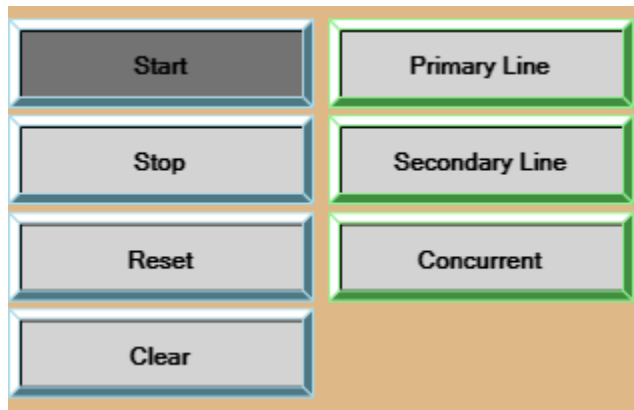
<u>ButtonChecked</u>	Get/Set the button check state.
<u>ButtonCheckedColor</u>	Get/Set the color of the button when the button is checked.
<u>ButtonCheckedText</u>	Get/Set the button text when the button is checked.
<u>ButtonCheckedTextColor</u>	Get/Set the color of the button text when the button is checked.
<u>ButtonSubtype</u>	Get/Set the button subtype. RADIOBUTTON, MOMENTARYBUTTON, TOGGLEBUTTON
<u>ButtonUncheckedColor</u>	Get/Set the color of the button when the button is unchecked.
<u>ButtonUncheckedText</u>	Get/Set the button text when the button is unchecked.
<u>ButtonUncheckedTextColor</u>	Get/Set the color of the button text when the button is unchecked.
<u>RoundedButonCornerRadius</u>	Get/Set the a radius to use on the corners of the button.
<u>ButtonFont</u>	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: QCRTGraphTS.RTControlButton[] = [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 FetalMonitor has examples of all three types of buttons, momentary, toggle and radio.



[TypeScript]

```
StartButton: QCRTGraphTS.RTControlButton =
QCRTGraphTS.RTControlButton.newRTControlButton(QCRTGraphTS.ChartConstants.RT_CONTROL_RADIOBUTTON_
SUBTYPE);

StopButton: QCRTGraphTS.RTControlButton =
QCRTGraphTS.RTControlButton.newRTControlButton(QCRTGraphTS.ChartConstants.RT_CONTROL_RADIOBUTTON_
SUBTYPE);

ResetButton: QCRTGraphTS.RTControlButton =
QCRTGraphTS.RTControlButton.newRTControlButton(QCRTGraphTS.ChartConstants.RT_CONTROL_MOMENTARYBUT
TON_SUBTYPE);

ClearButton: QCRTGraphTS.RTControlButton =
QCRTGraphTS.RTControlButton.newRTControlButton(QCRTGraphTS.ChartConstants.RT_CONTROL_MOMENTARYBUT
TON_SUBTYPE);

PrimaryLineButton: QCRTGraphTS.RTControlButton =
QCRTGraphTS.RTControlButton.newRTControlButton(QCRTGraphTS.ChartConstants.RT_CONTROL_TOGGLEBUTTON
_SUBTYPE);

SecondaryLineButton: QCRTGraphTS.RTControlButton =
QCRTGraphTS.RTControlButton.newRTControlButton(QCRTGraphTS.ChartConstants.RT_CONTROL_TOGGLEBUTTON
_SUBTYPE);
```

```

Concurrent: QCRTGraphTS.RTControlButton =
QCRTGraphTS.RTControlButton.newRTControlButton(QCRTGraphTS.ChartConstants.RT_CONTROL_TOGGLEBUTTON
_SUBTYPE);

startStopControlGrid: QCRTGraphTS.RTFormControlGrid = new QCRTGraphTS.RTFormControlGrid();
.
.
.
public InitializeStartStopButtons() {
    let buttonfont: QCRTGraphTS.ChartFont = this.font12Bold;
    if (!this.chartVu) return;
    let pTransform1: QCRTGraphTS.CartesianCoordinates =
QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
    pTransform1.setGraphBorderDiagonal(0.01, .65, 0.2, 0.98);
    let attrib1: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 5,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.LIGHTBLUE);
    let buttonlist1: QCRTGraphTS.RTControlButton[] = new Array<QCRTGraphTS.RTControlButton>();
    this.StartButton.ButtonUncheckedText = "Start";
    this.StartButton.ButtonChecked = true;
    this.StartButton.ButtonAreaEventListener = (e: QCRTGraphTS.ButtonAreaEventArgs) => {
        this.selector_Button_Click(e);
    }
    this.chartVu.addMouseListener(this.StartButton);

    this.StartButton.ButtonFont = buttonfont;
    buttonlist1.push(this.StartButton);
    this.StopButton.ButtonUncheckedText = "Stop";
    this.StopButton.ButtonChecked = false;
    this.StopButton.ButtonAreaEventListener = (e: QCRTGraphTS.ButtonAreaEventArgs) => {
        this.selector_Button_Click(e);
    }
    this.chartVu.addMouseListener(this.StopButton);
    this.StopButton.ButtonFont = buttonfont;
    buttonlist1.push(this.StopButton);
    // Start and Stop and Radio buttons, so set the setButtonGroupAssociates
    let buttonassoc: QCRTGraphTS.RTControlButton[] = [this.StartButton, this.StopButton];
    this.StartButton.setButtonGroupAssociates(buttonassoc);

    this.ResetButton.ButtonUncheckedText = "Reset";
    this.ResetButton.ButtonChecked = false;
    this.ResetButton.ButtonAreaEventListener = (e: QCRTGraphTS.ButtonAreaEventArgs) => {
        this.selector_Button_Click(e);
    }
    this.chartVu.addMouseListener(this.ResetButton);
    this.ResetButton.ButtonFont = buttonfont;
    buttonlist1.push(this.ResetButton);
    this.ClearButton.ButtonUncheckedText = "Clear";
    this.ClearButton.ButtonChecked = false;
    this.ClearButton.ButtonAreaEventListener = (e: QCRTGraphTS.ButtonAreaEventArgs) => {
        this.selector_Button_Click(e);
    }
    this.chartVu.addMouseListener(this.ClearButton);

    this.ClearButton.ButtonFont = buttonfont;
    buttonlist1.push(this.ClearButton);

    let numColumns: number = 1;
    let numRows: number = 4;
    this.startStopControlGrid =
QCRTGraphTS.RTFormControlGrid.newRTFormControlGridCoordsCRAattrib(pTransform1, buttonlist1,
numColumns, numRows, attrib1);
    this.startStopControlGrid.CellRowMargin = 0.1;
    this.startStopControlGrid.CellColumnMargin = 0;
    this.startStopControlGrid.FormControlTemplate.Frame3DEnable = this.flag3DBorder;
    this.chartVu.addChartObject(this.startStopControlGrid);
    let buttonlist2: QCRTGraphTS.RTControlButton[] = new Array<QCRTGraphTS.RTControlButton>();

```

```

let attrib2: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 5,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.LIGHTGREEN);
let pTransform2: QCRTGraphTS.CartesianCoordinates =
QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
pTransform2.setGraphBorderDiagonal(0.21, .65, 0.4, 0.98);
this.PrimaryLineButton.ButtonUncheckedText = "Primary Line";
this.PrimaryLineButton.ButtonChecked = false;
this.PrimaryLineButton.ButtonAreaEventListener = (e: QCRTGraphTS.ButtonAreaEventArgs) => {
    this.selector_Button_Click(e);
}
this.chartVu.addMouseListener(this.PrimaryLineButton);

this.PrimaryLineButton.ButtonFont = buttonfont;
buttonlist2.push(this.PrimaryLineButton);
this.SecondaryLineButton.ButtonUncheckedText = "Secondary Line";
this.SecondaryLineButton.ButtonChecked = false;
this.SecondaryLineButton.ButtonAreaEventListener = (e: QCRTGraphTS.ButtonAreaEventArgs) => {
    this.selector_Button_Click(e);
}
this.chartVu.addMouseListener(this.SecondaryLineButton);
this.SecondaryLineButton.ButtonFont = buttonfont;
buttonlist2.push(this.SecondaryLineButton);
this.Concurrent.ButtonUncheckedText = "Concurrent";
this.Concurrent.ButtonChecked = false;
this.Concurrent.ButtonAreaEventListener = (e: QCRTGraphTS.ButtonAreaEventArgs) => {
    this.selector_Button_Click(e);
}
this.chartVu.addMouseListener(this.Concurrent);

this.Concurrent.ButtonFont = buttonfont;
buttonlist2.push(this.Concurrent);

numColumns = 1;
numRows = 4;
let controlgrid2: QCRTGraphTS.RTFormControlGrid =
QCRTGraphTS.RTFormControlGrid.newRTFormControlGridCoordsCRAAttrib(pTransform2, buttonlist2,
numColumns, numRows, attrib2);
controlgrid2.CellRowMargin = 0.1;
controlgrid2.CellColumnMargin = 0;
controlgrid2.FormControlTemplate.Frame3DEnable = this.flag3DBorder;
this.chartVu.addChartObject(controlgrid2);
}

```

[JavaScript]

```

var StartButton =
QCRTGraphTS.RTControlButton.newRTControlButton(QCRTGraphTS.ChartConstants.RT_CONTROL_RADIOBUTTON_
SUBTYPE);
var StopButton =
QCRTGraphTS.RTControlButton.newRTControlButton(QCRTGraphTS.ChartConstants.RT_CONTROL_RADIOBUTTON_
SUBTYPE);
var ResetButton =
QCRTGraphTS.RTControlButton.newRTControlButton(QCRTGraphTS.ChartConstants.RT_CONTROL_MOMENTARYBUT
TON_SUBTYPE);
var ClearButton =
QCRTGraphTS.RTControlButton.newRTControlButton(QCRTGraphTS.ChartConstants.RT_CONTROL_MOMENTARYBUT
TON_SUBTYPE);
var PrimaryLineButton =
QCRTGraphTS.RTControlButton.newRTControlButton(QCRTGraphTS.ChartConstants.RT_CONTROL_TOGGLEBUTTON
_SUBTYPE);
var SecondaryLineButton =
QCRTGraphTS.RTControlButton.newRTControlButton(QCRTGraphTS.ChartConstants.RT_CONTROL_TOGGLEBUTTON
_SUBTYPE);
var Concurrent =
QCRTGraphTS.RTControlButton.newRTControlButton(QCRTGraphTS.ChartConstants.RT_CONTROL_TOGGLEBUTTON
_SUBTYPE);
var startStopControlGrid = new QCRTGraphTS.RTFormControlGrid();

```

```

.
.
.
function InitializeStartStopButtons() {
  let buttonfont = font12Bold;
  if (!chartVu) return;
  let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
  pTransform1.setGraphBorderDiagonal(0.01, .65, 0.2, 0.98);
  let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 5,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.LIGHTBLUE);
  let buttonlist1 = new Array();
  StartButton.ButtonUncheckedText = "Start";
  StartButton.ButtonChecked = true;
  StartButton.ButtonAreaEventListener = (e) => {
    selector_Button_Click(e);
  };
  chartVu.addMouseListener(StartButton);
  StartButton.ButtonFont = buttonfont;
  buttonlist1.push(StartButton);
  StopButton.ButtonUncheckedText = "Stop";
  StopButton.ButtonChecked = false;
  StopButton.ButtonAreaEventListener = (e) => {
    selector_Button_Click(e);
  };
  chartVu.addMouseListener(StopButton);
  StopButton.ButtonFont = buttonfont;
  buttonlist1.push(StopButton);
  // Start and Stop and Radio buttons, so set the setButtonGroupAssociates
  let buttonassoc = [StartButton, StopButton];
  StartButton.setButtonGroupAssociates(buttonassoc);
  ResetButton.ButtonUncheckedText = "Reset";
  ResetButton.ButtonChecked = false;
  ResetButton.ButtonAreaEventListener = (e) => {
    selector_Button_Click(e);
  };
  chartVu.addMouseListener(ResetButton);
  ResetButton.ButtonFont = buttonfont;
  buttonlist1.push(ResetButton);
  ClearButton.ButtonUncheckedText = "Clear";
  ClearButton.ButtonChecked = false;
  ClearButton.ButtonAreaEventListener = (e) => {
    selector_Button_Click(e);
  };
  chartVu.addMouseListener(ClearButton);
  ClearButton.ButtonFont = buttonfont;
  buttonlist1.push(ClearButton);
  let numColumns = 1;
  let numRows = 4;
  startStopControlGrid =
QCRTGraphTS.RTFormControlGrid.newRTFormControlGridCoordsCRAattrib(pTransform1, buttonlist1,
numColumns, numRows, attrib1);
  startStopControlGrid.CellRowMargin = 0.1;
  startStopControlGrid.CellColumnMargin = 0;
  startStopControlGrid.FormControlTemplate.Frame3DEnable = flag3DBorder;
  chartVu.addChartObject(startStopControlGrid);
  let buttonlist2 = new Array();
  let attrib2 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 5,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.LIGHTGREEN);
  let pTransform2 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
  pTransform2.setGraphBorderDiagonal(0.21, .65, 0.4, 0.98);
  PrimaryLineButton.ButtonUncheckedText = "Primary Line";
  PrimaryLineButton.ButtonChecked = false;
  PrimaryLineButton.ButtonAreaEventListener = (e) => {
    selector_Button_Click(e);
  };
  chartVu.addMouseListener(PrimaryLineButton);
  PrimaryLineButton.ButtonFont = buttonfont;
  buttonlist2.push(PrimaryLineButton);
  SecondaryLineButton.ButtonUncheckedText = "Secondary Line";
  SecondaryLineButton.ButtonChecked = false;

```

```

SecondaryLineButton.ButtonAreaEventListener = (e) => {
    selector_Button_Click(e);
};
chartVu.addMouseListener(SecondaryLineButton);
SecondaryLineButton.ButtonFont = buttonfont;
buttonlist2.push(SecondaryLineButton);
Concurrent.ButtonUncheckedText = "Concurrent";
Concurrent.ButtonChecked = false;
Concurrent.ButtonAreaEventListener = (e) => {
    selector_Button_Click(e);
};
chartVu.addMouseListener(Concurrent);
Concurrent.ButtonFont = buttonfont;
buttonlist2.push(Concurrent);
numColumns = 1;
numRows = 4;
let controlgrid2 =
QCRTGraphTS.RTFormControlGrid.newRTFormControlGridCoordsCRAAttrib(pTransform2, buttonlist2,
numColumns, numRows, attrib2);
controlgrid2.CellRowMargin = 0.1;
controlgrid2.CellColumnMargin = 0;
controlgrid2.FormControlTemplate.Frame3DEnable = flag3DBorder;
chartVu.addChartObject(controlgrid2);
}

```

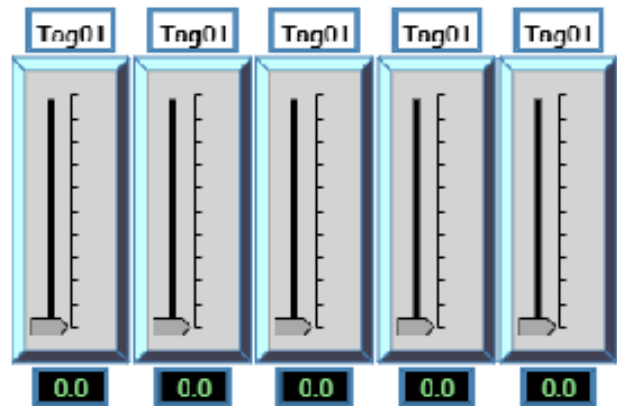
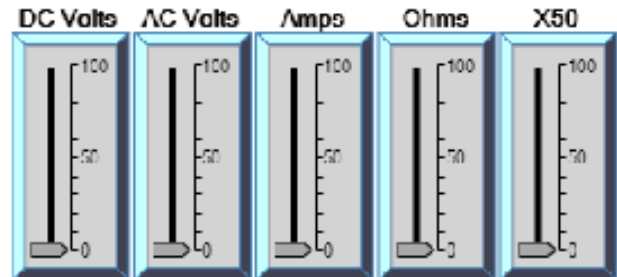
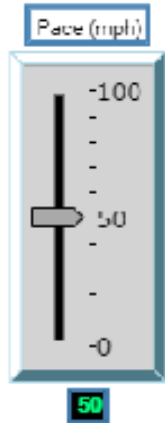
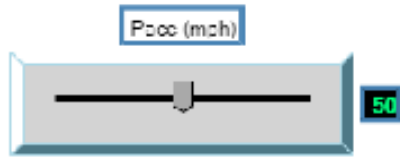
Control TrackBars

Class RTControlTrackBar

ScrollbarArea

RTControlTrackBar

The **RTControlTrackBar** class is subclassed from the **ScrollbarArea** class. It has a slightly different look, more like what you expect in a trackbar as opposed to a scroll bar.



RTControlButton constructor

```
public static newRTControlTrackBar(orientation: number): RTControlTrackBar

public static newRTControlTrackBarComponentOrient(orientation: number): RTControlTrackBar

public static newRTControlTrackBarOrientMinMax(orientation: number, scrollmin: number, scrollmax:
number): RTControlTrackBar

public static newRTControlTrackBarOrientMinMaxValueAttrib(orientation: number, scrollmin: number,
scrollmax: number, initialvalue: number, scrollareaattrib: ChartAttribute): RTControlTrackBar

public static newRTControlTrackBarOrientMinMaxValue(orientation: number, scrollmin: number,
scrollmax: number, initialvalue: number): RTControlTrackBar

public static newRTControlTrackBarOrientMinMaxValuePos(orientation: number, scrollmin: number,
scrollmax: number, initialvalue: number, pos: ChartRectangle2D): RTControlTrackBar

public static newRTControlTrackBarMinMaxLargeSmallTickFreq(minvalue: number, maxvalue: number,
largechange: number, smallchange: number, tickfrequency: number): RTControlTrackBar
```

Parameters

minvalue

Specifies the floating point minimum value for the track bar.

maxvalue

Specifies the floating point maximum value for the track bar.

largechange

Specifies the floating point large change value for the track bar. Not used at this time

smallchange

Specifies the floating point small change value for the track bar. Not used at this time

tickfrequency

Specifies the floating point tick frequency value for the track bar. Equivalent to the **TrackBar.TickFrequency** property, except allows floating point numbers.

Selected Public Instance Properties

RTOrientation	Gets or sets a value indicating the horizontal or vertical orientation (Orientation.Horizontal or Orientation.Vertical) of the track bar.
RTMaximum	Specifies the floating point maximum value for the track bar.
RTMinimum	Specifies the floating point minimum value for the track bar.
RTTickFrequency	Specifies the floating point tick frequency value for the track bar.
RTValue	Specifies the double value of the RTControlTrackBar slider.

A complete listing of **RTControlTrackBar** properties is found in the `_qcrgraphTS_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example for single **RTControlTrackBar** combined with an **RTNumericPanelMeter**

The example below, extracted from the Treadmill example, creates a single **RTControlTrackBar** and positions a large numeric readout of the trackbar value next to it.



[TypeScript]

```
public InitializeLeftPanelMeters() {
    let trackbarfont: QCRTGraphTS.ChartFont = this.font64Numeric;
    let trackbarTitlefont: QCRTGraphTS.ChartFont = this.font12Bold;
    if (!this.chartVu) return;
    let pTransform1: QCRTGraphTS.CartesianCoordinates =
        QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
```

252 Buttons, Track Bars and Other Form Control Classes

```
pTransform1.setGraphBorderDiagonal(0.01, .25, 0.05, 0.38);
let attrib1: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.LIGHTBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.LIGHTBLUE);
this.runnersPaceTrackbar =
QCRTGraphTS.RTControlTrackBar.newRTControlTrackBarOrientMinMaxValue(QCRTGraphTS.ChartConstants.VE
RT_DIR, 0, 15, 0.1);
this.runnersPaceTrackbar.Orientation = QCRTGraphTS.ChartConstants.VERT_DIR;
this.runnersPaceTrackbar.RTValue = 3;
this.chartVu.addMouseListener(this.runnersPaceTrackbar);
// MUST USE RTValue to set double value

let formControlTrackBar1: QCRTGraphTS.RTFormControlPanelMeter =
QCRTGraphTS.RTFormControlPanelMeter.newRTFormControlPanelMeterCoordsAttrib(pTransform1,
this.runnersPaceTrackbar, attrib1);
formControlTrackBar1.RTDataSource = this.runnersPace;
formControlTrackBar1.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
formControlTrackBar1.setLocationXYType(0.01, 0.3, QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);

//formControlTrackBar1.FormControlSize = QCRTGraphTS.ChartDimension.newChartDimension(1, 1);
// Must be in same units as SetLocation
let panelmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let panelmeter1: QCRTGraphTS.RTNumericPanelMeter =
QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(pTransform1, this.runnersPace,
panelmeterattrib);
panelmeter1.NumericTemplate.TextFont = trackbarfont;
panelmeter1.NumericTemplate.DecimalPos = 1;
panelmeter1.PanelMeterPosition = QCRTGraphTS.ChartConstants.RIGHT_REFERENCED_TEXT;
panelmeter1.setPositionReference(formControlTrackBar1);
formControlTrackBar1.addPanelMeter(panelmeter1);

let panelmetertagattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BEIGE, 0,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BEIGE);
let panelmeter2: QCRTGraphTS.RTStringPanelMeter =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(pTransform1,
this.runnersPace, panelmetertagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
panelmeter2.setPositionReference(panelmeter1);
panelmeter2.StringTemplate.TextFont = trackbarTitlefont;
panelmeter2.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
panelmeter2.TextColor = QCRTGraphTS.ChartColor.BLACK;
formControlTrackBar1.addPanelMeter(panelmeter2);
this.chartVu.addChartObject(formControlTrackBar1);

let pTransform2: QCRTGraphTS.CartesianCoordinates =
QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
pTransform2.setGraphBorderDiagonal(0.01, .75, 0.05, 0.88);
this.treadmillElevationTrackbar =
QCRTGraphTS.RTControlTrackBar.newRTControlTrackBarOrientMinMax(QCRTGraphTS.ChartConstants.VERT_DI
R, 0, 15);
this.treadmillElevationTrackbar.Orientation = QCRTGraphTS.ChartConstants.VERT_DIR;
this.treadmillElevationTrackbar.RTValue = 0;
this.chartVu.addMouseListener(this.treadmillElevationTrackbar);

let formControlTrackBar2: QCRTGraphTS.RTFormControlPanelMeter =
QCRTGraphTS.RTFormControlPanelMeter.newRTFormControlPanelMeterCoordsAttrib(pTransform2,
this.treadmillElevationTrackbar, attrib1);
formControlTrackBar2.RTDataSource = this.treadmillElevation;

formControlTrackBar2.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
formControlTrackBar2.setLocationXYType(0.01, 0.7, QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
```

```

let panelmeter3: QCRTGraphTS.RTNumericPanelMeter =
QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(pTransform2, this.runnersPace,
panelmeterattrib);
panelmeter3.NumericTemplate.TextFont = trackbarfont;
panelmeter3.NumericTemplate.DecimalPos = 1;
panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.RIGHT_REFERENCED_TEXT;
panelmeter3.setPositionReference(formControlTrackBar2);
formControlTrackBar2.addPanelMeter(panelmeter3);
let panelmeter4: QCRTGraphTS.RTStringPanelMeter =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(pTransform2,
this.runnersPace, panelmeterattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
panelmeter4.setPositionReference(panelmeter3);
panelmeter4.StringTemplate.TextFont = trackbarTitlefont;
panelmeter4.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
panelmeter4.TextColor = QCRTGraphTS.ChartColor.BLACK;
formControlTrackBar2.addPanelMeter(panelmeter4);
this.chartVu.addChartObject(formControlTrackBar2);
}

```

[JavaScript]

```

function InitializeLeftPanelMeters() {
    let trackbarfont = font64Numeric;
    let trackbarTitlefont = font12Bold;
    if (!chartVu)
        return;
    let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
    pTransform1.setGraphBorderDiagonal(0.01, .25, 0.05, 0.38);
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.LIGHTBL
UE, 3, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.LIGHTBLUE);
    runnersPaceTrackbar = QCRTGraphTS.RTControlTrackBar.newRTControlTrackBarOrientMinMaxValue
(QCRTGraphTS.ChartConstants.VERT_DIR, 0, 15, 0.1);
    runnersPaceTrackbar.Orientation = QCRTGraphTS.ChartConstants.VERT_DIR;
    runnersPaceTrackbar.RTValue = 3;
    chartVu.addMouseListener(runnersPaceTrackbar);
    // MUST USE RTValue to set double value
    let formControlTrackBar1 = QCRTGraphTS.RTFormControlPanelMeter.newRTFormControlPanelMeter
CoordsAttrib(pTransform1, runnersPaceTrackbar, attrib1);
    formControlTrackBar1.RTDataSource = runnersPace;
    formControlTrackBar1.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
    formControlTrackBar1.setLocationXYType(0.01, 0.3, QCRTGraphTS.ChartConstants.NORM_GRAPH_P
OS);
    //formControlTrackBar1.FormControlSize = QCRTGraphTS.ChartDimension.newChartDimension(1,
1);
    // Must be in same units as SetLocation
    let panelmeterattrib = QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartCol
or.STEELBLUE, 3, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let panelmeter1 = QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(pTransform1, run
nersPace, panelmeterattrib);
    panelmeter1.NumericTemplate.TextFont = trackbarfont;
    panelmeter1.NumericTemplate.DecimalPos = 1;
    panelmeter1.PanelMeterPosition = QCRTGraphTS.ChartConstants.RIGHT_REFERENCED_TEXT;
    panelmeter1.setPositionReference(formControlTrackBar1);
    formControlTrackBar1.addPanelMeter(panelmeter1);
    let panelmeterattrib = QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.Chart
Color.BEIGE, 0, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BEIGE);
    let panelmeter2 = QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAtt
ribStringType(pTransform1, runnersPace, panelmeterattrib, QCRTGraphTS.ChartConstants.RT_TAG_ST
RING);
    panelmeter2.setPositionReference(panelmeter1);
    panelmeter2.StringTemplate.TextFont = trackbarTitlefont;
    panelmeter2.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
    panelmeter2.TextColor = QCRTGraphTS.ChartColor.BLACK;
    formControlTrackBar1.addPanelMeter(panelmeter2);
    chartVu.addChartObject(formControlTrackBar1);
    let pTransform2 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
    pTransform2.setGraphBorderDiagonal(0.01, .75, 0.05, 0.88);
}

```

```

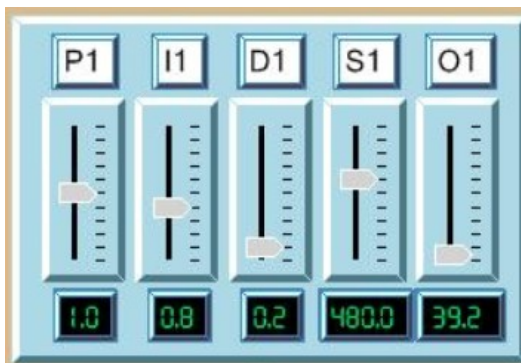
        treadmillElevationTrackbar =
            QCRTGraphTS.RTControlTrackBar.newRTControlTrackBarOrientMinMax(QCRTGraphTS.ChartConstants.VERT_DIR, 0, 15);
        treadmillElevationTrackbar.Orientation = QCRTGraphTS.ChartConstants.VERT_DIR;
        treadmillElevationTrackbar.RTValue = 0;
        chartVu.addMouseListener(treadmillElevationTrackbar);
        // MUST USE RTValue to set double value
        let formControlTrackBar2 = QCRTGraphTS.RTFormControlPanelMeter.newRTFormControlPanelMeter
            CoordsAttrib(pTransform2, treadmillElevationTrackbar, attrib1);
        formControlTrackBar2.RTDataSource = treadmillElevation;
        formControlTrackBar2.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
        formControlTrackBar2.setLocationXYType(0.01, 0.7, QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
        let panelmeter3 = QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeter(pTransform2, runnersPace, panelmeterattrib);
        panelmeter3.NumericTemplate.TextFont = trackbarfont;
        panelmeter3.NumericTemplate.DecimalPos = 1;
        panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.RIGHT_REFERENCED_TEXT;
        panelmeter3.setPositionReference(formControlTrackBar2);
        formControlTrackBar2.addPanelMeter(panelmeter3);
        let panelmeter4 = QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsProcessVarAttribStringType(pTransform2, runnersPace, panelmetertagattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
        panelmeter4.setPositionReference(panelmeter3);
        panelmeter4.StringTemplate.TextFont = trackbarTitlefont;
        panelmeter4.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
        panelmeter4.TextColor = QCRTGraphTS.ChartColor.BLACK;
        formControlTrackBar2.addPanelMeter(panelmeter4);
        chartVu.addChartObject(formControlTrackBar2);
    }

```

Example for multiple RTControlTrackBar controls in an RTFormControlGrid

The example below, extracted from the PIDControlTuner example creates four **RTControlTrackBar** controls. The trackbars are added to an **RTFormControlGrid** in order to position them as a logical group.

Note: If an **RTNumericPanelMeter** template is applied to the **RTControlTrackBar** controls in an **RTFormControlGrid**, they will all end up with the same number of digits to the right of the decimal, since one template applies to all of the track bars. If the dynamic range of the track bars different enough to require unique decimal precision settings, separate them into different grids.



See the method `InitializePIDParameterTrackbars()` in the `RTPIDControlTuner` example for the source to this example.

Form Control Panel Meter

This panel meter class encapsulates the **RTControlButton** and **RTControlTrackBar** objects in a panel meter class, so that controls can be added to indicator objects.

Class **RTFormControlPanelMeter**

RTPanelMeter

RTFormControlPanelMeter

```
public static newRTFormControlPanelMeterCoords(transform: PhysicalCoordinates):
RTFormControlPanelMeter

public static newRTFormControlPanelMeterCoordsAttrib(transform: PhysicalCoordinates, formcontrol:
MouseListener, attrib: ChartAttribute): RTFormControlPanelMeter

public static newRTFormControlPanelMeter(transform: PhysicalCoordinates, datasource:
RTProcessVar, formcontrol: MouseListener, attrib: ChartAttribute): RTFormControlPanelMeter

public static newRTFormControlPanelMeterArray(transform: PhysicalCoordinates, formcontrolarray:
MouseListener[], attrib: ChartAttribute): RTFormControlPanelMeter
```

Parameters

transform

The coordinate system for the new **RTFormControlPanelMeter** object.

datasource

The process variable associated with the control.

formcontrol

A reference to the Control assigned to this panel meter.

attrib

The color attributes of the panel meter.

Selected Public Instance Properties

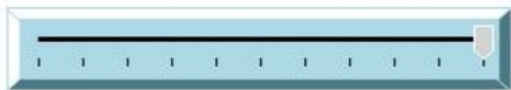
ChartObjAttributes (inherited from GraphObj)	Sets the attributes for a chart object using a ChartAttribute object.
ChartObjScale (inherited from GraphObj)	Sets the reference to the PhysicalCoordinates object that the chart object is placed in
ControlSizeMode	Set/Get to the size mode for the Control. Use one of the Control size mode constants:

	RT_ORIG_CONTROL_SIZE, RT_MIN_CONTROL_SIZE, RT_INDICATORRECT_CONTROL_SIZE.
FormControlSize	Get the size of the form control in device units.
Frame3DEnable (inherited from RTPanelMeter)	Set/Get to true to enable a 3D frame for the panel meter.
IndicatorRect (inherited from RTPanelMeter)	Get/Set Indicator positioning rect.
PanelMeterNudge (inherited from RTPanelMeter)	Set/Get the xy values of the panelMeterNudge property. It moves the relative position, using window device coordinates, of the text relative to the specified location of the text.
PanelMeterPosition (inherited from RTPanelMeter)	Set/Get the panel meter position value. See the panel meter position table in the RTPanelMeter chapter.
PanelMeterRectangle (inherited from RTPanelMeter)	Set/Get the panel meter bounding rectangle.
PanelMeterTemplate (inherited from RTPanelMeter)	Get a ChartLabel object representing the panel meters template.
PositionReference (inherited from RTPanelMeter)	Set/Get an RTPanelMeter object used as a positioning reference for this RTPanelMeter object.
PositionType (inherited from GraphObj)	Get/Sets the current position type.
PrimaryChannel (inherited from RTPlot)	Set/Get the primary channel of the indicator.
RTDataSource (inherited from RTSingleValueIndicator)	Get/Set the array list holding the RTProcessVar variables for the indicator.

A complete listing of **RTFormControlPanelMeter** properties is found in the _qcrgraphTS_.html documentation file, located in the QCRTGraphTS\docs\modules subdirectory.

Example for RTControlPanelMeter encapsulating an RTControlTrackBar

The example below, extracted from the Polygraph example, creates an **RTControlPanelMeter** using an **RTControlTrackBar**.



[TypeScript]

```
timeAxisControlTrackbar: QCRTGraphTS.RTControlTrackBar = new QCRTGraphTS.RTControlTrackBar();
```

```
:
```

```

    public InitializeTimeAxisTrackbar() {
        let attrib1: QCRTGraphTS.ChartAttribute =
        QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.WHITE, 3,
        QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.CORAL);
        if (!this.chartVu) return;
        let pTransform2: QCRTGraphTS.CartesianCoordinates =
        QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
        pTransform2.setGraphBorderDiagonal(0.7, 0.93, 0.98, 0.99);
        let tbattrib: QCRTGraphTS.ChartAttribute =
        QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.LIGHTBLUE, 7,
        QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.LIGHTBLUE);
        let starttime: number = this.origStartTime.getTime();
        let endtime: number = this.origEndTime.getTime();
        let range: number = (endtime - starttime);
        this.timeAxisControlTrackbar =
        QCRTGraphTS.RTControlTrackBar.newRTControlTrackBarMinMaxLargeSmallTickFreq(0, 100, 10, 1, 10);
        this.timeAxisControlTrackbar.Orientation = QCRTGraphTS.ChartConstants.HORIZ_DIR;
        this.timeAxisControlTrackbar.TrackBarEventListener = (e: QCRTGraphTS.ScrollbarAreaEventArgs)
=> {
            this.timeAxisControlTrackbar_Click(e);
        }
        this.timeAxisControlTrackbar.RTValue = 100;
        this.chartVu.addMouseListener(this.timeAxisControlTrackbar);

        // MUST USE RTValue to set double value
        let timeAxisControlPanelTrackBar: QCRTGraphTS.RTFormControlPanelMeter =
        QCRTGraphTS.RTFormControlPanelMeter.newRTFormControlPanelMeterCoordsAttrib(pTransform2,
        this.timeAxisControlTrackbar, tbattrib);
        timeAxisControlPanelTrackBar.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
        timeAxisControlPanelTrackBar.setLocation(0, 0);
        timeAxisControlPanelTrackBar.FormControlSize = QCRTGraphTS.ChartDimension.newChartDimension(1,
        1);
        this.chartVu.addChartObject(timeAxisControlPanelTrackBar);
    }

```

[JavaScript]

```

timeAxisControlTrackbar = new QCRTGraphTS.RTControlTrackBar();
.
.
.

function InitializeTimeAxisTrackbar() {
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.WHITE, 3,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.CORAL);
    if (!chartVu) return;
    let pTransform2 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
    pTransform2.setGraphBorderDiagonal(0.7, 0.93, 0.98, 0.99);
    let tbattrib = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.LIGHTBLUE,
    7, QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.LIGHTBLUE);
    let starttime = origStartTime.getTime();
    let endtime = origEndTime.getTime();
    let range = (endtime - starttime);
    timeAxisControlTrackbar =
    QCRTGraphTS.RTControlTrackBar.newRTControlTrackBarMinMaxLargeSmallTickFreq(0, 100, 10, 1, 10);
    timeAxisControlTrackbar.Orientation = QCRTGraphTS.ChartConstants.HORIZ_DIR;
    timeAxisControlTrackbar.TrackBarEventListener = (e) => {
        timeAxisControlTrackbar_Click(e);
    };
    timeAxisControlTrackbar.RTValue = 100;
    chartVu.addMouseListener(timeAxisControlTrackbar);
    // MUST USE RTValue to set double value
    let timeAxisControlPanelTrackBar =
    QCRTGraphTS.RTFormControlPanelMeter.newRTFormControlPanelMeterCoordsAttrib(pTransform2,
    timeAxisControlTrackbar, tbattrib);
    timeAxisControlPanelTrackBar.PanelMeterPosition = QCRTGraphTS.ChartConstants.CUSTOM_POSITION;
    timeAxisControlPanelTrackBar.setLocation(0, 0);
}

```

```
timeAxisControlPanelTrackBar.FormControlSize = QCRTGraphTS.ChartDimension.newChartDimension(1,
1);
chartVu.addChartObject(timeAxisControlPanelTrackBar);
}
```

Form Control Grid

The **RTFormControlGrid** organizes a collection of **Form Control** objects functionally and visually in a grid format.

Class RTFormControlGrid

RTMultiValueIndicator RTFormControlGrid

RTFormControlGrid constructors

```
public static newRTFormControlGridCoords(transform: PhysicalCoordinates): RTFormControlGrid

public static newRTFormControlGrid(transform: PhysicalCoordinates, datasource: RTProcessVar[],
formcontrolarray: MouseListener[],attrib: ChartAttribute): RTFormControlGrid

public static newRTFormControlGridCRAttrib(transform: PhysicalCoordinates, datasource:
RTProcessVar[],
formcontrolarray: MouseListener[], numcols: number, numrows: number, attrib: ChartAttribute) :
RTFormControlGrid

public static newRTFormControlGridCoordsCRAttrib(transform: PhysicalCoordinates,
formcontrolarray: MouseListener[], numcols: number, numrows: number, attrib: ChartAttribute):
RTFormControlGrid

public static newRTFormControlGridCoordsCRStringsAttrib(transform: PhysicalCoordinates,
formcontrolarray: MouseListener[],
numcols: number, numrows: number, colheads: string[], rowheads: string[], attrib:
ChartAttribute): RTFormControlGrid

public static newRTFormControlGridCRStringsAttrib(transform: PhysicalCoordinates, datasource:
RTProcessVar[],
formcontrolarray: MouseListener[],numcols: number, numrows: number, colheads: string[], rowheads:
string[], attrib: ChartAttribute): RTFormControlGrid
```

Parameters

transform

The coordinate system for the new **RTFormControlGrid** object.

datasource

An array of the process variables associated with the control grid objects.

formcontrolarray

An array of the Controls assigned to the control grid.

numcols

The number of columns in the control grid.

numrows

The number of rows in the control grid.

colheads

An array of string that is used as the column heads for the control grid.

rowheads

An array of string that is used as the row heads for the control grid.

attrib

A single attribute object that applies to all control grid objects.

Selected Public Instance Properties

CellColumnMargin	Get/Set the extra space between columns of the grid, specified in normalized NORM_PLOT_POS coordinates.
CellRowMargin	Get/Set the extra space between rows of the grid, specified in normalized NORM_PLOT_POS coordinates.
ChartObjAttributes (inherited from GraphObj)	Sets the attributes for a chart object using a ChartAttribute object.
ChartObjScale (inherited from GraphObj)	Sets the reference to the PhysicalCoordinates object that the chart object is placed in
ColumnHeaders	Set/Get the column headers.
FormControlTemplate	Set/Get the row headers.
HeadersTemplate	Set/Get the string template.
InternalAction	Set/Get to true if you want radio button click and scroll bar value changed events processed to update colors of buttons and numeric values of scroll bars.
NumberColumns	Get the number of rows in the annunciator.
NumberRows	Get the number of rows in the annunciator.
NumChannels (inherited from RTPlot)	Get the number of channels in the indicator.
PanelMeterList (inherited from RTPlot)	Set/Get the panel meter list of the RT Plot.
PositionType (inherited from GraphObj)	Get/Sets the current position type.
RadioButtonChecked	Get/Set the extra space between columns of the grid, specified in normalized NORM_PLOT_POS coordinates.
RowHeaders	Set/Get the row headers.
RTDataSource (inherited from RTMultiValueIndicator)	Get/Set the array list holding the RTProcessVar variables for the indicator.

A complete listing of **RTFormControlGrid** properties is found in the `_qcrtrgraphs_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example for RTFormControlGrid encapsulation RTControlButton objects.

The example below, extracted from the MiniScope example, creates an **RTFormControlGrid** using a collection of **RTControlButtons**.

Frequency	10	100	1K	10K
Ohms	10	100	1K	10K
Capacitance	10u	100u	1m	10m
DC Volts	0.1	1	10	100
AC Volts	0.1	1	10	100
DC Amps	0.1	1	10	100
AC Amps	0.1	1	10	100

[TypeScript]

```

rangeSelectorButtons: QCRTGraphTS.RTControlButton[] = new Array<QCRTGraphTS.RTControlButton>();
.
.
.
public InitializeRangeSelectorButtons() {
    if (!this.chartVu) return;

    let selectorStrings: string[] = [
        "10", "100", "1K", "10K",
        "10", "100", "1K", "10K",
        "10u", "100u", "1m", "10m",
        "0.1", "1", "10", "100",
        "0.1", "1", "10", "100",
        "0.1", "1", "10", "100",
        "0.1", "1", "10", "100"];
    let rowStrings: string[] = [
        "Frequency",
        "Ohms",
        "Capacitance",
        "DC Volts",
        "AC Volts",
        "DC Amps",
        "AC Amps"];
    let colStrings: string[] = [
        "",
        "",
        "",
        ""];
    let rtbutton: QCRTGraphTS.RTControlButton;

    let buttonfont: QCRTGraphTS.ChartFont = this.font12Bold;
    let pTransform1: QCRTGraphTS.CartesianCoordinates =
        QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
    pTransform1.setGraphBorderDiagonal(0.25, .68, 0.95, 0.97);
    let attrib1: QCRTGraphTS.ChartAttribute =
        QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.WHITE, 3,
        QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.SANDYBROWN);
    for (let i: number = 0; i < selectorStrings.length; i++) {

```

```

        rtbutton =
QCRTGraphTS.RTControlButton.newRTControlButton(QCRTGraphTS.ChartConstants.RT_CONTROL_RADIOBUTTON_
SUBTYPE);
        rtbutton.ButtonUncheckedText = selectorStrings[i];
        if ((i == this.currentRangeSelectorIndex)) {
            rtbutton.ButtonChecked = true;
        }
        else {
            rtbutton.ButtonChecked = false;
        }

        rtbutton.ButtonAreaEventListener = (e: QCRTGraphTS.ButtonAreaEventArgs) => {
            this.selector_button_Click(e);
        }
        this.chartVu.addMouseListener(rtbutton);

        rtbutton.ButtonFont = buttonfont;
        this.rangeSelectorButtons.push(rtbutton);

    }
    // set the first button with associates for the radio buttons
    this.rangeSelectorButtons[0].setButtonGroupAssociates(this.rangeSelectorButtons);

    let numColumns: number = 4;
    let numRows: number = 7;
    let controlgrid: QCRTGraphTS.RTFormControlGrid =
QCRTGraphTS.RTFormControlGrid.newRTFormControlGridCoordsCRStringsAttrib(pTransform1,
this.rangeSelectorButtons, numColumns, numRows, colStrings, rowStrings, attrib1);
    controlgrid.CellRowMargin = 0.05;
    controlgrid.CellColumnMargin = 0.1;
    controlgrid.FormControlTemplate.Frame3DEnable = true;
    controlgrid.HeadersTemplate.LineColor = QCRTGraphTS.ChartColor.BLACK;
    controlgrid.HeadersTemplate.TextFont = this.font14Bold;
    this.chartVu.addChartObject(controlgrid);
}

```

[JavaScript]

```

function InitializeRangeSelectorButtons() {
    if (!chartVu) return;
    let selectorStrings = [
        "10", "100", "1K", "10K",
        "10", "100", "1K", "10K",
        "10u", "100u", "1m", "10m",
        "0.1", "1", "10", "100",
        "0.1", "1", "10", "100",
        "0.1", "1", "10", "100",
        "0.1", "1", "10", "100"
    ];
    let rowStrings = [
        "Frequency",
        "Ohms",
        "Capacitance",
        "DC Volts",
        "AC Volts",
        "DC Amps",
        "AC Amps"
    ];
    let colStrings = [
        "",
        "",
        "",
        ""
    ];
    let rtbutton;
    let buttonfont = font12Bold;
}

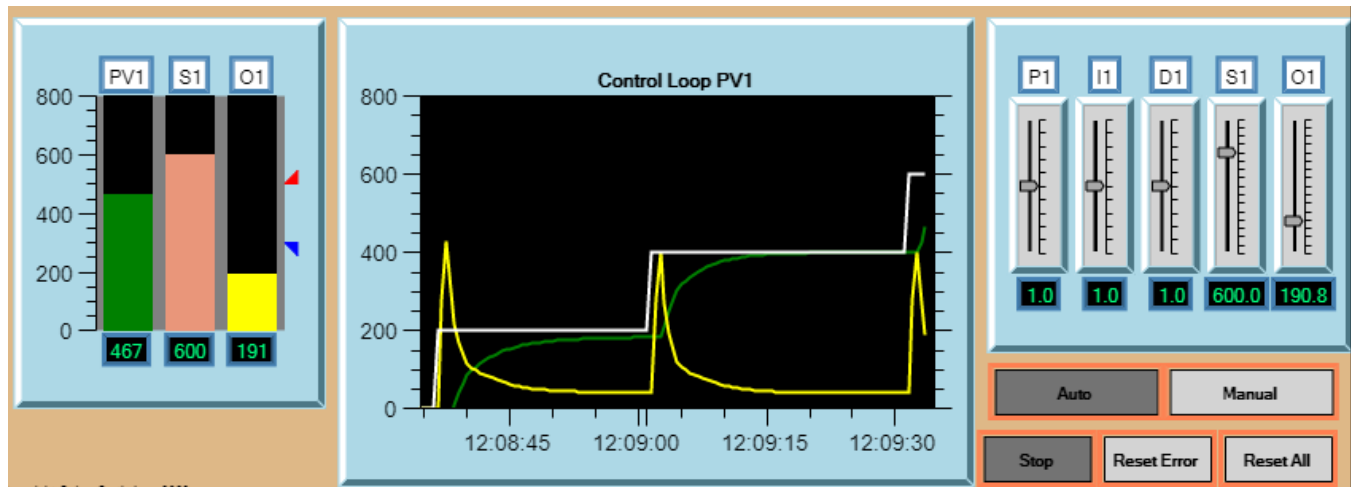
```

```

    let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, 0, 1, 1);
    pTransform1.setGraphBorderDiagonal(0.25, .68, 0.95, 0.97);
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.WHITE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.SANDYBROWN);
    for (let i = 0; (i < selectorStrings.length); i++) {
        rtbutton =
QCRTGraphTS.RTControlButton.newRTControlButton(QCRTGraphTS.ChartConstants.RT_CONTROL_RADIOBUTTON_
SUBTYPE);
        rtbutton.ButtonUncheckedText = selectorStrings[i];
        if ((i == currentRangeSelectorIndex)) {
            rtbutton.ButtonChecked = true;
        }
        else {
            rtbutton.ButtonChecked = false;
        }
        rtbutton.ButtonAreaEventListener = (e) => {
            selector_button_Click(e);
        };
        chartVu.addMouseListener(rtbutton);
        rtbutton.ButtonFont = buttonfont;
        rangeSelectorButtons.push(rtbutton);
    }
    // set the first button with associates for the radio buttons
    rangeSelectorButtons[0].setButtonGroupAssociates(rangeSelectorButtons);
    let numColumns = 4;
    let numRows = 7;
    let controlgrid =
QCRTGraphTS.RTFormControlGrid.newRTFormControlGridCoordsCRStringsAttrib(pTransform1,
rangeSelectorButtons, numColumns, numRows, colStrings, rowStrings, attrib1);
    controlgrid.CellRowMargin = 0.05;
    controlgrid.CellColumnMargin = 0.1;
    controlgrid.FormControlTemplate.Frame3DEnable = true;
    controlgrid.HeadersTemplate.LineColor = QCRTGraphTS.ChartColor.BLACK;
    controlgrid.HeadersTemplate.TextFont = font14Bold;
    chartVu.addChartObject(controlgrid);
}

```

15. PID Control



Theory

Proportional-Integral-Derivative (PID) control algorithm is used to drive the process variable (measurement) to the preset value (setpoint).

Temperature control is the most common form of closed loop control. For example, in a simple temperature control system the temperature of a vat of material is to be maintained at a given setpoint, $s(t)$. The output of the controller sets the valve of the actuator to apply less heat to the vat if the current temperature of the vat is greater than the setpoint and more heat to the vat if the current temperature is less than the setpoint.

The PID algorithm calculates its output by summing three terms. One term is proportional to the error (error is defined as the setpoint minus the current measured value). The second term is proportional to the integral of the error over time, and the third term is proportional to the rate of change (first derivative) of the error. The general form of the PID control equation in analog form is:

Eqn. 1

$$m(t) = K_c * (\underset{\text{proportional}}{e(t)} + K_i \underset{\text{integral}}{\int e(t)dt} + K_d \underset{\text{derivative}}{\frac{de}{dt}})$$

where:

$m(t)$ = controller output deviation

K_c = proportional gain

K_i = reset multiplier (integral time constant)

K_d = derivative time constant

$S(t)$ = current process setpoint

$X(t)$ = actual process measured variable (temperature, for example)

$e(t)$ = error as a function of time = $S(t) - X(t)$

The variables K_c , K_i and K_d are adjustable and are used to customize a controller for a given process control application. The K_i constant term is listed in some textbooks as $(1/K_i)$. It is simply a matter of the units K_i is specified in. In the K_i form, the units are repeats per minute while in the $1/K_i$ form the units are minutes per repeat (also called reciprocal time). The K_i version presented here is preferred because increasing K_i will increase the integral gain action, just like increasing K_c and K_d will increase the proportional gain and derivative gain action. If $1/K_i$ is used, then decreasing values of K_i will increase the amount of integral gain.

The proportional term of the PID equation contributes an amount to the controller output directly proportional to the current process error. For example, if the setpoint of the process is 100 degrees and the current temperature of the process is 90 degrees, then the current process error is 10 degrees. The proportional term adds to the controller output an amount equal to $K_c * 10$. The gain term K_c determines how much the control output should change in response to a given error level. If the error is 10, a K_c gain of 0.5 will add 5 to the output of the controller, while a gain of 3 will add 30 to the output of the controller. The larger the value of K_c , the harder the system reacts to differences between the setpoint and the actual temperature. A PID controller can be used as a simple proportional controller by setting the reset rate and derivative time values to 0.0.

Simple proportional control cannot take into account load changes in the process under control. An example of a load for a temperature control loop is the ambient temperature of the room the process is in. The lower the ambient temperature of the room, the larger is the heat loss in the room. It will take more energy to maintain the vat at a given temperature in a cold room than in a warm room. A simple proportional controller cannot account for load changes which take place while the system is under control. Integral control converts the first-order proportional controller into a second order system which is capable of tracking process disturbances. It adds to the controller output a term that corrects for any changes in the load level of the system under control. This integral term is proportional to the sum of all previous process errors in the system. As long as there is a process error, the integral term will add more and more to the controller output until the sum of all previous errors is zero.

The term 'reset rate' is used to describe the integral action of a PID controller. The term derives from the output response of a PI controller (the derivative term set to zero in this case) to a step change in the process error signal. The response consists of an initial jump from zero to the proportional gain setting K_c , followed by a ramp (the integrating action of the integral term) which adds the initial proportional response each integral time T . Therefore the reset rate is defined as the repeats per minute of the initial proportional response.

For example, if the reset rate is 1.0, then for every minute that the error signal is non-zero, the amount of corrective action added to the controller output by the integral term will be equal to the amount added by the proportional term alone. The higher the reset rate, the harder the system will react to non zero error terms.

The addition of the derivative term to the PI controller described above results in the classic three mode PID controller. The derivative action of a PID controller adds to the controller output the value proportional to the slope (rate of change) of the process error. The derivative term "anticipates" the error, providing a harder control response when the error term is going in the wrong direction and a dampening response when the error term is going in the right direction. It reduces overshoot for transient upsets. Proper use of the derivative term can result in much faster process response.

Computer based versions of the PID algorithm are based on sampled data discrete time control theory. The discrete time equivalent of the PID equation is:

Eqn. 2

$$m(i) = K_c * (e(i) + T * K_i \sum_{k=0}^i e(k) + (k_d/T) * (e(i)-e(i-1)))$$

where

T = sampling interval

e(i) = error at ith sampling interval = S(t) -X(t)

e(i-1) = error at previous sampling interval

m(i) = controller output deviation

K_c = proportional gain

K_i = integral action time

K_d = derivative action time

The proportional term is the same between the Eqn. 1 and Eqn. 2. The integral term of the first equation is replaced by a summation term and the derivative term is replaced by the a first order difference approximation. In actual practice, the first order difference term, (e_i - e_{i-1}), is very susceptible to noise problems. In most practical systems this term is replaced by the more stable, higher order equation:

$$\Delta e = (e(i) + 3 * e(i-1) - 3 * e(i-2) - e(i-3))/6$$

A common problem in discrete control systems arises from the summation of the error term for the integral action of the control equation. If a process maintains an error for a long period of time, it is possible that this summation can build to a very large numerical value. Even though the error term returns to zero or moves in the opposite direction, it will take a very long time to reduce the sum below the D/A saturation levels. Practical systems stop the summation of error terms if the current PID output

level is outside a user specified range of high and low output values. This limiting of the summation term is commonly referred to as anti-reset-windup.

Implementation

Real-Time Graphics Tools can maintain an unlimited number of control loops simultaneously; the only limit being memory and CPU power. A PID control object (the terms *PID controller* and *PID object* are used interchangeably in this documentation) is created and configured using the RTPID class. The **RTCalcPID** function calculates the PID algorithm's output. It should be called at equal time intervals. PID algorithm constants can be tuned by adjusting corresponding property values.

A typical problem occurs when a PID object is switched from manual to automatic mode or when a PID constant is changed: the output value can change very quickly, possibly damaging the control equipment. The Quinn-Curtis implementation of the PID algorithm uses the "bumpless transfer" technique to prevent this problem. The algorithm also uses *anti-reset-windup* technique.

PID Control

Class RTPIDControl

RTPIDControl constructors

```
public static newRTPIDControlFull(setpnt: number, steadstat: number, prop: number, integ: number,
deriv: number, lowclmp: number, highclmp: number, rateclmp: number, sampleper: number,
filterconst: number): RTPIDControl
```

```
public static newRTPIDControl(setpnt: number, steadstat: number, prop: number, integ: number,
deriv: number, sampleper: number, filterconst: number): RTPIDControl
```

Parameters

setpnt

Specifies the desired value for the process variable.

steadstat

Anticipated steady state value for the output, also known as bias. If you do not know the steady state value, use 0.0 for this parameter. Setting this value properly improves response because it does not have to rely on integral response, starting with a zero initial error summation term, to add enough to the control output to make up for system losses.

prop

Specifies the proportional gain constant. The proportional term adjust the output value proportional to the current error term.

integ

Specifies the integral gain constant. The integral term adjusts the output value by accumulating, or integrated the error term over time.

deriv

Specifies the derivative gain constant. The derivative term looks at the rate of change of the input and adjusts the output based on the rate of change. The derivative function uses the time derivative of the error term.

lowclmp

Specifies the low clamping value for output. If the output of the PID calculation results in a value less than *lowclmp*, the value will be clamped to *lowclmp*.

highclmp

Specifies the high clamping value for output. If the output of the PID calculation results in a value higher than *highclmp*, the value will be clamped to *highclmp*.

rateclmp

Clamping limit for the output rate of change, measured in output units per minute. It limits the rate of change of the algorithm output.

sampleper

Sample period of PID updates, in minutes. For example, if the controller's output is calculated two times a second, the value of this parameter is $1 / (2 * 60) = 0.0084$ minutes.

filterconst

A value in the range 0.0 to 1.0, affecting the filtering of the noisy measurement signal. A value of 0.0 means that no filtering takes place. The filtering effect is maximal when *rFiltConst* is 1.0. The formula for filtering is:

Filtered value = $(1.0 - rFiltConst) * \text{Measured value} + rFiltConst * (\text{Previous filtered value})$

Selected Public Instance Properties

DerivativeConstant	Set/Get derivative constant value
E1	Set/Get error term t-1.
E2	Set/Get error term t-2.
E3	Set/Get error term t-3.
HighClamp	Set/Get high clamping value for output .
IntegralConstant	Set/Get integral constant value
LastMv	Set/Get previous exponential smoothing constant.
LastPIDValue	Set/Get previous PID output value.
LowClamp	Set/Get low clamping value for output .
MvFilter	Set/Get exponential smoothing constant.
NewError	Set/Get new error value.
OldError	Set/Get previous error value.

ProportionalConstant	Set/Get proportional constant value
RateClamp	Set/Get rate (first derivative of output) clamping value for output .
SamplePeriod	Set/Get time between adjacent updates.
SetPoint	Set/Get setpoint value.
SteadyState	Set/Get steady state output position.
SumError	Set/Get sum of all previous errors .

Selected Public Instance Methods

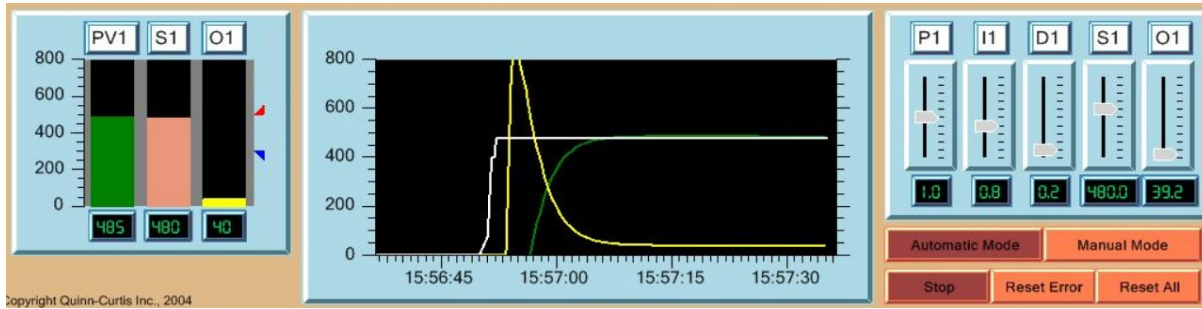
RTCalcPID	This method performs a PID loop calculation.
RTResetErrorTerms	This method resets all of the error terms for the PID calculations.
UpdatePIDIntermediateParameters	This method updates the intermediate values in the PID calculation.

A complete listing of **RTPIDControl** properties is found in the `_qcrgraphts_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

The output of the PID equation, calculated using the **RTCalcPID** method, is in the same units as the measured variable. If the measured variable is temperature with a potential range of 0-300, then the output of the PID equation will also be temperature in the same range, though it can have even wider swings than the measured variable. The output of the PID equation is expected to drive some sort of control device, either an actuator, heater, pressure control valve, or dc servomotor, which has completely different units than the control output. It is up to the control engineer to calculate the transfer function, usually a basic $mx + b$ equation, to the control output so that it matches the input range of the control device, whether it be 0-10V, 4-20mA or some other input range. This is completely dependent on the application, and resolving this final stage transfer function is entirely up to a control engineer and not part of this software.

Example for RTPIDControl.

The example below is extracted from the `PIDControlTuner` example. The code really needs to be studied in the context of the complete program so study that example program instead of the abbreviated code below.



[TypeScript]

```
PIDProcessItems: QCRTGraphTS.RTProcessVar[] = new Array<QCRTGraphTS.RTProcessVar>(16);
ProportionalControl: QCRTGraphTS.RTProcessVar[] = new Array<QCRTGraphTS.RTProcessVar>(16);
IntegralControl: QCRTGraphTS.RTProcessVar[] = new Array<QCRTGraphTS.RTProcessVar>(16);
DerivativeControl: QCRTGraphTS.RTProcessVar[] = new Array<QCRTGraphTS.RTProcessVar>(16);
ControlSetpoints: QCRTGraphTS.RTProcessVar[] = new Array<QCRTGraphTS.RTProcessVar>(16);
ControlTrackBarOutputs: QCRTGraphTS.RTProcessVar[] = new Array<QCRTGraphTS.RTProcessVar>(16);
ControlOutputs: QCRTGraphTS.RTProcessVar[] = new Array<QCRTGraphTS.RTProcessVar>(16);
PIDControlLoops: QCRTGraphTS.RTPIDControl[] = new Array<QCRTGraphTS.RTPIDControl>(16);

RTProcessVar []ControlTrackBarOutputs= new RTProcessVar[16];
RTProcessVar []ControlOutputs= new RTProcessVar[16];

.
.
for (let i: number = 0; (i < this.numChannels); i++) {
.
.
.

this.PIDControlLoops[i] =
QCRTGraphTS.RTPIDControl.newRTPIDControl(this.ControlSetpoints[i].CurrentValue, pidSteadyState,
this.ProportionalControl[i].CurrentValue, this.IntegralControl[i].CurrentValue,
(this.DerivativeControl[i].CurrentValue / 100), (sampleper / 60), filterConstant);

.
.
.
```

```

    CalculatePIDValues() {
    let rMeas: number = 0;
    let rSetpoint: number = 0;
    let rOutput: number = 0;
    for (let i: number = 0; (i < 8); i++) {
// Not calculating the PID value will prevent integral errors from continuing to be added to
error sum
    // simulate measurement
    rOutput = this.ControlOutputs[i].CurrentValue;
    rMeas = this.ProcessModel(i, rOutput);
    this.PIDProcessItems[i].setCurrentValueNumber(rMeas);
    if (this.autoModeEnable[i]) {
    rSetpoint = this.ControlSetpoints[i].CurrentValue;
    rOutput = this.PIDControlLoops[i].rTCalcPID(rMeas, rSetpoint);
    this.ControlOutputs[i].setCurrentValueNumber(rOutput);
    }

    }

    this.outputControlTrackBar.RTValue =
this.ControlOutputs[this.currentTuningChannel].CurrentValue;
}

```

[JavaScript]

```

var PIDProcessItems = new Array(16);
var ProportionalControl = new Array(16);
var IntegralControl = new Array(16);
var DerivativeControl = new Array(16);
var ControlSetpoints = new Array(16);
var ControlTrackBarOutputs = new Array(16);
var ControlOutputs = new Array(16);
var PIDControlLoops = new Array(16);

.
.
.
for (let i = 0; (i < numChannels); i++) {
.
.
.

    PIDControlLoops[i] =
QCRTGraphTS.RTPIDControl.newRTPIDControl(ControlSetpoints[i].CurrentValue, pidSteadyState,
ProportionalControl[i].CurrentValue, IntegralControl[i].CurrentValue,
(DerivativeControl[i].CurrentValue / 100), (sampleper / 60), filterConstant);
}
.
.
.

function CalculatePIDValues() {
    let rMeas = 0;
    let rSetpoint = 0;
    let rOutput = 0;
    for (let i = 0; (i < 8); i++) {
    // Not calculating the PID value will prevent integral errors from continuing to be added
to error sum
    // simulate measurement
    rOutput = ControlOutputs[i].CurrentValue;
    rMeas = ProcessModel(i, rOutput);
    PIDProcessItems[i].setCurrentValueNumber(rMeas);
    if (autoModeEnable[i]) {
    rSetpoint = ControlSetpoints[i].CurrentValue;
    rOutput = PIDControlLoops[i].rTCalcPID(rMeas, rSetpoint);
    ControlOutputs[i].setCurrentValueNumber(rOutput);
    }
    }

    outputControlTrackBar.RTValue = ControlOutputs[currentTuningChannel].CurrentValue;
}

```

}

16. Zooming Real-Time Data

ChartZoom

Zooming is the interactive re-scaling of a chart's 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.

The **ChartZoom** class is used for zooming of **RTProcessVar** historical data. The features of this class 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.
- User-definable zoom limits prevent numeric under and overflows

The **Real-Time Graphics Tools for JavaScript/TypeScript** software was designed to allow zooming of real-time data, while the data is being collected. Real-time data values are updated, using the **RTProcessVar** class, asynchronous to the update of the screen. The current graphics display can be historical data from the same **RTProcessVar** object, or it can be based on an entirely different **RTProcessVar** object. It is therefore possible to zoom back to the beginning of an **RTProcessVar** object's historical buffer, without affecting current data collection. At any time the graph returns to a view that includes the most current information.

When you want to zoom or pan backwards into the historical buffer of the **RTProcessVar** object, first you must disable the **RTScrollFrame** updates. Since the **RTScrollFrame** will attempt to always graph the most recent data values, you don't want it interfering with a zoom or a pan which explicitly does NOT want the most recent values displayed. Disable the **RTScrollFrame** updates using the **RTScrollFrame.ChartObjEnable** method: set it to **ChartObj.OBJECT_DISABLE**. When you want to start scrolling again set it to **ChartConstants.OBJECT_ENABLE**.

Simple Zooming of a single channel scroll frame

Class ChartZoom

GraphObj

|
+--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 four 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 from 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
```

<i>component</i>	A reference to the ChartView object that the chart is placed in.
<i>transform</i>	The PhysicalCoordinates object associated with the scale being zoomed.
<i>brescale</i>	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.setEnabled(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 to its original condition, after which the popZoomStack method has no effect.

Integrated zoom stack processing

.

```
zoomObj.InternalZoomStackProcessing = 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 initialized.

```
zoomObj.ArCorrectionMode = QCRTGraphTS.ChartConstants.ZOOM_X_AR_CORRECTION
```

Integrated zoom stack processing

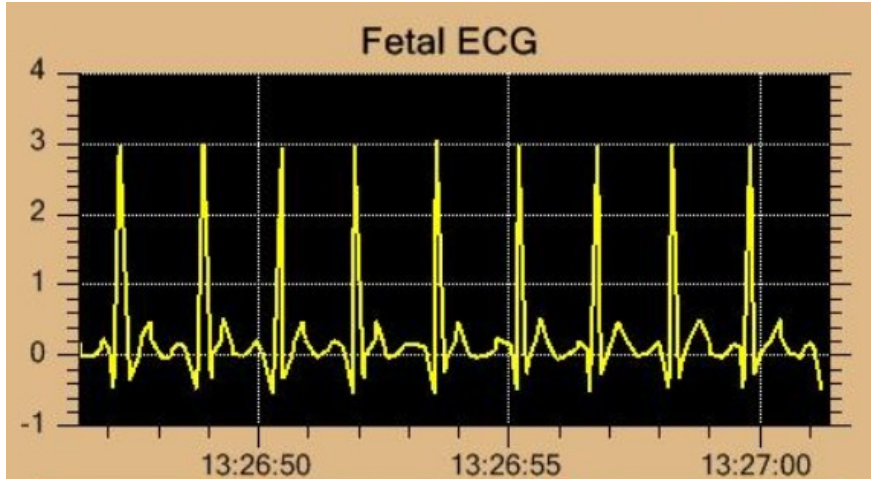
Zoom stack processing is internal to **ChartZoom** class. There is no need to subclass the **ChartZoom** class in order to implement a zoom stack. Just set the ChartZoom.InternalZoomStackProcessing property true.

```
zoomObj.InternalZoomStackProcessing = true;
```

Return to a previous zoom level by clicking the mouse.

Simple zoom example (Extracted from the FetalMonitor example)

In this example, a new class derives from the **ChartZoom** class and the **MousePressed** event overridden. The event invokes the **PopZoomStack** method. Otherwise, the default operation of the **ChartZoom** class controls everything else.



[TypeScript]

```
this.zoomObj = QCRTGraphTS.ChartZoom.newChartZoom(this.chartVu, pTransform1, true);
this.zoomObj.setZoomYEnable(true);
this.zoomObj.setZoomXEnable(true);
this.zoomObj.setZoomXRoundMode(QCRTGraphTS.ChartConstants.AUTOAXES_FAR);
this.zoomObj.setZoomYRoundMode(QCRTGraphTS.ChartConstants.AUTOAXES_FAR);
this.zoomObj.setEnabled(false);
this.zoomObj.InternalZoomStackProcessing = true;
this.chartVu.setCurrentMouseListener(this.zoomObj);

private zoomOn_Button_Click(e: QCRTGraphTS.ButtonAreaEventArgs) {
    // Change to display of all collected data
    this.fetalHeartECGScrollFrame.ScrollScaleModeX =
QCRTGraphTS.ChartConstants.RT_AUTOSCALE_X_MINMAX;
    // Look at updatecounter number of points, which is all of them
    this.fetalHeartECGScrollFrame.MaxDisplayHistory = this.updatecounter;
    // Render graph based on new scale, showing all past data points
    if (this.chartVu)
        this.chartVu.updateDraw();
    // Now disable scroll frame
    this.fetalHeartECGScrollFrame.ChartObjEnable = QCRTGraphTS.ChartConstants.OBJECT_DISABLE;
    // Turn on zooming
    this.zoomObj.setEnabled(true);
}

private zoomRestore_Button_Click(e: QCRTGraphTS.ButtonAreaEventArgs) {
    let button: QCRTGraphTS.RTControlButton = (<QCRTGraphTS.RTControlButton>(e.Source));
    // Turn off zooming
    this.zoomObj.setEnabled(false);
    // Restore original y-scale values
    this.fetalHeartECGScrollFrame.ChartObjScale.ScaleStartY = -1;
    this.fetalHeartECGScrollFrame.ChartObjScale.ScaleStopY = 4;
    // Re-establish scroll mode
    this.fetalHeartECGScrollFrame.ScrollScaleModeX =
QCRTGraphTS.ChartConstants.RT_FIEXEXTENT_MOVINGSTART_AUTOSCROLL;
    this.fetalHeartECGScrollFrame.ChartObjEnable = QCRTGraphTS.ChartConstants.OBJECT_ENABLE;
    // Render graph
    if (this.chartVu)
}
}
```

[JavaScript]

```

zoomObj = QCRTGraphTS.ChartZoom.newChartZoom(chartVu, pTransform1, true);
zoomObj.setZoomYEnable(true);
zoomObj.setZoomXEnable(true);
zoomObj.setZoomXRoundMode(QCRTGraphTS.ChartConstants.AUTOAXES_FAR);
zoomObj.setZoomYRoundMode(QCRTGraphTS.ChartConstants.AUTOAXES_FAR);
zoomObj.setEnabled(false);
zoomObj.InternalZoomStackProcesssing = true;
chartVu.setCurrentMouseListener(zoomObj);

async function zoomOn_Button_Click(e) {
    // Change to display of all collected data
    fetalHeartECGScrollFrame.ScrollScaleModeX = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_X_MINMAX;
    // Look at updatecounter number of points, which is all of them
    fetalHeartECGScrollFrame.MaxDisplayHistory = updatecounter;
    // Render graph based on new scale, showing all past data points
    if (chartVu)
        chartVu.updateDraw();
    // Now disable scroll frame
    fetalHeartECGScrollFrame.ChartObjEnable = QCRTGraphTS.ChartConstants.OBJECT_DISABLE;
    // Turn on zooming
    zoomObj.setEnabled(true);
}

async function zoomRestore_Button_Click(e) {
    let button = (e.Source);
    // Turn off zooming
    zoomObj.setEnabled(false);
    // Restore original y-scale values
    fetalHeartECGScrollFrame.ChartObjScale.ScaleStartY = -1;
    fetalHeartECGScrollFrame.ChartObjScale.ScaleStopY = 4;
    // Re-establish scroll mode
    fetalHeartECGScrollFrame.ScrollScaleModeX =
QCRTGraphTS.ChartConstants.RT_FIXEEXTENT_MOVINGSTART_AUTOSCROLL;
    fetalHeartECGScrollFrame.ChartObjEnable = QCRTGraphTS.ChartConstants.OBJECT_ENABLE;
    // Render graph
    if (chartVu)
        chartVu.updateDraw();
}

```

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
```

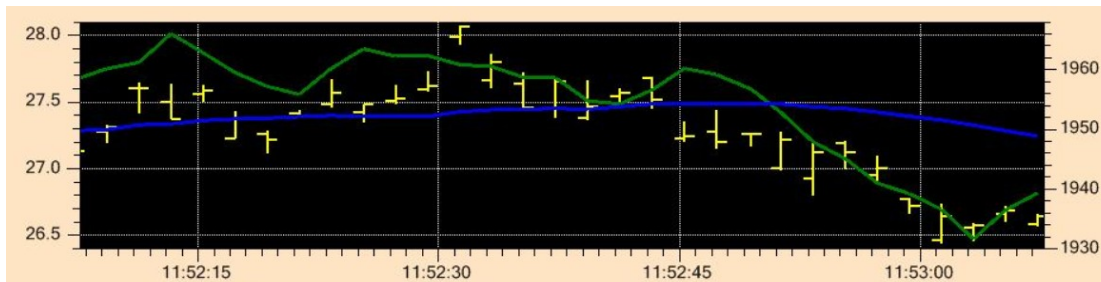
<i>component</i>	A reference to the ChartView object that the chart is placed in.
<i>transforms</i>	An array, size numtransforms, of the PhysicalCoordinates objects associated with the zoom operation.
<i>brescale</i>	True designates that the all of the scales should be re-scaled, once the final zoom rectangle is ascertained.

Call the **ChartZoom.setEnabled(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.

Super zoom example (Adapted from the RTStockDisplay example)

In this example, a new class derives from the **ChartZoom** class and the **MousePressed** event overridden. The event invokes the **PopZoomStack** method. Otherwise, the default operation of the **ChartZoom** class controls everything else.



[TypeScript]

```
let timecoordsarray: QCRTGraphTS.TimeCoordinates[] = [
    pTransform1,
    pTransform2];
this.zoomObj = QCRTGraphTS.ChartZoom.newChartSuperZoom(this.chartVu, timecoordsarray, true);

this.zoomObj.setZoomYEnable(true);
this.zoomObj.setZoomXEnable(true);
this.zoomObj.setZoomXRoundMode(QCRTGraphTS.ChartConstants.AUTOAXES_FAR);
this.zoomObj.setZoomYRoundMode(QCRTGraphTS.ChartConstants.AUTOAXES_FAR);
this.zoomObj.setEnabled(false);
this.zoomObj.setZoomStackEnable(true);
this.chartVu.setCurrentMouseListener(this.zoomObj);
.
.
```

```

private zoomOn_Button_Click(e: QCRTGraphTS.ButtonAreaEventArgs) {
    // Change to display of all collected data
    this.scrollFrame1.ScrollScaleModeX = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_X_MINMAX;
    // Look at updatecounter number of points, which is all of them
    this.scrollFrame1.MaxDisplayHistory = this.updatecounter;
    // Render graph based on new scale
    // Change to display of all collected data
    this.scrollFrame2.ScrollScaleModeX = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_X_MINMAX;
    // Look at updatecounter number of points, which is all of them
    this.scrollFrame2.MaxDisplayHistory = this.updatecounter;
    // Update first, to display all historical information, then disable to allow for zooming.
    if (this.chartVu)
        this.chartVu.updateDraw();
    this.scrollFrame2.ChartObjEnable = QCRTGraphTS.ChartConstants.OBJECT_DISABLE;
    this.scrollFrame1.ChartObjEnable = QCRTGraphTS.ChartConstants.OBJECT_DISABLE;
    // Turn on zooming
    this.zoomObj.setEnable(true);
}

private zoomRestore_Button_Click(e: QCRTGraphTS.ButtonAreaEventArgs) {
    let button: QCRTGraphTS.RTControlButton = (<QCRTGraphTS.RTControlButton>(e.Source));
    // Turn off zooming
    this.zoomObj.setEnable(false);
    // Re-establish scroll mode
    this.scrollFrame1.ScrollScaleModeX =
QCRTGraphTS.ChartConstants.RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL;
    this.scrollFrame1.ChartObjEnable = QCRTGraphTS.ChartConstants.OBJECT_ENABLE;
    // Re-establish scroll mode
    this.scrollFrame2.ScrollScaleModeX =
QCRTGraphTS.ChartConstants.RT_FIXEDEXTENT_MOVINGSTART_AUTOSCROLL;
    this.scrollFrame2.ChartObjEnable = QCRTGraphTS.ChartConstants.OBJECT_ENABLE;
    // Render graph
    if (this.chartVu)
        this.chartVu.updateDraw();
}

```

[JavaScript]

```

let timecoordsarray = [
    pTransform1,
    pTransform2
];
this.zoomObj = QCRTGraphTS.ChartZoom.newChartSuperZoom(this.chartVu, timecoordsarray, true);
this.zoomObj.setZoomYEnable(true);
this.zoomObj.setZoomXEnable(true);
this.zoomObj.setZoomXRoundMode(QCRTGraphTS.ChartConstants.AUTOAXES_FAR);
this.zoomObj.setZoomYRoundMode(QCRTGraphTS.ChartConstants.AUTOAXES_FAR);
this.zoomObj.setEnable(false);
this.zoomObj.setZoomStackEnable(true);
this.chartVu.setCurrentMouseListener(this.zoomObj)

zoomOn_Button_Click(e) {
    // Change to display of all collected data
    this.scrollFrame1.ScrollScaleModeX = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_X_MINMAX;
    // Look at updatecounter number of points, which is all of them
    this.scrollFrame1.MaxDisplayHistory = this.updatecounter;
    // Render graph based on new scale
    // Change to display of all collected data
    this.scrollFrame2.ScrollScaleModeX = QCRTGraphTS.ChartConstants.RT_AUTOSCALE_X_MINMAX;
    // Look at updatecounter number of points, which is all of them
    this.scrollFrame2.MaxDisplayHistory = this.updatecounter;
    // Update first, to display all historical information, then disable to allow for zooming.
    if (this.chartVu)
        this.chartVu.updateDraw();
    this.scrollFrame2.ChartObjEnable = QCRTGraphTS.ChartConstants.OBJECT_DISABLE;
    this.scrollFrame1.ChartObjEnable = QCRTGraphTS.ChartConstants.OBJECT_DISABLE;
    // Turn on zooming
}

```

```

        this.zoomObj.setEnabled(true);
    }
    zoomRestore_Button_Click(e) {
        let button = (e.Source);
        // Turn off zooming
        this.zoomObj.setEnabled(false);
        // Re-establish scroll mode
        this.scrollFrame1.ScrollScaleModeX =
        QCRTGraphTS.ChartConstants.RT_FIXEEXTENT_MOVINGSTART_AUTOSCROLL;
        this.scrollFrame1.ChartObjEnable = QCRTGraphTS.ChartConstants.OBJECT_ENABLE;
        // Re-establish scroll mode
        this.scrollFrame2.ScrollScaleModeX =
        QCRTGraphTS.ChartConstants.RT_FIXEEXTENT_MOVINGSTART_AUTOSCROLL;
        this.scrollFrame2.ChartObjEnable = QCRTGraphTS.ChartConstants.OBJECT_ENABLE;
        // Render graph
        if (this.chartVu)
            this.chartVu.updateDraw();
    }
}

```

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 infinite 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 =
QCRTGraphTS.ChartCalendar.getCalendarWidthValue(QCRTGraphTS.ChartConstants.MINUTE, 45);
let minZoomYRange = 0.01;
let zoomLimits = QCRTGraphTS.ChartDimension.newChartDimension(minZoomTimeRange, minZoomYRange);
zoomObj.setZoomRangeLimits(zoomLimits);

```

[JavaScript]

```

let minZoomTimeRange: number =
QCRTGraphTS.ChartCalendar.getCalendarWidthValue(QCRTGraphTS.ChartConstants.MINUTE, 45);
let minZoomYRange: number = 0.01;
let zoomLimits: QCRTGraphTS.ChartDimension =
QCRTGraphTS.ChartDimension.newChartDimension(minZoomTimeRange, minZoomYRange);
zoomObj.setZoomRangeLimits(zoomLimits);

```

17. Miscellaneous Shape Drawing

RTRoundedRectangle2D

RTGenShape

Often the look and feel of a real-time display is greatly enhanced with the addition of a few simple circular and rectangular drawing shapes. All of the example programs use these shapes, either directly, or indirectly as the 3D border element of the **RTPanelMeter** class. The chapter discusses how to explicitly add these objects to your program

3D Borders and Background Frames

Class RT3DFrame

GraphObj

RT3DFrame

This class is used to draw 3D borders and provide the background for many of the other graph objects, most noticeably the **RTPanelMeter** classes. It can also be used directly in your program to provide 3D frames the visually group objects together in a faceplate format.

RT3DFrame constructors

```
public static newRT3DFrameCoords(transform: PhysicalCoordinates): RT3DFrame

public static newRT3DFrameCoordsAttrib(transform: PhysicalCoordinates, attrib: ChartAttribute):
RT3DFrame

public static newRT3DFrameCoordsRectAttrib(transform: PhysicalCoordinates, rect:
ChartRectangle2D, attrib: ChartAttribute): RT3DFrame

public static newRT3DFrameCoordsRectAttribPos(transform: PhysicalCoordinates, rect:
ChartRectangle2D, attrib: ChartAttribute, postype: number): RT3DFrame

public static newRT3DFrame(transform: PhysicalCoordinates, rect: ChartRectangle2D, attrib:
ChartAttribute, postype: number): RT3DFrame
```

Parameters

transform

Places the **RT3DFrame** object in the coordinate system defined by transform.

rect

Specifies the position and size of the frame.

attrib

Specifies the attributes (line and fill color) for the frame.

postype

Specifies the positioning coordinate system.

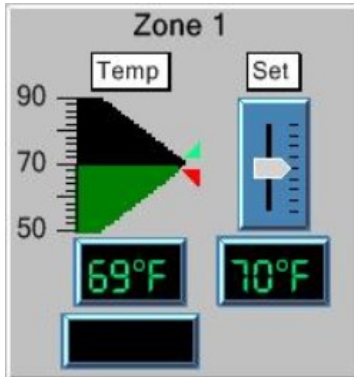
Selected Public Instance Properties

BoundingBox (inherited from GraphObj)	Returns the bounding box for the chart object. Not all chart objects have bounding boxes. Be sure and check for null.
ChartObjAttributes (inherited from GraphObj)	Sets the attributes for a chart object using a ChartAttribute object.
FillColor (inherited from GraphObj)	Sets the fill color for the chart object.
FrameRect	Set/Get the ChartRectangle2D object used to define the position and size of the 3D frame.
LightMode	Set/Get the apparent direction of the light source used to highlight the 3D frame. Use one of the direction constants. LIGHT_UPPER_LEFT, LIGHT_UPPER_RIGHT, LIGHT_LOWER_LEFT, LIGHT_LOWER_RIGHT, LIGHT_STRAIGHT_ON, LIGHT_NONE, OUTSET_3D_LOOK, INSET_3D_LOOK.
LineColor (inherited from GraphObj)	Sets the line color for the chart object.
LineStyle (inherited from GraphObj)	Sets the line style for the chart object.
LineWidth (inherited from GraphObj)	Sets the line width for the chart object.
PositionType (inherited from GraphObj)	Get/Sets the current position type.

A complete listing of **RT3DFrame** properties is found in the `_qcrgraphpts_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example for drawing RT3DFrame objects

The example below, extracted from the HomeAutomation example, file `HomeAutomation`, class `ThermostatUserController1`, draws an **RT3DFrame** object the size of the entire control area. Since each separate control in the example has a similar **RT3DFrame** background, it provides a visual grouping of the objects in each control.



[TypeScript]

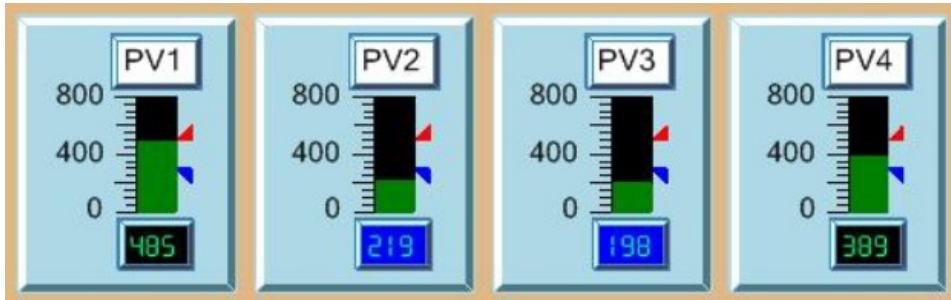
```
private InitializeBackgroundPanel() {
    if (!this.chartVu) return;

    let theFont: QCRTGraphTS.ChartFont = font10Bold;
    let normrect: QCRTGraphTS.ChartRectangle2D =
QCRTGraphTS.ChartRectangle2D.newChartRectangle2D(0, 0, 1, 1);
    let pTransform1: QCRTGraphTS.CartesianCoordinates = new QCRTGraphTS.CartesianCoordinates();
    let frame3d: QCRTGraphTS.RT3DFrame = QCRTGraphTS.RT3DFrame.newRT3DFrame(pTransform1, normrect,
this.facePlateAttrib, QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
    this.chartVu.addChartObject(frame3d);
}
```

[JavaScript]

```
InitializeBackgroundPanel() {
    if (!this.chartVu)
        return;
    let theFont = font10Bold;
    let normrect = QCRTGraphTS.ChartRectangle2D.newChartRectangle2D(0, 0, 1, 1);
    let pTransform1 = new QCRTGraphTS.CartesianCoordinates();
    let frame3d = QCRTGraphTS.RT3DFrame.newRT3DFrame(pTransform1, normrect, this.facePlateAttrib,
QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
    this.chartVu.addChartObject(frame3d);
}
```

The example below, extracted from the PIDControlTuner example, file PIDControlTuner, method **InitializeTopBargraphs**, draws an **RT3DFrame** object as a backdrop for each of the bargraphs.



[TypeScript]

```
for (let i: number = 0; (i < this.PIDProcessItems.length); i++) {
    row = QCRTGraphTS.ChartSupport.intDiv(i, 8);
    col = QCRTGraphTS.ChartSupport.intMod(i, 8);
    x1 = (xoffset
        + (col * faceplatewidthspacing));
    y1 = (yoffset
        + (row * faceplateheightspacing));
    x2 = (x1 + faceplatewidth);
    y2 = (y1 + faceplateheight);
    let mindisplayvalue: number = this.PIDProcessItems[i].DefaultMinimumDisplayValue;
    let maxdisplayvalue: number = this.PIDProcessItems[i].DefaultMaximumDisplayValue;
    pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, mindisplayvalue, 1,
maxdisplayvalue);
    let normrect: QCRTGraphTS.ChartRectangle2D =
QCRTGraphTS.ChartRectangle2D.newChartRectangle2D(x1, y1, faceplatewidth, faceplateheight);
    let frame3d: QCRTGraphTS.RT3DFrame = QCRTGraphTS.RT3DFrame.newRT3DFrame(pTransform1, normrect,
rectattrib, QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
    this.chartVu.addChartObject(frame3d);
    .
    .
    .
    .
}
```

[JavaScript]

```
for (let i = 0; (i < PIDProcessItems.length); i++) {
    row = QCRTGraphTS.ChartSupport.intDiv(i, 8);
    col = QCRTGraphTS.ChartSupport.intMod(i, 8);
    x1 = (xoffset
        + (col * faceplatewidthspacing));
    y1 = (yoffset
        + (row * faceplateheightspacing));
    x2 = (x1 + faceplatewidth);
    y2 = (y1 + faceplateheight);
    let mindisplayvalue = PIDProcessItems[i].DefaultMinimumDisplayValue;
    let maxdisplayvalue = PIDProcessItems[i].DefaultMaximumDisplayValue;
    pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0, mindisplayvalue, 1,
maxdisplayvalue);
    let normrect = QCRTGraphTS.ChartRectangle2D.newChartRectangle2D(x1, y1, faceplatewidth,
faceplateheight);
    let frame3d = QCRTGraphTS.RT3DFrame.newRT3DFrame(pTransform1, normrect, rectattrib,
QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
    chartVu.addChartObject(frame3d);
    .
    .
    .
    .
}
```

Rounded Rectangles

Class **RToundedRectangle2D**

GraphObj

RToundedRectangle2D

Rounded rectangles are just that, rectangles that have rounded corners.

RToundedRectangle2D constructors

```
public static newRToundedRectangle2D(transform: PhysicalCoordinates, xx: number, yy: number, ww: number, hh: number, corner: number, postype: number): RToundedRectangle2D
```

```
public static newRToundedRectangle2DCoordsRectPosType(transform: PhysicalCoordinates, r: RToundedRectangle2D, postype: number): RToundedRectangle2D
```

```
public static newRToundedRectangle2DCoordsRectCornerPosType(transform: PhysicalCoordinates, r: ChartRectangle2D, corner: number, postype: number): RToundedRectangle2D
```

Parameters

transform

Places the **RToundedRectangle2D** object in the coordinate system defined by transform.

r

The size and position of the rectangle.

corner

The radius of the rectangle corners.

postype

The coordinate system the rectangle is defined in. Use one of the coordinate system constants: DEV_POS, PHYS_POS, NORM_GRAPH_POS, NORM_PLOT_POS.

Selected Public Instance Properties

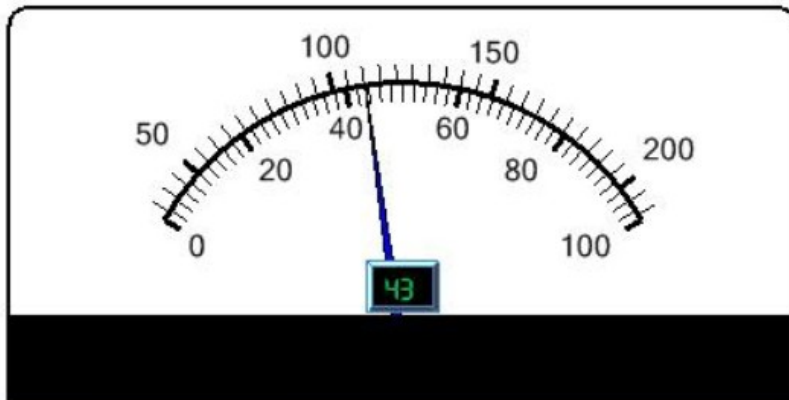
CornerRadius	Get/Set the corner radius of the rounded rectangle.
FillColor (inherited from GraphObj)	Sets the fill color for the chart object.
Height	Get/Set the height of the rectangle.
LineColor (inherited from GraphObj)	Sets the line color for the chart object.
LineStyle (inherited from GraphObj)	Sets the line style for the chart object.
LineWidth (inherited from GraphObj)	Sets the line width for the chart object.
PositionType (inherited from GraphObj)	Get/Sets the current position type.
Width	Get/Set the width of the rectangle.
X	Get/Set the x-value of the rectangle.
Y	Get/Set the y-value of the rectangle.
ZOrder (inherited from GraphObj)	Sets the z-order of the object in the chart. Every object has a z-order value. Each z-order value does NOT have to be unique. If

z-order sorting is turned on in the ChartView object, objects are sorted by z-order before they are drawn.

A complete listing of **RToundedRectangle2D** properties is found in the `_qcrgraphTS_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example for drawing **RToundedRectangle2D** objects

The example below, extracted from the Meters example, file `NeedleMeters`, method **InitializeMeter2.**, draws a large rectangle with rounded corners as the frame of the meter, and a smaller, filled rectangle at the bottom.



[TypeScript]

```
let rr: QCRTGraphTS.RToundedRectangle2D=
QCRTGraphTS.RToundedRectangle2D.newRToundedRectangle2D(meterframe1, 0.25, 0.01, 0.45, 0.43,
0.01, QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
let backattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 2,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
rr.setChartObjAttributes(backattrib);
rr.setChartObjClipping(QCRTGraphTS.ChartConstants.NO_CLIPPING);
this.chartVu.addChartObject(rr);

let rrr: QCRTGraphTS.RToundedRectangle2D=
QCRTGraphTS.RToundedRectangle2D.newRToundedRectangle2D(meterframe1, 0.25, 0.35, 0.45, 0.09, 0,
QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
let backattrib2: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 2,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
rrr.setChartObjAttributes(backattrib2);
rrr.ZOrder = 60;
rrr.setChartObjClipping(QCRTGraphTS.ChartConstants.NO_CLIPPING);
this.chartVu.addChartObject(rrr);
```

[JavaScript]

```
let rr = QCRTGraphTS.RToundedRectangle2D.newRToundedRectangle2D(meterframe1, 0.25, 0.01, 0.45,
0.43, 0.01, QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
let backattrib = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 2,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
```

```

rr.setChartObjAttributes(backattrib);
rr.setChartObjClipping(QCRTGraphTS.ChartConstants.NO_CLIPPING);
chartVu.addChartObject(rr);

let rrr = QCRTGraphTS.RTRoundedRectangle2D.newRTRoundedRectangle2D(meterframe1, 0.25, 0.35, 0.45,
0.09, 0, QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
let backattrib2 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.BLACK, 2,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
rrr.setChartObjAttributes(backattrib2);
rrr.ZOrder = 60;
rrr.setChartObjClipping(QCRTGraphTS.ChartConstants.NO_CLIPPING);
chartVu.addChartObject(rrr);

```

General Shapes

Class RTGenShape

GraphObj

RTGenShape

This class is used to draw filled and unfilled rectangles, rectangles with rounded corners, general ellipses and aspect ratio corrected circles. These shapes can be used by the programmer to add visual enhancements to graphs.

RTGenShape constructors

```

public static newRTGenShape(transform: PhysicalCoordinates, xx: number, yy: number, ww: number,
hh: number, corner: number, shape: number, postype: number): RTGenShape

public static newRTGenShapeRect(transform: PhysicalCoordinates, rect: ChartRectangle2D, corner:
number, shape: number, postype: number): RTGenShape

```

Parameters

transform

The coordinate system for the new **RTGenShape** object.

rect

The source rectangle.

corner

The corner radius of the rounded rectangle.

shape

The shape of the **RTGenShape** object. Use one of the generalized shape constants: RT_SHAPE_RECTANGLE, RT_SHAPE_ROUNDEDRECTANGLE, RT_SHAPE_ELLIPSE.

postype

Specifies what coordinate system the coordinates reference. Use one of the position type constants: DEV_POS, PHYS_POS, POLAR_POS, NORM_GRAPH_POS, NORM_PLOT_POS.

Selected Public Instance Properties

AspectRatioCorrection	Get/Set the aspect ratio correction mode for the

	RT_SHAPE_ELLIPSE shape: NO_ASPECT_RATIO_CORRECTION, FIXED_X_ASPECT_RATIO_CORRECTION, FIXED_Y_ASPECT_RATIO_CORRECTION.
ChartObjAttributes (inherited from GraphObj)	Sets the attributes for a chart object using a ChartAttribute object.
CornerRadius	Get/Set the corner radius of the rounded rectangle.
FillColor (inherited from GraphObj)	Sets the fill color for the chart object.
GenShape	Get/Set the shape control property genShape. Use one of the generalized shape constants: RT_SHAPE_RECTANGLE, RT_SHAPE_ROUNDEDRECTANGLE, RT_SHAPE_ELLIPSE.
Height	Get/Set the height of the shape rectangle.
LineColor (inherited from GraphObj)	Sets the line color for the chart object.
LineStyle (inherited from GraphObj)	Sets the line style for the chart object.
LineWidth (inherited from GraphObj)	Sets the line width for the chart object.
PositionType (inherited from GraphObj)	Get/Sets the current position type.
ShapeRect	Get/Set the rectangle control the size and position of the object.
Width	Get/Set the width of the rectangle.
X	Get/Set the x-value of the shape rectangle.
Y	Get/Set the y-value of the shape rectangle.
ZOrder (inherited from GraphObj)	Sets the z-order of the object in the chart. Every object has a z-order value. Each z-order value does NOT have to be unique. If z-order sorting is turned on in the ChartView object, objects are sorted by z-order before they are drawn.

A complete listing of **RTGenShape** properties is found in the `_qcrgraphts_.html` documentation file, located in the `QCRTGraphTS\docs\modules` subdirectory.

Example for drawing RTGenShape objects

The example below, extracted from the `AutoInstrumentPanel` example, file `AutoInstrumentPanel`, method **InitializeClock**, draws a circle around the borders of the clock.

[TypeScript]

```

let attrib2: QCRTGraphTS.ChartAttribute =
  QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GRAY, 5,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
let shaperect: QCRTGraphTS.ChartRectangle2D =
  QCRTGraphTS.ChartRectangle2D.newChartRectangle2D(0.8, 0.025, 0.19, 0.25);
let genshape: QCRTGraphTS.RTGenShape = QCRTGraphTS.RTGenShape.newRTGenShapeRect(meterframe,
  shaperect, 0, QCRTGraphTS.ChartConstants.RT_SHAPE_ELLIPSE,
  QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
genshape.setChartObjAttributes(attrib2);
this.chartVu.addChartObject(genshape);

```

[JavaScript]

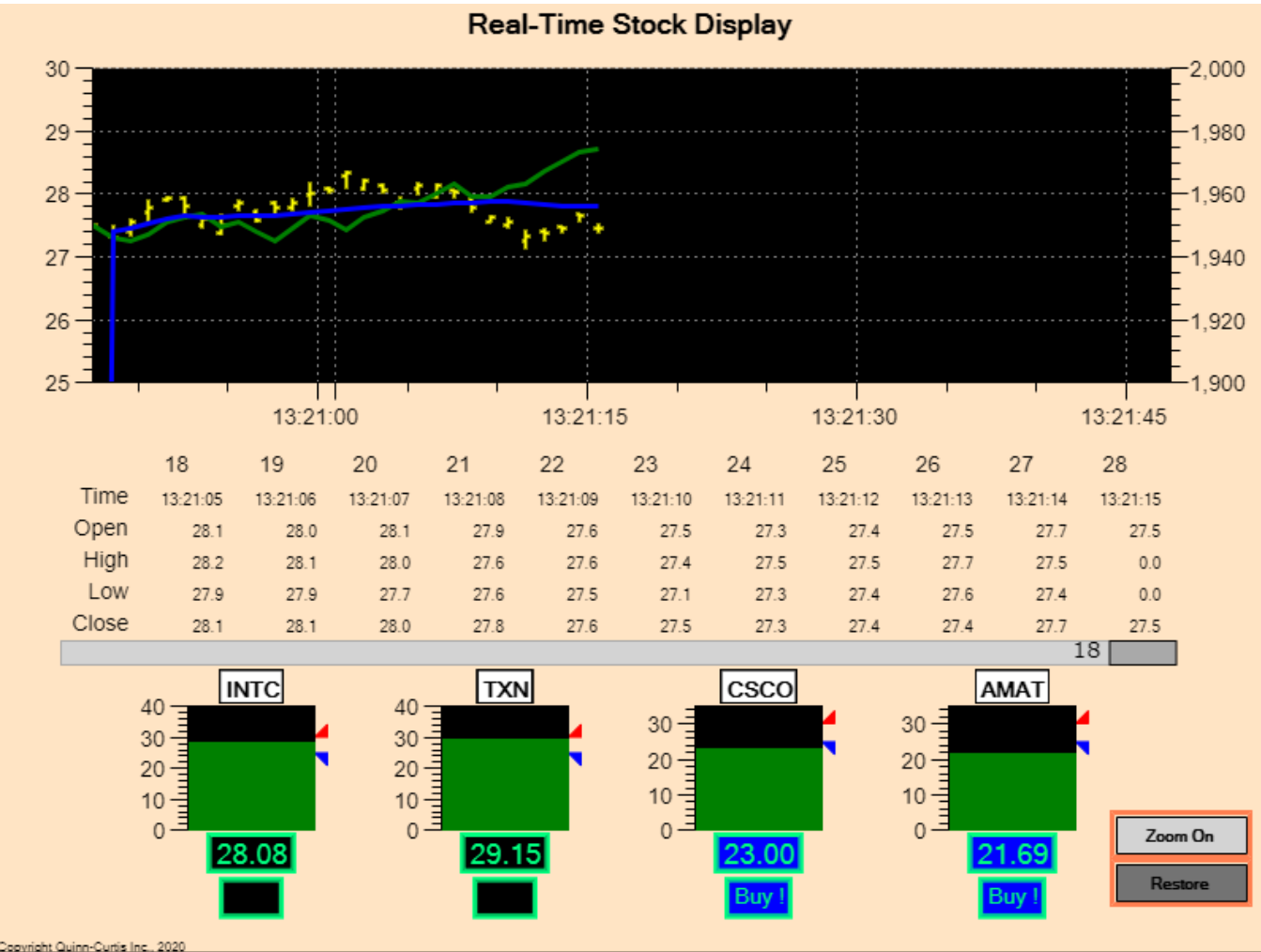
```

let attrib2 = QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GRAY, 5,
  QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
let shaperect = QCRTGraphTS.ChartRectangle2D.newChartRectangle2D(0.8, 0.025, 0.19, 0.25);
let genshape = QCRTGraphTS.RTGenShape.newRTGenShapeRect(meterframe, shaperect, 0,
  QCRTGraphTS.ChartConstants.RT_SHAPE_ELLIPSE, QCRTGraphTS.ChartConstants.NORM_GRAPH_POS);
genshape.setChartObjAttributes(attrib2);
chartVu.addChartObject(genshape);

```

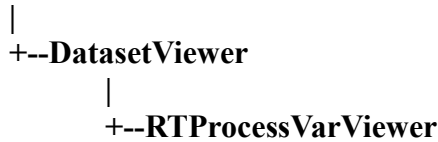
18. Process Variable Viewer

RTProcessVarViewer



The **RTProcessVarViewer** class extends the **DatasetViewer** class so that it can display the historical datasets stored in the **RTProcessVar** objects. The **RTProcessVarViewer** can be updated in real-time, and synchronized to the chart, so that scrolling of the **RTProcessVarViewer** can scroll the chart.

Class RTProcessVarViewer

ChartView

The **RTProcessVarViewer** is a **ChartView** derived object. Use it to view one or more **RTProcessVar** objects in a real-time display. Since it is usually not possible or practical to display the entire dataset, the **RTProcessVarViewer** windows a rectangular section of the dataset for display. Scroll bars are used to scroll the rows and columns of the dataset. The **RTProcessVarViewer** constructor defines the size, position, source matrix, the number of rows and columns of the **RTProcessVarViewer** grid, and the starting position of the **RTProcessVarViewer** scrollbar.

RTProcessVarViewer constructor

```

public static newRTProcessVarViewer(chartvu: ChartView, transform: PhysicalCoordinates,
posrect: ChartRectangle2D, pv: RTProcessVar, rows: number, cols: number, start: number):
RTProcessVarViewer

public static newRTProcessVarViewerProcessVarRC(chartvu: ChartView, posrect: ChartRectangle2D,
pv: RTProcessVar,
rows: number, cols: number, start: number): RTProcessVarViewer

public static newRTProcessVarViewerProcessVarRCOrient(chartvu: ChartView, posrect:
ChartRectangle2D, pv: RTProcessVar,
rows: number, cols: number, start: number, orient: number): RTProcessVarViewer

public static newRTProcessVarViewerCoordsProcessVarRCOrient(chartvu: ChartView, transform:
PhysicalCoordinates,
posrect: ChartRectangle2D, pv: RTProcessVar, rows: number, cols: number, start: number, orient:
number): RTProcessVarViewer

```

<i>chartvu</i>	The ChartView object the DatasetViewer is placed in.
<i>transform</i>	The coordinate system the DatasetViewer is placed in.
<i>posrect</i>	A positioning rectangle (using normalized chart coordinates) for the dataset viewer, use null if not used.
<i>pv</i>	The initial process variable.
<i>rows</i>	Number of rows to display
<i>cols</i>	Number of columns to display.
<i>start</i>	Starting column of the dataset viewer.
<i>orioent</i>	Data table orientation. Use HORIZ_DIR for horizontal (default) or VERT_DIR for vertical..

Set unique fonts for the column headers, row headers and grid cells using the **ColumnHeaderFont**, **RowHeaderFont** 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 **RTProcessVar** objects to a **RTProcessVarViewer** using the `RTProcessVarViewer.addProcessVar` method. When adding additional process variables, 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.

Selected Public Instance Properties

AltGridAttribute	(Inherited from DataGridBase.)
AutoRedrawTable	Set to true and the table will redraw using the current data associated with the update of the RTProcessVar.
ColumnHeaderAttribute	(Inherited from DataGridBase.)
ColumnHeaderFont	(Inherited from DatasetViewer.)
ColumnHeads	(Inherited from DataGridBase.)
DataArray	(Inherited from DatasetViewer.)
Dock	Gets or sets which control borders are docked to its parent control and determines how a control is resized with its parent. (Inherited from Control .)
DoubleBufferEnable	(Inherited from ChartView.)
DrawEnable	(Inherited from ChartView.)
GridAttribute	(Inherited from DataGridBase.)
GridCellFont	(Inherited from DatasetViewer.)
Height	Gets or sets the height of the control. (Inherited from Control .)
HorizontalGroupPlot	(Inherited from DatasetViewer.)
HorizontalScroll	Gets the characteristics associated with the horizontal scroll bar. (Inherited from ScrollableControl .)
HScroll	Gets or sets a value indicating whether the horizontal scroll bar is visible. (Inherited from ScrollableControl .)

HScrollBar1	(Inherited from DataGridBase.)
Left	Gets or sets the distance, in pixels, between the left edge of the control and the left edge of its container's client area. (Inherited from Control .)
Location	Gets or sets the coordinates of the upper-left corner of the control relative to the upper-left corner of its container. (Inherited from Control .)
NumCols	(Inherited from DataGridBase.)
NumericFormat	(Inherited from DataGridBase.)
NumRows	(Inherited from DataGridBase.)
ParentChartView	(Inherited from DataGridBase.)
ParentTransform	(Inherited from DataGridBase.)
PreferredSize	(Inherited from ChartView.)
SizeMode	(Inherited from ChartView.)
Right	Gets the distance, in pixels, between the right edge of the control and the left edge of its container's client area. (Inherited from Control .)
RowHeaderAttribute	(Inherited from DataGridBase.)
RowHeaderFont	(Inherited from DatasetViewer.)
RowHeads	(Inherited from DataGridBase.)
SmoothingMode	(Inherited from ChartView.)
SourceDataset	(Inherited from DatasetViewer.)
StartCol	(Inherited from DataGridBase.)
StartRow	(Inherited from DataGridBase.)
SyncChart	(Inherited from DatasetViewer.)
TableGreenBarFlag	(Inherited from DataGridBase.)
TableStartPosX	(Inherited from DataGridBase.)
TableStartPosY	(Inherited from DataGridBase.)
TableStopPosX	(Inherited from DataGridBase.)
TableStopPosY	(Inherited from DataGridBase.)
TextRenderingHint	(Inherited from ChartView.)
Title	(Inherited from DataGridBase.)
Top	Gets or sets the distance, in pixels, between the top edge of the control and the top edge of its container's client area. (Inherited from Control .)
TransformList	(Inherited from DataGridBase.)
UseStripedGridBackground	(Inherited from DataGridBase.)
VerticalScroll	Gets the characteristics associated with the vertical scroll bar. (Inherited from ScrollableControl .)
Visible	Gets or sets a value indicating whether the control is displayed. (Inherited from Control .)
VScroll	Gets or sets a value indicating whether the vertical scroll bar

VScrollBar1	is visible. (Inherited from ScrollableControl .) (Inherited from DataGridBase.)
Width	Gets or sets the width of the control. (Inherited from Control .)

Simple RTProcessVarViewer example (extracted from the example program RTStockDisplay)

[TypeScript]

```
let posrect: QCRTGraphTS.ChartRectangle2D = QCRTGraphTS.ChartRectangle2D.newChartRectangle2D(0.05
, 0.67, 0.87, 0.26);
let startindex: number = 0;
let rows: number = 3;
let columns: number = 11;

this.rtProcessVarViewer1 = QCRTGraphTS.RTProcessVarViewer.newRTProcessVarViewer(this.chartVu, pTr
ansform1, posrect, this.currentTemperature1, rows, columns, startindex);

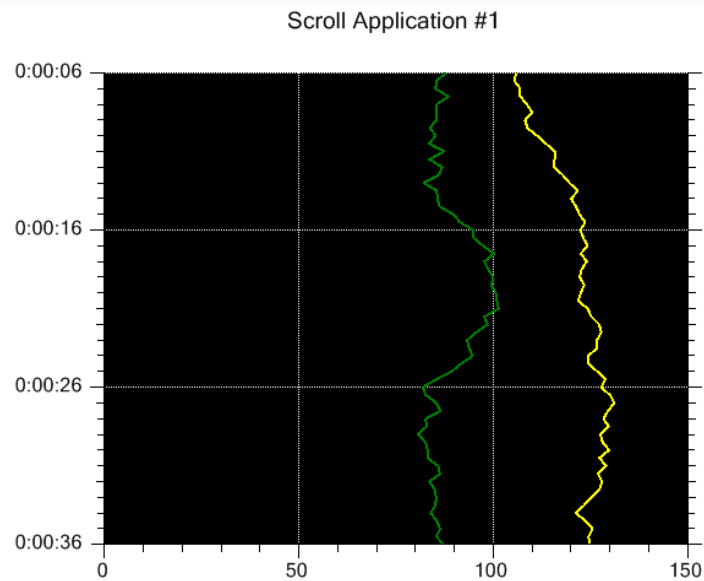
this.rtProcessVarViewer1.UseStripedGridBackground = true;
this.rtProcessVarViewer1.GridCellFont = this.font10;
this.rtProcessVarViewer1.addProcessVar(this.currentTemperature2);
// Set custom decimal precision for each row
this.rtProcessVarViewer1.setFormatDecimalPos(1, 1);
this.rtProcessVarViewer1.setFormatDecimalPos(2, 2);

this.chartVu.addChartObject(this.rtProcessVarViewer1);
```

[JavaScript]

```
let posrect = QCRTGraphTS.ChartRectangle2D.newChartRectangle2D(0.05, 0.67, 0.87, 0.26);
let startindex = 0;
let rows = 3;
let columns = 11;
this.rtProcessVarViewer1 = QCRTGraphTS.RTProcessVarViewer.newRTProcessVarViewer(this.chartVu,
pTransform1, posrect, this.currentTemperature1, rows, columns, startindex);
this.rtProcessVarViewer1.UseStripedGridBackground = true;
this.rtProcessVarViewer1.GridCellFont = this.font10;
this.rtProcessVarViewer1.addProcessVar(this.currentTemperature2);
// Set custom decimal precision for each row
this.rtProcessVarViewer1.setFormatDecimalPos(1, 1);
this.rtProcessVarViewer1.setFormatDecimalPos(2, 2);
this.chartVu.addChartObject(this.rtProcessVarViewer1);
```

Vertical Orientation DatasetViewer example (extracted from the example program ProcessVarDataTables.ElapsedTimeVerticalScrolling.)



	Time	Temp 1	Temp 2
61	0:00:31	129.01	85.79
62	0:00:31	126.97	86.26
63	0:00:32	127.96	83.55
64	0:00:32	127.33	84.92
65	0:00:33	125.11	85.29
66	0:00:33	123.37	85.13
67	0:00:34	121.23	83.89
68	0:00:34	123.64	85.39
69	0:00:35	125.55	86.29
70	0:00:35	124.50	85.36
71	0:00:36	124.72	87.21

[TypeScript]

```
let posrect: QCRTGraphTS.ChartRectangle2D =
  QCRTGraphTS.ChartRectangle2D.newChartRectangle2D(0.7, 0.1, 0.29, 0.8);
this.rtProcessVarViewer1 =
  QCRTGraphTS.RTProcessVarViewer.newRTProcessVarViewerCoordsProcessVarRCOrient(this.chartVu,
    pTransform1, posrect, this.currentTemperature1, 11, 3, 0, QCRTGraphTS.ChartConstants.VERT_DIR);

this.rtProcessVarViewer1.GridCellFont = this.font10;
this.rtProcessVarViewer1.addProcessVar(this.currentTemperature2);

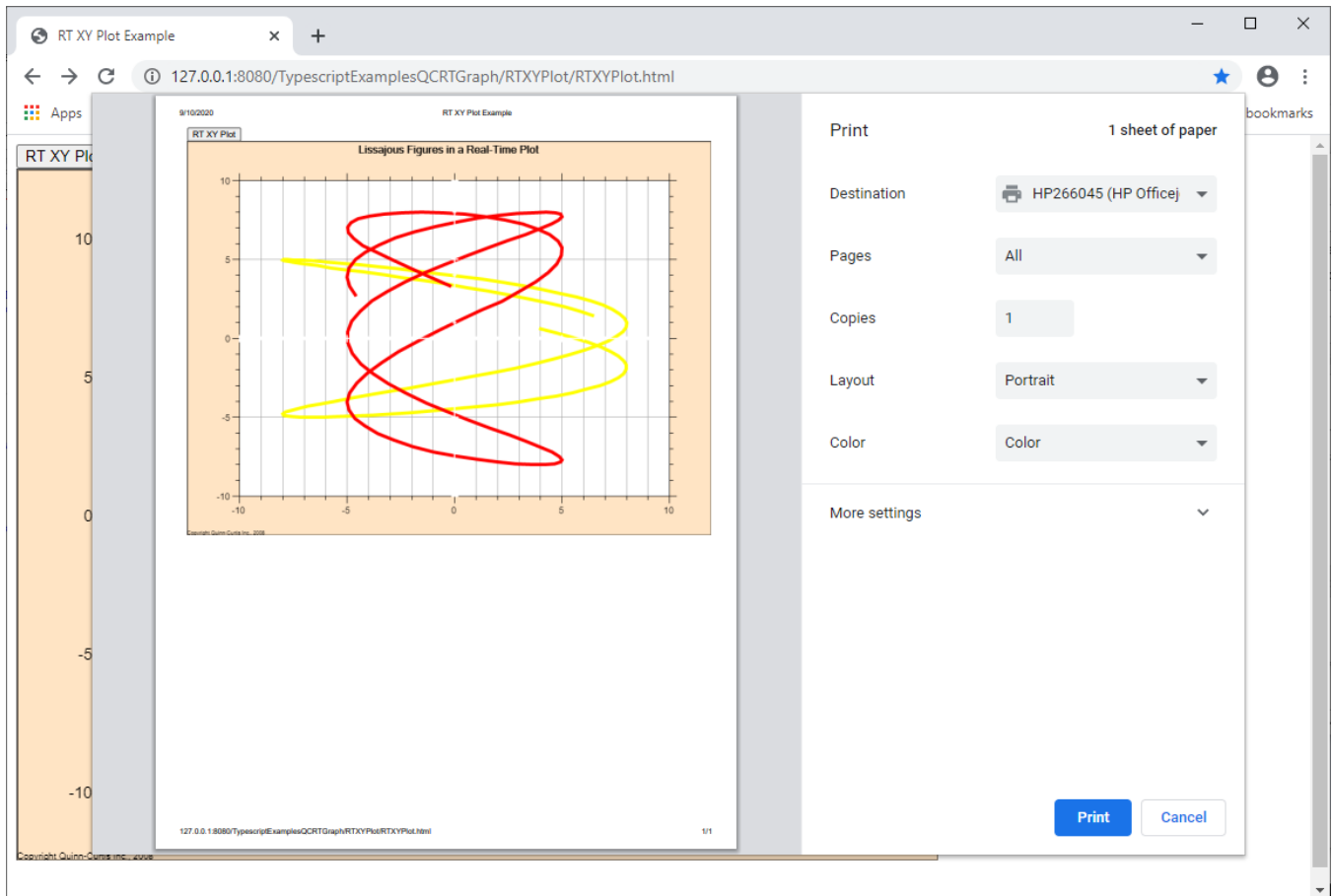
this.chartVu.addChartObject(this.rtProcessVarViewer1);
```

[JavaScript]

```
let posrect = QCRTGraphTS.ChartRectangle2D.newChartRectangle2D(0.7, 0.1, 0.29, 0.8);
this.rtProcessVarViewer1 =
  QCRTGraphTS.RTProcessVarViewer.newRTProcessVarViewerCoordsProcessVarRCOrient(this.chartVu,
    pTransform1, posrect, this.currentTemperature1, 11, 3, 0, QCRTGraphTS.ChartConstants.VERT_DIR);
this.rtProcessVarViewer1.GridCellFont = this.font10;
this.rtProcessVarViewer1.addProcessVar(this.currentTemperature2);
this.chartVu.addChartObject(this.rtProcessVarViewer1);
```


19. Printing and Copying the Canvas Image

The **QCRTGraph 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.toDataURL` 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 programmatically. 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]

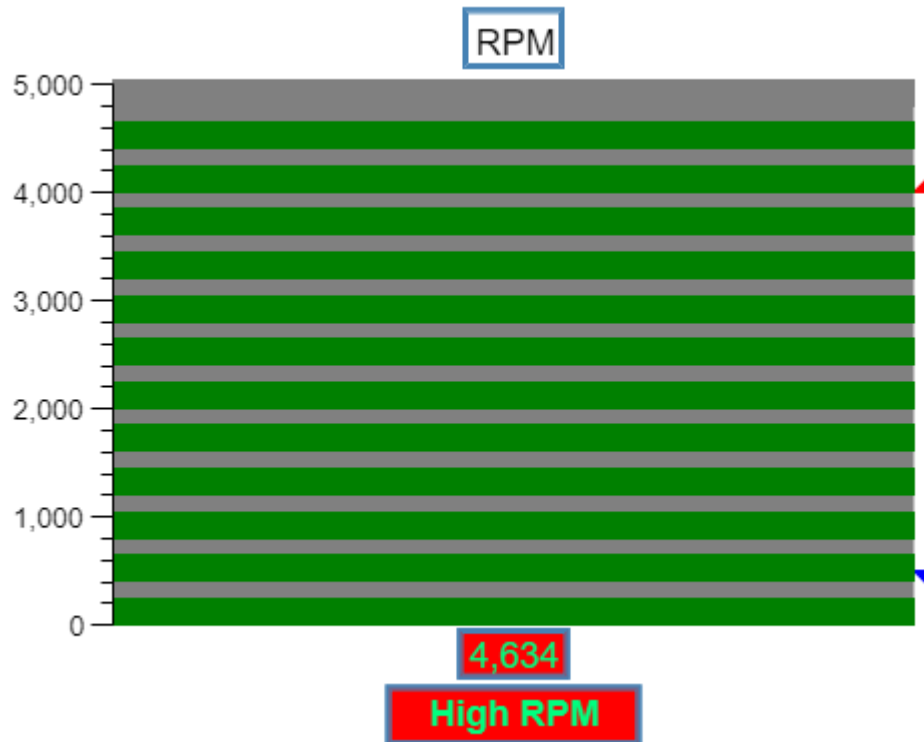
```
chartVu.updateDraw();
.
.
.
```

```
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();
```


20. Using Real-Time Graphics Tools 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 QCRTGraphTS library file (qcrgraph.ts.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 { BuildRTBarPlot } from './HelloRTBarPlot.js';
```

The “./” in the folder specification above represents the current folder that contains the HTML file you are working with.

Even though the TypescriptExamplesQCRTGraph 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/JavascriptExamplesQCRTGraph/HelloRTBarPlot/HelloRTBarPlot.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/JavascriptExamplesQCRTGraph/HelloRTBarPlot/HelloRTBarPlot.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.

Canvas Overlap Mode

You can have individual charts, sharing the same Canvas as most of our examples do. You use buttons to toggle between the different charts. In this case you do not want the mouse events from the charts not on top (the most recently drawn chart is the one on top) colliding with the current, on top, chart. So we keep track of the most recently drawn chart and assume that one is on top. Only mouse events for that chart are processed. However, this logic fails if there are multiple canvases on the HTML page, and a different chart placed in each one. In this case, since the charts are not overlapping, you want the mouse processing to be active on every chart.

You will need to adjust a static property, depending on how you arrange your charts on the HTML page.

If the charts are on top of one another, as most all of example use this motif, you don't need to do anything. The default handles this situation. If the charts are in multiple canvases and do not overlap, you should set the `QCRTGraphTS.ChartView.overlapCanvasDefault = false`.

```
QCRTGraphTS.ChartView.overlapCanvasDefault = false;
```

This only needs to be done once, at the very beginning of the first chart of the HTML page. Before the `ChartView` chart constructor (ie. `new ChartView...`, or one of the many others) Since it is static, it will apply to all of the charts in that page.

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
```

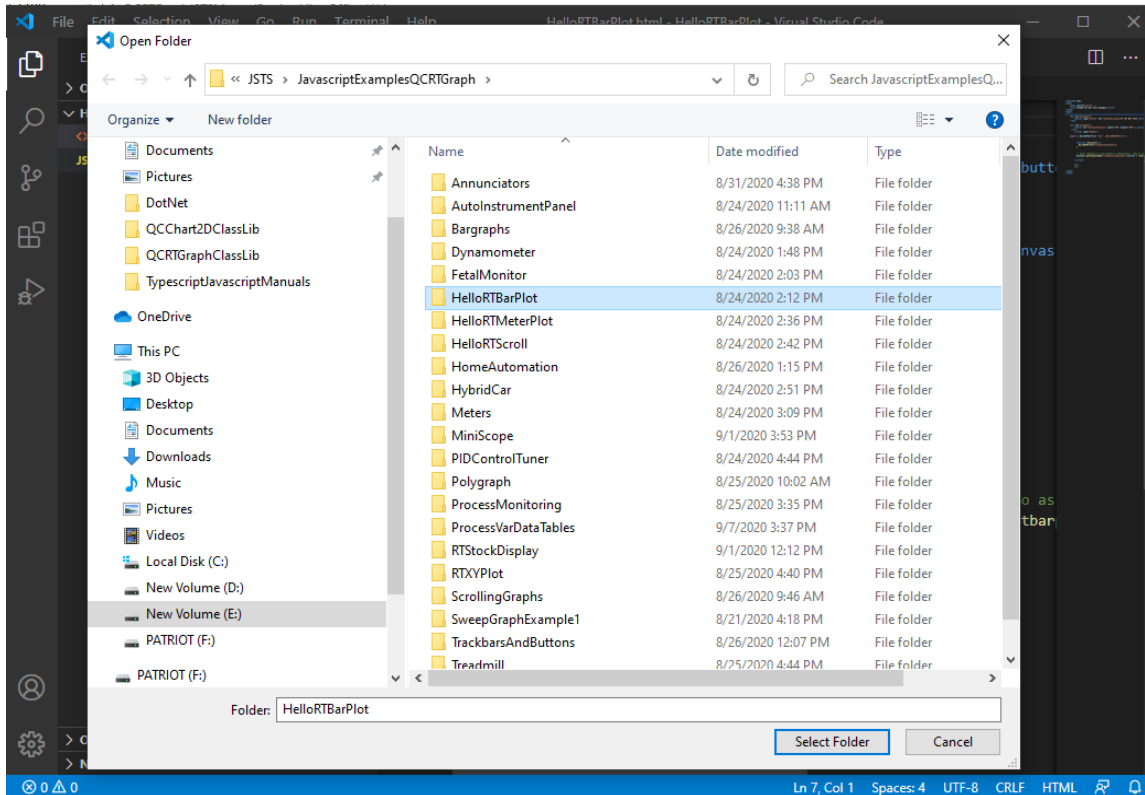
from the command prompt. If you need to install npm, or upgrade it, you will find resources on the web which tell you how to do that.

Using JavaScript Only

While we recommend that you use a TypeScript intermediary to interact with the `QCRTGraphTS` library, you can also call it directly from JavaScript. While `QCRTGraphTS` 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/JavascriptExamplesQCRTGraph/` folder. *We will use the folder name `HelloRTBarPlot`. Since we include the `HelloRTBarPlot` folder as one of our examples, if you plan on following this tutorial exactly, just rename the existing `HelloRTBarPlot` folder something else, `HelloRTBarPlot2` for example.* In its simplest form, that folder will contain just two files: your test HTML page (`HelloRTBarPlot.html`) and the JavaScript source page (`HelloRTBarPlot.js`). In this example, the `HelloRTBarPlot.html` page can just be a bare-bones HTML page, the same as our `JavascriptExamplesQCRTGraph/HelloRTBarPlot.html`. The only difference is in the embedded filename used to access the `HelloRTBarPlot.js` JavaScript file.

From your JavaScript Development Environment (Visual Studio Code in our case), open the HelloRTBarPlot folder using the IDE File | Open Folder option, and select the HelloRTBarPlot folder you created.



Next, create two new files in the HelloRTBarPlot directory: HelloRTBarPlot.html and HelloRTBarPlot.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 HelloRTBarPlot.html and HelloRTBarPlot.js. So you should end up with a folder named HelloRTBarPlot containing two blank files, HelloRTBarPlot.html and HelloRTBarPlot.js.

```

6 <body>
7
8 <div id="buttondiv">
9   <button type="button" id="rtbarplot_menuitem">RT Bar Plot</button>
10 </div>
11
12 <div id="canvasdiv">
13   <canvas id="rtbarplotCanvas1" width="800" height="600"></canvas>
14 </div>
15   <script type="module">
16
17 import { BuildRTBarPlot} from './HelloRTBarPlot.js';
18
19
20   function rtbarplot() {
21     BuildRTBarPlot("rtbarplotCanvas1");
22   }
23
24
25   // Since loading of script module is asynchronous, need to as
26   document.getElementById("rtbarplot_menuitem").onclick = rtbarplot;
27
28 </script>
29
30 <br>
31 <br>
32
33 </body>
34 </html>

```

Copy the code below into the HelloRTBarPlot.html file.

```

<!DOCTYPE html>
<head>
  <meta charset="utf-8" />
  <title>Simple RT Bar Plot Example</title>
</head>
<body>

  <div id="buttondiv">
    <button type="button" id="rtbarplot_menuitem">RT Bar Plot</button>
  </div>

  <div id="canvasdiv">
    <canvas id="rtbarplotCanvas1" width="800" height="600"></canvas>
  </div>
  <script type="module">

import { BuildRTBarPlot} from './HelloRTBarPlot.js';

    function rtbarplot() {
      BuildRTBarPlot("rtbarplotCanvas1");
    }

```

```

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

</script>

<br>
<br>

</body>
</html>

```

Note that the import statement looks to the current directory for the HelloRTBarPlot.js file.

```
import { BuildRTBarPlot } from './HelloRTBarPlot.js';
```

When the button (id = rtbarplot_menuitem) is pressed, it triggers a call to the BuildRTBarPlot function imported from the HelloRTBarPlot.js file. Note that the event handler (rtbarplot) for the button:

```
document.getElementById("rtbarplot_menuitem").onclick = rtbarplot;
```

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 *rtbarplot* 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 *rtbarplot* function has been defined, it will work.

Save the HelloRTBarPlot.html file and go to the HelloRTBarPlot.js file. You can copy the code below into that file.

```

import * as QCRTGraphTS from '../QCRTGraphTS/qcrtgraphts.js';

var EngineRPM1Value = 3500;
var EngineRPM1 = new QCRTGraphTS.RTPProcessVar();
var timerId = 0;
var timercount = 0;
var timercountlimit = 500;
var chartVu = null;

async function TimerTickProc() {
    if (chartVu) {
        EngineRPM1Value = 5000 * QCRTGraphTS.ChartSupport.getRandomDouble();
        EngineRPM1Value = Math.max(EngineRPM1Value, 100);
        EngineRPM1Value = Math.min(EngineRPM1Value, 5000);
        EngineRPM1.setCurrentValueNumber(EngineRPM1Value);
        chartVu.updateDraw();
    }
    timercount++;
    if (timercount > timercountlimit)
        StopTimer();
}

async function StartTimer(milliseconds, count) {
    if (timerId != 0)
        StopTimer();
    timercountlimit = count;
    timerId = setInterval(() => TimerTickProc(), milliseconds);
}

```

```

async function StopTimer() {
    clearInterval(timerId);
    timerId = 0;
}
function InitializeProcessVar() {
    let lowrpmalarm =
    QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_LOWERTHAN, 500);
    lowrpmalarm.AlarmMessage = "Low RPM";
    lowrpmalarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.BLUE;
    lowrpmalarm.AlarmTextColor = QCRTGraphTS.ChartColor.BLUE;
    let highrpmalarm =
    QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_GREATERTHAN, 4000);
    highrpmalarm.AlarmMessage = "High RPM";
    highrpmalarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.RED;
    highrpmalarm.AlarmTextColor = QCRTGraphTS.ChartColor.RED;
    let shutdownrpmalarm =
    QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_GREATERTHAN, 4900);
    shutdownrpmalarm.AlarmMessage = "Take Cover !!!";
    shutdownrpmalarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.WHITE;
    shutdownrpmalarm.AlarmTextColor = QCRTGraphTS.ChartColor.WHITE;
    EngineRPM1 = QCRTGraphTS.RTPProcessVar.newRTPProcessVar("RPM",
    QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN));
    EngineRPM1.MinimumValue = 0;
    EngineRPM1.MaximumValue = 5000;
    EngineRPM1.DefaultMinimumDisplayValue = 0;
    EngineRPM1.DefaultMaximumDisplayValue = 5000;
    EngineRPM1.addAlarm(lowrpmalarm);
    EngineRPM1.addAlarm(highrpmalarm);
    EngineRPM1.addAlarm(shutdownrpmalarm);
    EngineRPM1.setCurrentValueNumber(EngineRPM1Value);
}
export async function BuildRTBarPlot(canvasid) {
    let htmlcanvas = document.getElementById(canvasid);
    chartVu = QCRTGraphTS.ChartView.newChartView(htmlcanvas);
    if (!chartVu) return;
    chartVu.setPreferredSize(800, 600);

    InitializeProcessVar();

    let theFont;
    let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0.0, 0.0, 1.0,
    5000.0);
    pTransform1.setGraphBorderDiagonal(0.3, 0.15, 0.8, 0.6);
    let background = QCRTGraphTS.Background.newBackground(pTransform1,
    QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
    chartVu.addChartObject(background);
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.GREEN, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN);
    let barwidth = 1.0, barbase = 0.0;
    let barjust = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
    let barorient = QCRTGraphTS.ChartConstants.VERT_DIR;
    let baraxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
    QCRTGraphTS.ChartConstants.Y_AXIS);
    baraxis.calcAutoAxis();
    chartVu.addChartObject(baraxis);
    let barAxisLab = QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(baraxis);
    chartVu.addChartObject(barAxisLab);

    let barplot = QCRTGraphTS.RTBarIndicator.newRTBarIndicatorCoordsProcessVarAttrib(pTransform1,
    EngineRPM1, barwidth, barbase, attrib1, barjust, barorient);
    barplot.SegmentSpacing = 400;
    barplot.SegmentWidth = 250;
    barplot.IndicatorBackground.setColor(QCRTGraphTS.ChartColor.GRAY); // = new
    ChartAttribute(ChartColor.Black, 1, DashStyle.Solid, ChartColor.Black);
    barplot.SegmentValueRoundMode = QCRTGraphTS.ChartConstants.RT_CEILING_VALUE;
    barplot.SegmentCornerRadius = 0;
    barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SEGMENTED_SUBTYPE;
    let baralarms = QCRTGraphTS.RTAlarmIndicator.newRTAlarmIndicatorAxisRTPlot(baraxis, barplot);
    chartVu.addChartObject(baralarms);

```

```

    let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let paneltagmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
    let panelmeter =
QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeterCoordsAttrib(pTransform1,
panelmeterattrib);
    panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MIN;
    panelmeter.NumericTemplate.TextBgMode = true;
    panelmeter.NumericTemplate.LineColor = QCRTGraphTS.ChartColor.SPRINGGREEN;
    panelmeter.NumericTemplate.TextFont = QCRTGraphTS.ChartFont.newChartFont("Arial",
QCRTGraphTS.ChartFont.REGULAR, 18);
    panelmeter.NumericTemplate.DecimalPos = 0;
    panelmeter.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
    barplot.addPanelMeter(panelmeter);
    let panelmeter2 = QCRTGraphTS.RTAlarmPanelMeter.newRTAlarmPanelMeterCoordsAttrib(pTransform1,
panelmeterattrib);
    panelmeter2.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
    panelmeter2.AlarmTemplate.TextBgMode = true;
    panelmeter2.AlarmTemplate.LineColor = QCRTGraphTS.ChartColor.SPRINGGREEN;
    panelmeter2.AlarmTemplate.TextFont = QCRTGraphTS.ChartFont.newChartFont("Arial",
QCRTGraphTS.ChartFont.BOLD, 18);
    panelmeter2.setPositionReference(panelmeter);
    panelmeter2.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
    barplot.addPanelMeter(panelmeter2);
    let panelmeter3 =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsAttribStringType(pTransform1,
paneltagmeterattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
    panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MAX;
    panelmeter3.StringTemplate.TextBgMode = true;
    panelmeter3.StringTemplate.LineColor = QCRTGraphTS.ChartColor.BLACK;
    panelmeter3.StringTemplate.TextFont = QCRTGraphTS.ChartFont.newChartFont("Arial",
QCRTGraphTS.ChartFont.REGULAR, 18);
    panelmeter3.PanelMeterNudge = QCRTGraphTS.ChartPoint2D.newChartPoint2D(0, -6);
    panelmeter3.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_INDICATOR_COLOR_NO_ALARM_CHANGE;
    barplot.addPanelMeter(panelmeter3);
    chartVu.addChartObject(barplot);
    chartVu.updateDraw();
    StartTimer(1000, 500);
}

```

Note that the QCRTGraphTS library is being imported from its location in the file system, relative to the HelloRTBarPlot.js file.

```
import * as QCRTGraphTS from '.././QCRTGraphTS/qcrtgraphts.js';
```

The “import * as QCRTGraphTS” in the import statement assigns all classes, functions, properties and constants in the software to the namespace QCRTGraphTS, so you must preface any reference to the formal name of any class, static method within a class, or static constants, found within the QCRTGraph library, by explicitly referencing that namespace.

```

let htmlcanvas = document.getElementById(canvasid);
chartVu = QCRTGraphTS.ChartView.newChartView(htmlcanvas);
if (!chartVu) return;
chartVu.setPreferredSize(800, 600);
let theFont;
let pTransform1 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0.0, 0.0, 1.0,
5000.0);
pTransform1.setGraphBorderDiagonal(0.3, 0.15, 0.8, 0.6);

```



```
let background = QCRTGraphTS.Background.newBackground(pTransform1,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
chartVu.addChartObject(background);
```

The main function in the HelloRTBarPlot.js file is BuildRTBarPlot. It first creates an instance of a ChartView object using a static constructor.:

```
export async function BuildRTBarPlot(canvasid) {
  let htmlcanvas = document.getElementById(canvasid);
  chartVu = QCRTGraphTS.ChartView.newChartView(htmlcanvas);
  if (!chartVu) return;
  chartVu.setPreferredSize(800, 600);
```

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 it defines the RTPProcessVar objects it plans to plot. It does that in the call to InitializeProcessVar, which defines the RTPProcessVar variable EngineRPM1.

```
function InitializeProcessVar() {
  let lowrpmalarm =
QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_LOWERTHAN, 500);
  lowrpmalarm.AlarmMessage = "Low RPM";
  lowrpmalarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.BLUE;
  lowrpmalarm.AlarmTextColor = QCRTGraphTS.ChartColor.BLUE;
  let highrpmalarm =
QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_GREATERTHAN, 4000);
  highrpmalarm.AlarmMessage = "High RPM";
  highrpmalarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.RED;
  highrpmalarm.AlarmTextColor = QCRTGraphTS.ChartColor.RED;
  let shutdownrpmalarm =
QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_GREATERTHAN, 4900);
  shutdownrpmalarm.AlarmMessage = "Take Cover !!!";
  shutdownrpmalarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.WHITE;
  shutdownrpmalarm.AlarmTextColor = QCRTGraphTS.ChartColor.WHITE;
  EngineRPM1 = QCRTGraphTS.RTPProcessVar.newRTPProcessVar("RPM",
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN));
  EngineRPM1.MinimumValue = 0;
  EngineRPM1.MaximumValue = 5000;
  EngineRPM1.DefaultMinimumDisplayValue = 0;
  EngineRPM1.DefaultMaximumDisplayValue = 5000;
  EngineRPM1.addAlarm(lowrpmalarm);
  EngineRPM1.addAlarm(highrpmalarm);
  EngineRPM1.addAlarm(shutdownrpmalarm);
  EngineRPM1.setCurrentValueNumber(EngineRPM1Value);
}
```

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 = QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0.0, 0.0, 1.0,
5000.0);
pTransform1.setGraphBorderDiagonal(0.3, 0.15, 0.8, 0.6);
```

Next you define the graphical objects you want displayed in the graph. This would include background objects, axes, axes labels, panel meters, 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 = QCRTGraphTS.Background.newBackground(pTransform1,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
    chartVu.addChartObject(background);
    let attrib1 = QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.GREEN, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN);
    let barwidth = 1.0, barbase = 0.0;
    let barjust = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
    let barorient = QCRTGraphTS.ChartConstants.VERT_DIR;
    let baraxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
QCRTGraphTS.ChartConstants.Y_AXIS);
    baraxis.calcAutoAxis();
    chartVu.addChartObject(baraxis);
    let barAxisLab = QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(baraxis);
    chartVu.addChartObject(barAxisLab);

    let barplot = QCRTGraphTS.RTBarIndicator.newRTBarIndicatorCoordsProcessVarAttrib(pTransform1,
EngineRPM1, barwidth, barbase, attrib1, barjust, barorient);
    barplot.SegmentSpacing = 400;
    barplot.SegmentWidth = 250;
    barplot.IndicatorBackground.setColor(QCRTGraphTS.ChartColor.GRAY); // = new
ChartAttribute(ChartColor.Black, 1, DashStyle.Solid, ChartColor.Black);
    barplot.SegmentValueRoundMode = QCRTGraphTS.ChartConstants.RT_CEILING_VALUE;
    barplot.SegmentCornerRadius = 0;
    barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SEGMENTED_SUBTYPE;
    let baralarms = QCRTGraphTS.RTAlarmIndicator.newRTAlarmIndicatorAxisRTPlot(baraxis, barplot);
    chartVu.addChartObject(baralarms);
    let panelmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let paneltagmeterattrib =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);
    let panelmeter =
QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeterCoordsAttrib(pTransform1,
panelmeterattrib);
    panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MIN;
    panelmeter.NumericTemplate.TextBgMode = true;
    panelmeter.NumericTemplate.LineColor = QCRTGraphTS.ChartColor.SPRINGGREEN;
    panelmeter.NumericTemplate.TextFont = QCRTGraphTS.ChartFont.newChartFont("Arial",
QCRTGraphTS.ChartFont.REGULAR, 18);
    panelmeter.NumericTemplate.DecimalPos = 0;
    panelmeter.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
    barplot.addPanelMeter(panelmeter);
    let panelmeter2 = QCRTGraphTS.RTAlarmPanelMeter.newRTAlarmPanelMeterCoordsAttrib(pTransform1,
panelmeterattrib);
    panelmeter2.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
    panelmeter2.AlarmTemplate.TextBgMode = true;
    panelmeter2.AlarmTemplate.LineColor = QCRTGraphTS.ChartColor.SPRINGGREEN;
    panelmeter2.AlarmTemplate.TextFont = QCRTGraphTS.ChartFont.newChartFont("Arial",
QCRTGraphTS.ChartFont.BOLD, 18);
    panelmeter2.setPositionReference(panelmeter);
    panelmeter2.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
    barplot.addPanelMeter(panelmeter2);
    let panelmeter3 =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsAttribStringType(pTransform1,
paneltagmeterattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
    panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MAX;
    panelmeter3.StringTemplate.TextBgMode = true;
    panelmeter3.StringTemplate.LineColor = QCRTGraphTS.ChartColor.BLACK;
    panelmeter3.StringTemplate.TextFont = QCRTGraphTS.ChartFont.newChartFont("Arial",
QCRTGraphTS.ChartFont.REGULAR, 18);
    panelmeter3.PanelMeterNudge = QCRTGraphTS.ChartPoint2D.newChartPoint2D(0, -6);
    panelmeter3.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_INDICATOR_COLOR_NO_ALARM_CHANGE;
    barplot.addPanelMeter(panelmeter3);

```

```
chartVu.addChartObject(barplot);
chartVu.updateDraw();
```

Last is a call to a simple timer used to update the real-time bar chart using a 1 second interval.

```
StartTimer(1000, 500);
```

The timer is implemented at the top of the HelloRTBarPlot.js file using the functions:

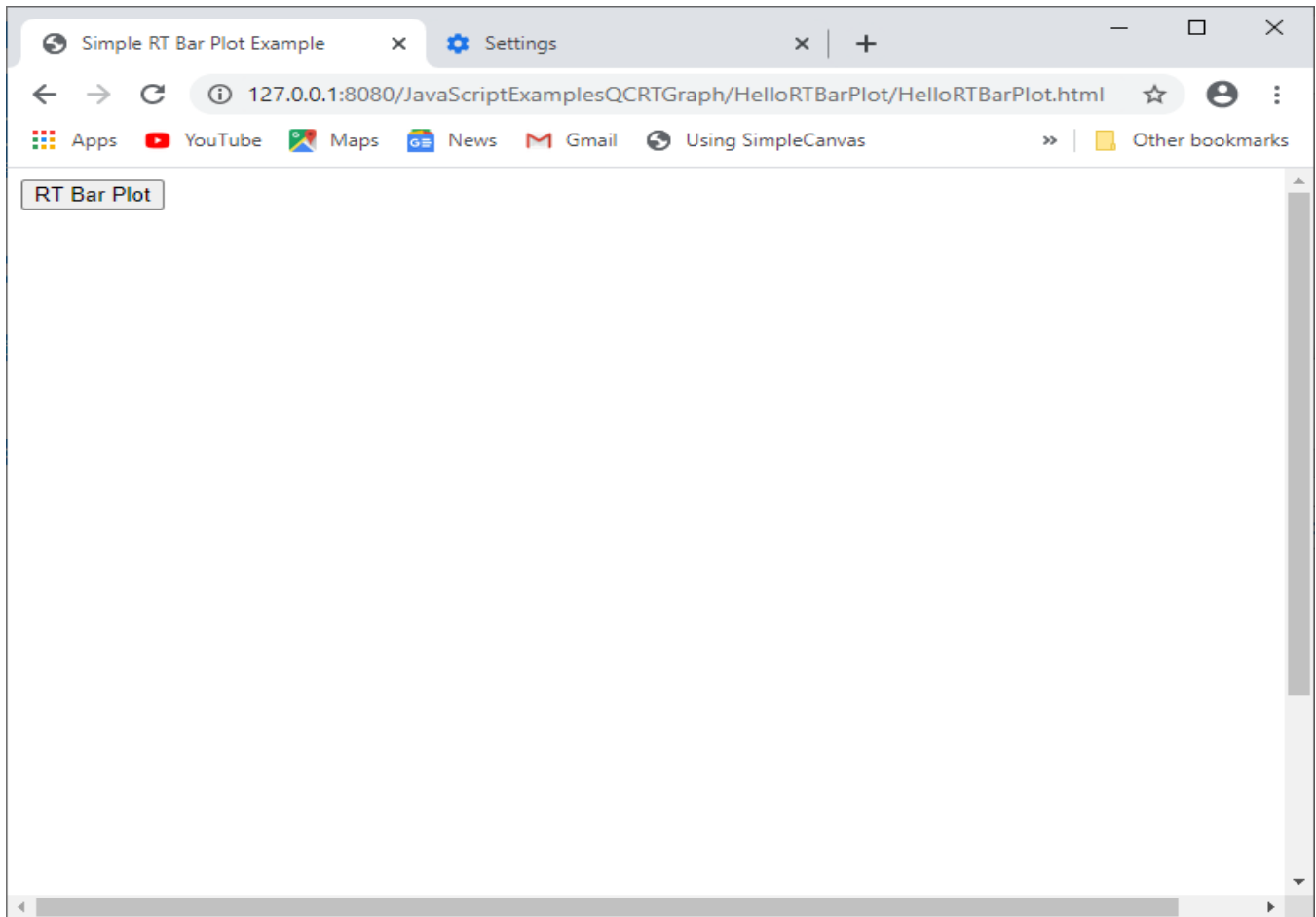
```
.
async function TimerTickProc() {
  if (chartVu) {
    EngineRPM1Value = 5000 * QCRTGraphTS.ChartSupport.getRandomDouble();
    EngineRPM1Value = Math.max(EngineRPM1Value, 100);
    EngineRPM1Value = Math.min(EngineRPM1Value, 5000);
    EngineRPM1.setCurrentValueNumber(EngineRPM1Value);
    chartVu.updateDraw();
  }
  timercount++;
  if (timercount > timercountlimit)
    StopTimer();
}
async function StartTimer(milliseconds, count) {
  if (timerId != 0)
    StopTimer();
  timercountlimit = count;
  timerId = setInterval(() => TimerTickProc(), milliseconds);
}
async function StopTimer() {
  clearInterval(timerId);
  timerId = 0;
}
```

The call to StartTimer(1000, 500) starts the timer with a 1000 ms interval (1 second) and stops the timer after a count of 500 updates has been reached. In the timer update function (TimerTickProc), the EngineRPM1 RTPProcessVar object is updated with a new value using the call to the function EngineRPM1.setCurrentValueNumber. The real-time bar plots is updated to reflect this new value using the call to chartVu.updateDraw(). No matter how many bar plots (or any other real-time objects such as meters, scrolling graphics, annunciators, or panelmeter) you have in the chart, you first update all of the associated RTPProcessVar objects first, then call the chartVu.updateDraw function just once. All of the real-time graph objects will be updated simultaneously with a single call.

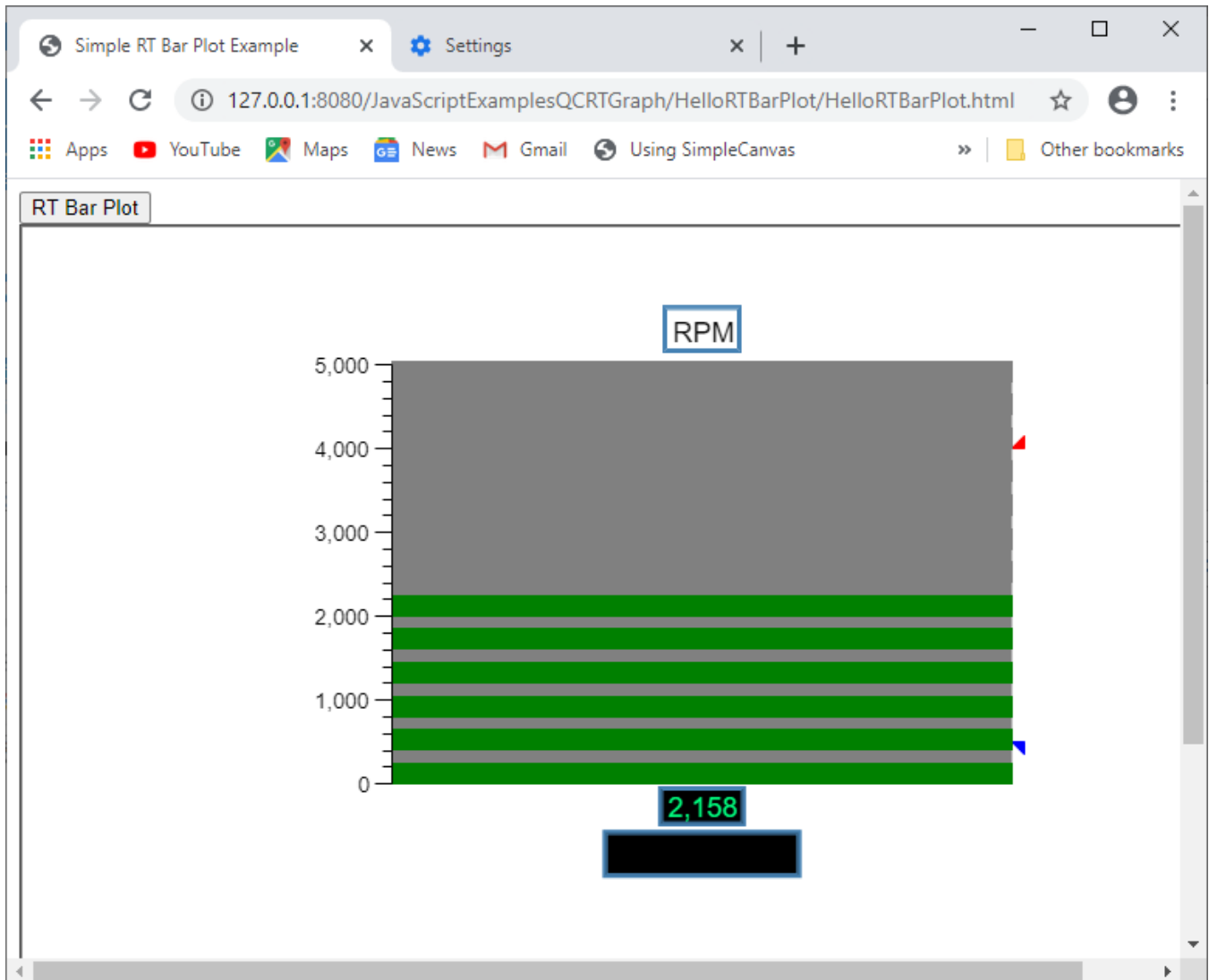
Assuming you have the two files setup correctly, and you have started some sort of test server which references the Quinn-Curtis/JSTS folder as its root directory, you should be able to run the test program by setting a browser to load the file at the URL:

<http://127.0.0.1:8080/JavascriptExamplesQCRTGraph/HelloRTBarPlot/HelloRTBarPlot.html>

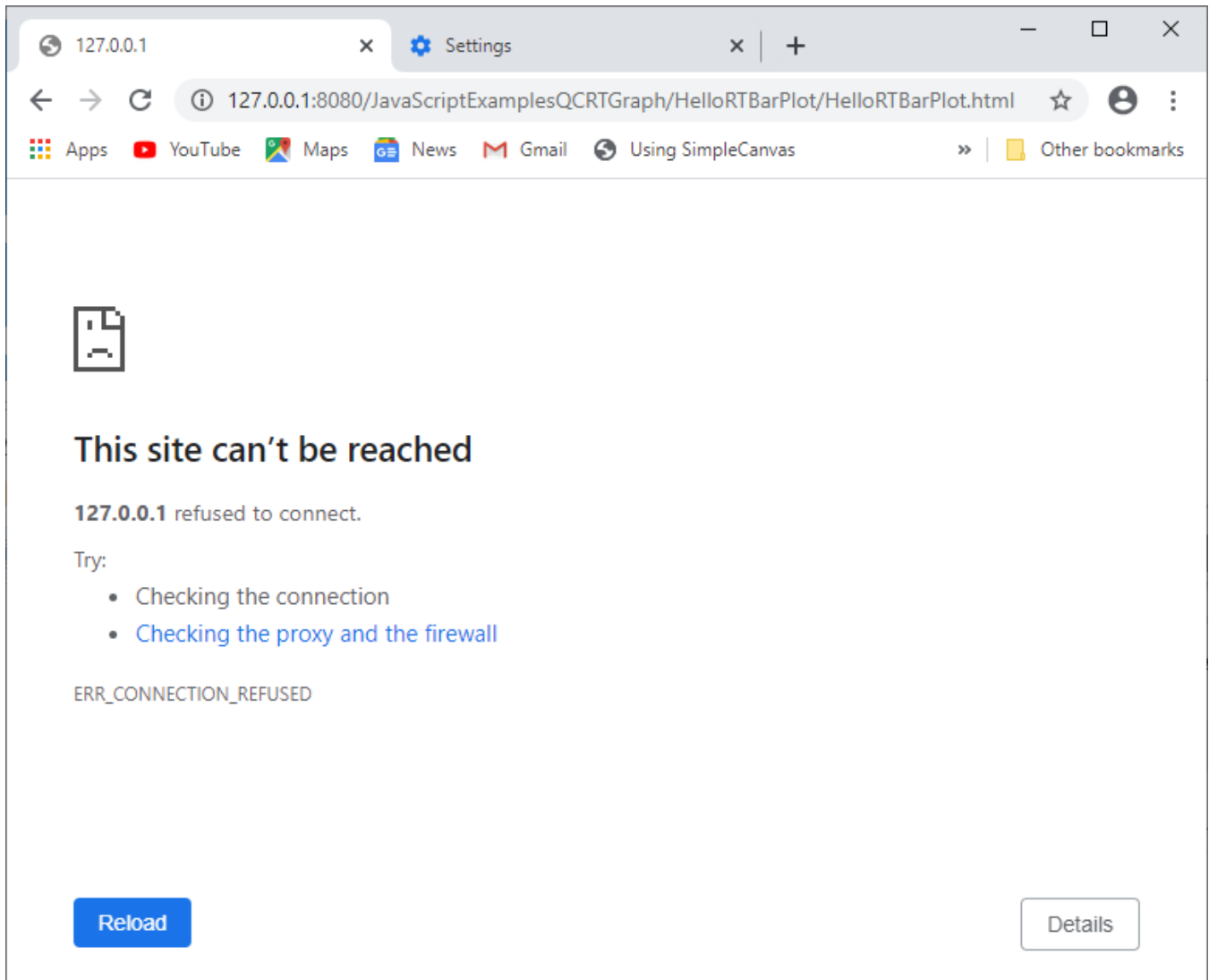
The result will be the following display:



showing just the button. Click the button and the chart will appear.



\nIf you get a screen that looks like:



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:

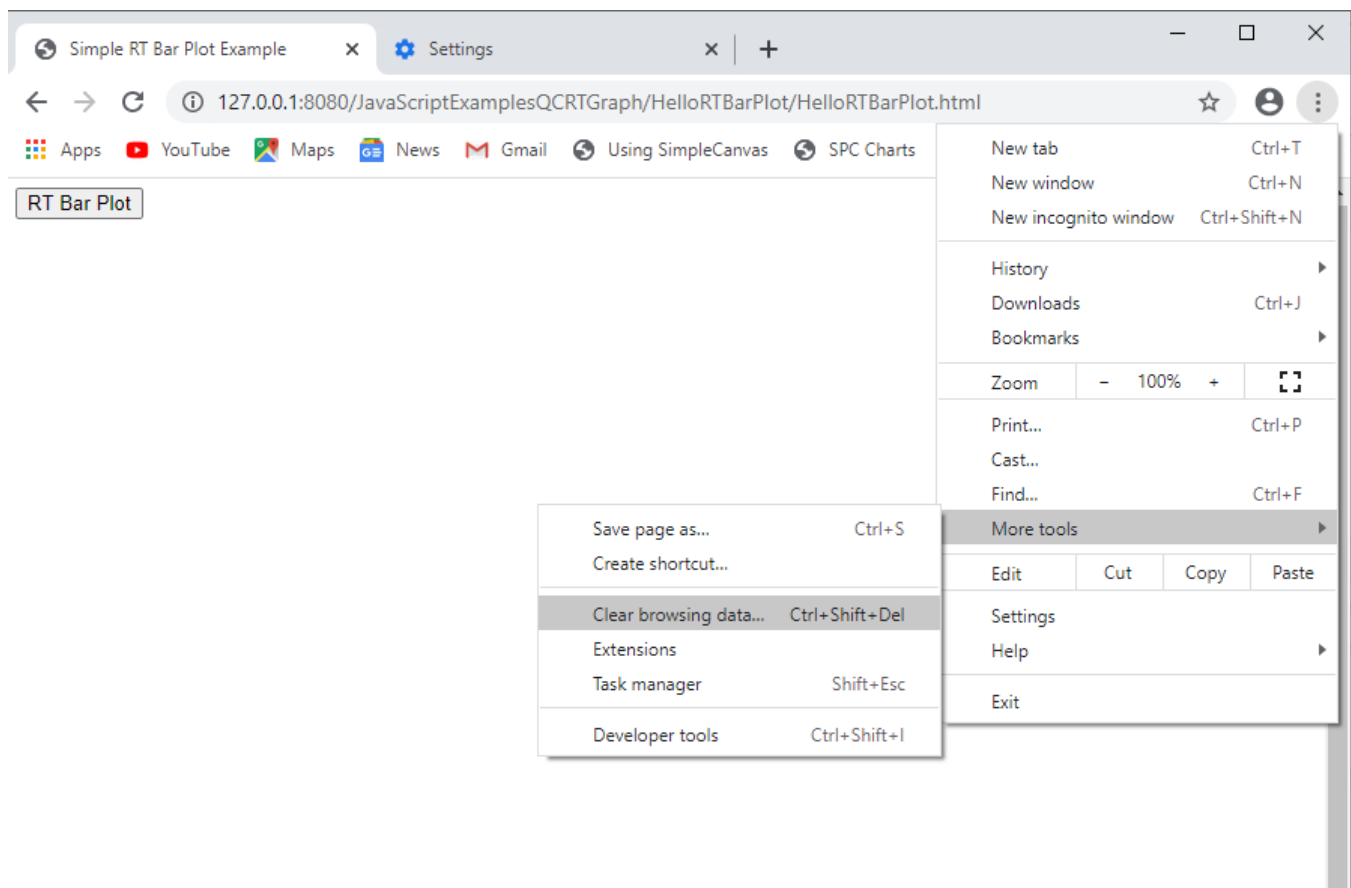
```
http-server ./
```

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:

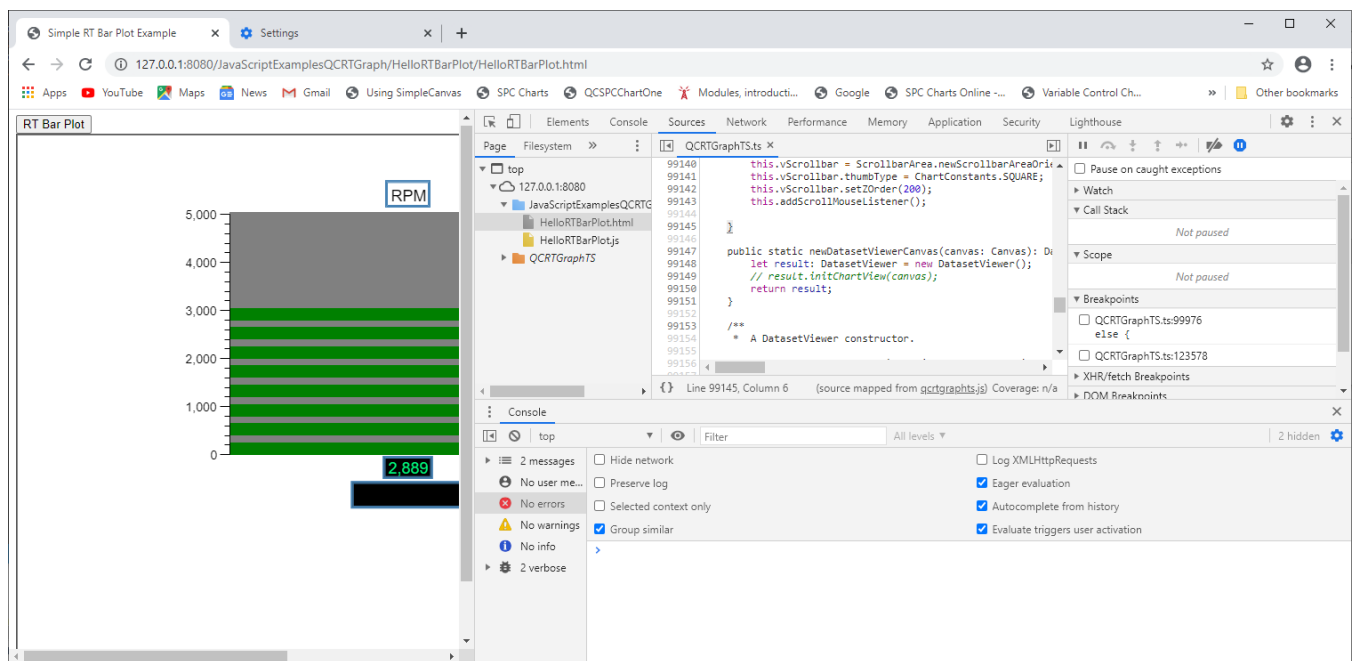


Or, the preferred method, is that you can turn off caching from within the browser Development Mode. Here is a good discussion of that option: <https://nicholasbering.ca/tools/2016/10/09/devtools-disable-caching/>. That article discusses the options for Chrome, Firefox and Safari, but Microsoft Edge has a similar option. The browser cache only clears if the browser Developer Debugger is visible in the broser window. If the debugger closes, so does clearing of the chache.

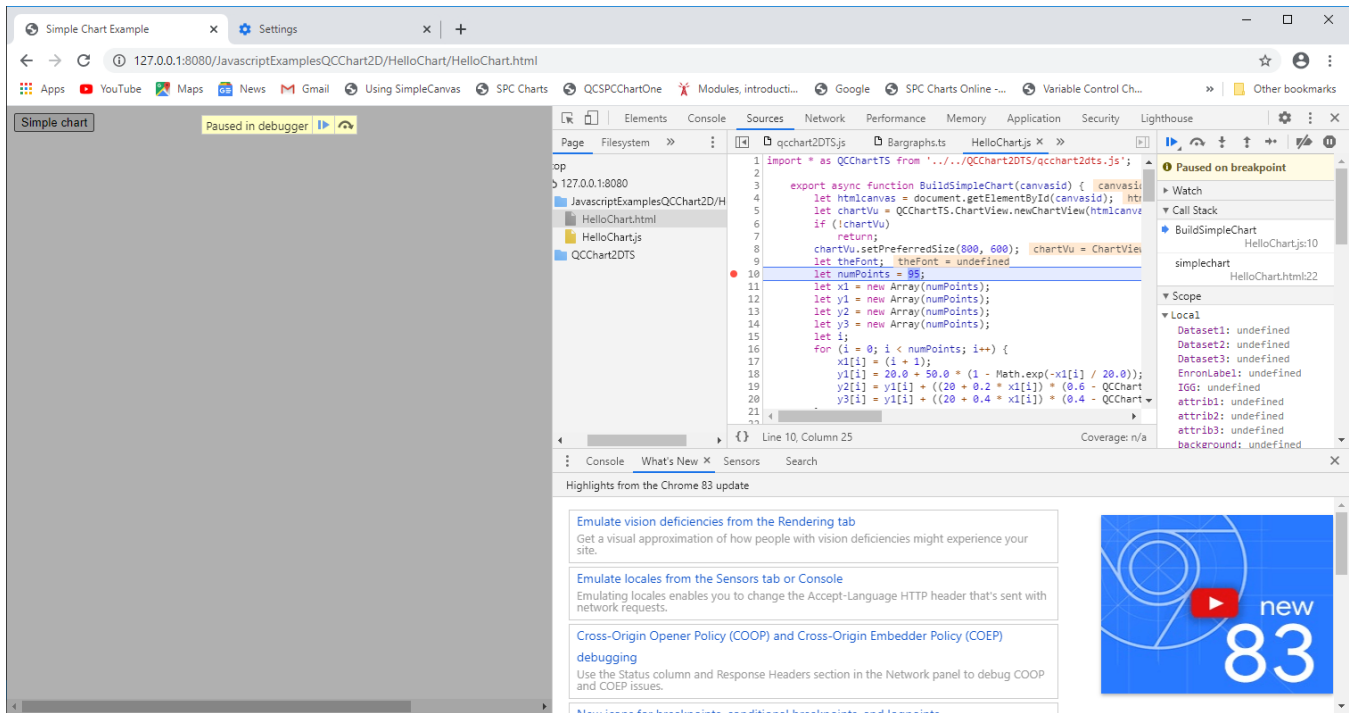
Debugging

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 HelloRTBarPlot.html file you should see the RT Bar Plot button. Press F12 and the browsers integrated debugger will appear on the right. The debugger portion of the window will look something like:



You should be able to select either the HelloRTBarPlot.js file, or the HelloRTBarPlot.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 HelloRTBarPlot.js file, and then click the RT Bar Plot in the left hand side of the browser, program execution will stop in the code window when the breakpoint is hit.



Using TypeScript

The QCRTGraphTS 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 qchartgraphs.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 qchartgraphs.js file is located in the Quinn-Curtis/JSTS/QCRTGraphTS folder. So your TypeScript source files which reference the QCRTGraphTS library must import it from that location. We specify the location of the qchartgraphs.js file relative to the source folder of the file we will be working with, using:

```
import * as QCRTGraphTS from '.././QCRTGraphTS/qchartgraphs.js';
```

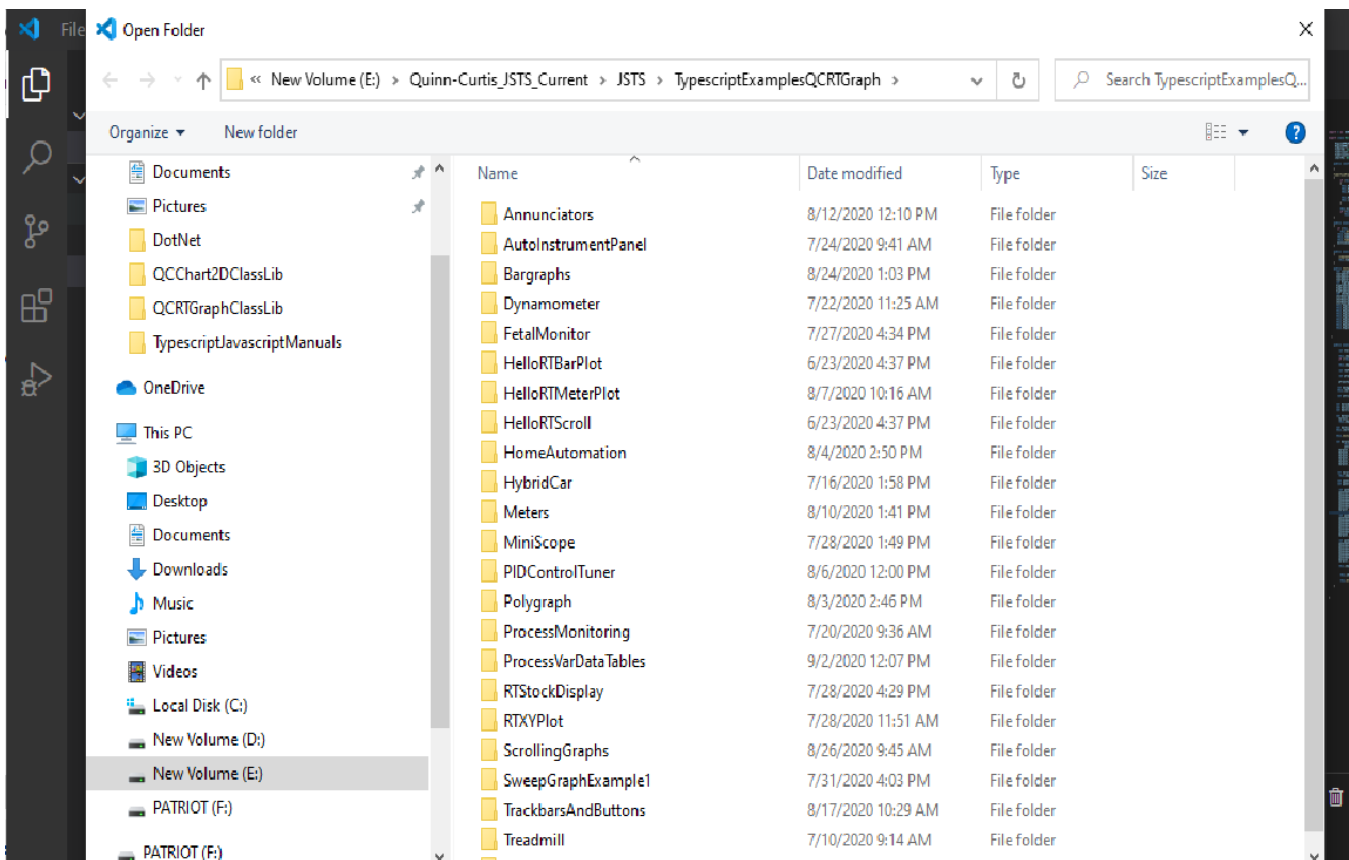
Note: The entire QCChart2D charting library is embedded in the qchartgraphs.js file. Since it is part of that file, it uses the same namespace as the QCRTGraph classes, functions, and constants. So when you are using the QCRTGraph library, you only need to use one namespace identifier in the import statement, QCRTGraphTS above. All of the QCChart2D and QCRTGraph functions will be within that namespace.

Start off by creating a folder unique to your intended code, under our JSTS/TypescriptExamplesQCRTGraph/ folder. We will use the folder name HelloRTBarPlot. Since we

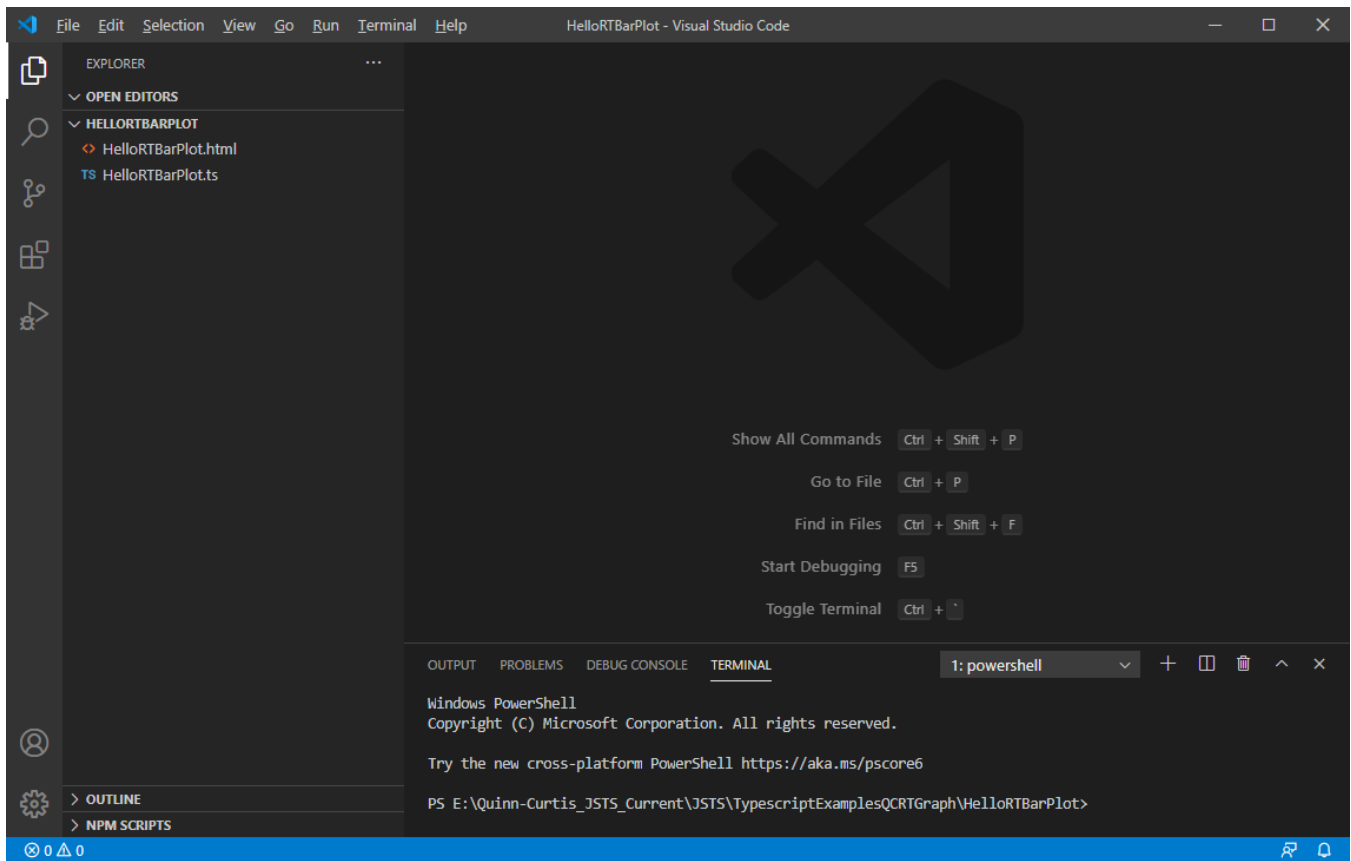
include the HelloRTBarPlot folder as one of our examples, if you plan on following this tutorial exactly, just rename the existing HelloRTBarPlot folder something else, HelloRTBarPlot2 for example. In its simplest form, that folder will contain just two files: your test HTML page (HelloRTBarPlot.html) and the TypeScript source page (HelloRTBarPlot.ts). In this example, the HelloRTBarPlot.html page can just be an bare-bones HTML page, the same as our

TypescriptExamplesQCRTGraph/HelloRTBarPlot/HelloRTBarPlot.html. The only difference is in the embedded filename used to access the HelloRTBarPlot.js JavaScript file. Even though your HelloRTBarPlot.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.

From your JavaScript Development Environment (Visual Studio Code in our case), open the HelloRTBarPlot folder using the IDE File | Open Folder option, and select the HelloRTBarPlot folder you created.



Next, create two new files in the HelloRTBarPlot directory: HelloRTBarPlot.html and HelloRTBarPlot.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 HelloRTBarPlot.html and HelloRTBarPlot.ts. So you should end up with a folder named HelloRTBarPlot containing two blank file, HelloRTBarPlot.html and HelloRTBarPlot.ts.



Copy the code below into the HelloRTBarPlot.html file.

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

  <div id="buttondiv">
    <button type="button" id="rtbarplot_menuitem">RT Bar Plot</button>
  </div>

  <div id="canvasdiv">
    <canvas id="rtbarplotCanvas1" width="800" height="600"></canvas>
  </div>
  <script type="module">

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

var rtplot = new HelloRTBarPlot();

    function rtbarplot() {
      rtplot.BuildRTBarPlot("rtbarplotCanvas1");
    }

  </script>
</body>
```

```

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

</script>

<br>
<br>

</body>
</html>

```

Note that the import statement looks to the current directory for the HelloRTBarPlot.js file.

```
import { HelloRTBarPlot } from './HelloRTBarPlot.js';
```

The import statement loads the HelloRTBarPlot.js file and makes the HelloRTBarPlot class inside available as the class HelloRTBarPlot. When that is completed, an instance of the HelloRTBarPlot class found in that file, is instantiated using new HelloRTBarPlot(). Until you compile the HelloRTBarPlot project, the HelloRTBarPlot.js file will not exist. When the project is compiled, the HelloRTBarPlot.ts file is compiled into the HelloRTBarPlot.js file and placed in the same folder, HelloRTBarPlot, from which it is loaded when needed.

When the button (id = rtbarplot_menuitem) is pressed, it triggers a call to the BuildRTBarPlot function imported from the HelloRTBarPlot.js file. Note that the event handler (rtbarplot) for the button:

```
document.getElementById("rtbarplot_menuitem").onclick = rtbarplot;
```

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 rtbarplot 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 rtbarplot function has been defined, it will work.

Save the HelloRTBarPlot.html file and go to the HelloRTBarPlot.ts file. You can copy the code below into that file.

```

import * as QCRTGraphTS from '../../QCRTGraphTS/qcrtgraphts.js';

export class HelloRTBarPlot {

    EngineRPM1Value: number = 3500;
    EngineRPM1: QCRTGraphTS.RTPProcessVar = new QCRTGraphTS.RTPProcessVar();
    timerId: number = 0;
    timercount: number = 0;
    timercountlimit: number = 500;
    chartVu: QCRTGraphTS.ChartView | null = null;

    public constructor() {

    }

    TimerTickProc() {
        if (this.chartVu) {
            this.EngineRPM1Value = 5000 * QCRTGraphTS.ChartSupport.getRandomDouble();
            this.EngineRPM1Value = Math.max(this.EngineRPM1Value, 100);
            this.EngineRPM1Value = Math.min(this.EngineRPM1Value, 5000);

```

```

        this.EngineRPM1.setCurrentValueNumber(this.EngineRPM1Value);
        this.chartVu.updateDraw();
    }

    this.timercount++;
    if (this.timercount > this.timercountlimit)
        this.StopTimer();
}

public async StartTimer(milliseconds: number, count: number) {
    if (this.timerId != 0)
        this.StopTimer();
    this.timercountlimit = count;
    this.timerId = setInterval(() =>
        this.TimerTickProc(),
        milliseconds);
}

public async StopTimer() {
    clearInterval(this.timerId);
    this.timerId = 0;
}

public InitializeProcessVar() {
    let lowrpmalarm: QCRTGraphTS.RTAlarm =
QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_LOWER_THAN, 500);
    lowrpmalarm.AlarmMessage = "Low RPM";
    lowrpmalarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.BLUE;
    lowrpmalarm.AlarmTextColor = QCRTGraphTS.ChartColor.BLUE;
    let highrpmalarm: QCRTGraphTS.RTAlarm =
QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_GREATER_THAN, 4000);
    highrpmalarm.AlarmMessage = "High RPM";
    highrpmalarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.RED;
    highrpmalarm.AlarmTextColor = QCRTGraphTS.ChartColor.RED;
    let shutdownrpmalarm: QCRTGraphTS.RTAlarm =
QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_GREATER_THAN, 4900);
    shutdownrpmalarm.AlarmMessage = "Take Cover !!!";
    shutdownrpmalarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.WHITE;
    shutdownrpmalarm.AlarmTextColor = QCRTGraphTS.ChartColor.WHITE;
    this.EngineRPM1 = QCRTGraphTS.RTProcessVar.newRTProcessVar("RPM",
QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN));
    this.EngineRPM1.MinimumValue = 0;
    this.EngineRPM1.MaximumValue = 5000;
    this.EngineRPM1.DefaultMinimumDisplayValue = 0;
    this.EngineRPM1.DefaultMaximumDisplayValue = 5000;
    this.EngineRPM1.addAlarm(lowrpmalarm);
    this.EngineRPM1.addAlarm(highrpmalarm);
    this.EngineRPM1.addAlarm(shutdownrpmalarm);
    this.EngineRPM1.setCurrentValueNumber(this.EngineRPM1Value);
}

public async BuildRTBarPlot(canvasid: string) {

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

    this.chartVu = QCRTGraphTS.ChartView.newChartView(htmlcanvas);
    if (!this.chartVu) return;

    this.chartVu.setPreferredSize(800, 600);

    let theFont: QCRTGraphTS.ChartFont;

    let pTransform1: QCRTGraphTS.CartesianCoordinates =
QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0.0, 0.0, 1.0, 5000.0);

    pTransform1.setGraphBorderDiagonal(0.3, 0.15, 0.8, 0.6);

```

```

    let background: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(pTransform1,
QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
    this.chartVu.addChartObject(background);

    let attrib1: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.GREEN, 1,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN);

    let barwidth: number = 1.0, barbase = 0.0;
    let barjust: number = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
    let barorient: number = QCRTGraphTS.ChartConstants.VERT_DIR;

    let baraxis: QCRTGraphTS.LinearAxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
QCRTGraphTS.ChartConstants.Y_AXIS);
    baraxis.calcAutoAxis();
    this.chartVu.addChartObject(baraxis);

    let barAxisLab: QCRTGraphTS.NumericAxisLabels =
QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(baraxis);
    this.chartVu.addChartObject(barAxisLab);

    this.InitializeProcessVar();

    let barplot: QCRTGraphTS.RTBarIndicator =
QCRTGraphTS.RTBarIndicator.newRTBarIndicatorCoordsProcessVarAttrib(pTransform1,
    this.EngineRPM1, barwidth, barbase,
    attrib1, barjust, barorient);
    barplot.SegmentSpacing = 400;
    barplot.SegmentWidth = 250;
    barplot.IndicatorBackground.setColor(QCRTGraphTS.ChartColor.GRAY); // = new
ChartAttribute(ChartColor.Black, 1, DashStyle.Solid, ChartColor.Black);
    barplot.SegmentValueRoundMode = QCRTGraphTS.ChartConstants.RT_CEILING_VALUE;
    barplot.SegmentCornerRadius = 0;
    barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SEGMENTED_SUBTYPE;

    let baralarms: QCRTGraphTS.RTAlarmIndicator =
QCRTGraphTS.RTAlarmIndicator.newRTAlarmIndicatorAxisRTPlot(baraxis, barplot);
    this.chartVu.addChartObject(baralarms);

    let panelmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
    let paneltagmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);

    let panelmeter: QCRTGraphTS.RTNumericPanelMeter =
QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeterCoordsAttrib(pTransform1,
    panelmeterattrib);
    panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MIN;
    panelmeter.NumericTemplate.TextBgMode = true;
    panelmeter.NumericTemplate.LineColor = QCRTGraphTS.ChartColor.SPRINGGREEN;
    panelmeter.NumericTemplate.TextFont = QCRTGraphTS.ChartFont.newChartFont("Arial",
QCRTGraphTS.ChartFont.REGULAR, 18);
    panelmeter.NumericTemplate.DecimalPos = 0;
    panelmeter.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
    barplot.addPanelMeter(panelmeter);

    let panelmeter2: QCRTGraphTS.RTAlarmPanelMeter =
QCRTGraphTS.RTAlarmPanelMeter.newRTAlarmPanelMeterCoordsAttrib(pTransform1, panelmeterattrib);
    panelmeter2.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
    panelmeter2.AlarmTemplate.TextBgMode = true;
    panelmeter2.AlarmTemplate.LineColor = QCRTGraphTS.ChartColor.SPRINGGREEN;
    panelmeter2.AlarmTemplate.TextFont = QCRTGraphTS.ChartFont.newChartFont("Arial",
QCRTGraphTS.ChartFont.BOLD, 18);
    panelmeter2.setPositionReference(panelmeter);

```

```

    panelmeter2.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
    barplot.addPanelMeter(panelmeter2);

    let panelmeter3: QCRTGraphTS.RTStringPanelMeter =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsAttribStringType(pTransform1,
paneltagmeterattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
    panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MAX;
    panelmeter3.StringTemplate.TextBgMode = true;
    panelmeter3.StringTemplate.LineColor = QCRTGraphTS.ChartColor.BLACK;
    panelmeter3.StringTemplate.TextFont = QCRTGraphTS.ChartFont.newChartFont("Arial",
QCRTGraphTS.ChartFont.REGULAR, 18);
    panelmeter3.PanelMeterNudge = QCRTGraphTS.ChartPoint2D.newChartPoint2D(0, -6);
    panelmeter3.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_INDICATOR_COLOR_NO_ALARM_CHANGE;
    barplot.addPanelMeter(panelmeter3);

    this.chartVu.addChartObject(barplot);

    this.chartVu.updateDraw();

    this.StartTimer(1000, 500);
}
}

```

In the `HelloRTBarPlot.ts` file, the body of the program is setup as a TypeScript class, named `HelloRTBarPlot`. It has two member functions: the public default constructor, the public `BuildRTBarPlot` member function which is called from the `HelloRTBarPlot.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 `HelloRTBarPlot`, found under `JavascriptExamplesQCRTGraph/HelloRTBarPlot`.

Note that the `QCRTGraphTS` library is being imported from its location in the file system, relative to the `HelloRTBarPlot.js` file.

```
import * as QCRTGraphTS from '../.. /QCRTGraphTS/qcrtgraphts.js';
```

The “`import * as QCRTGraphTS`” in the import statement assigns all classes, static functions within a class, properties and constants in the software to the namespace `QCRTGraphTS`, so you must preface any reference to the formal name of any classes or static constants, found within the `QCRTGraph` library by explicitly referencing that namespace.

```

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

this.chartVu = QCRTGraphTS.ChartView.newChartView(htmlcanvas);
if (!this.chartVu) return;

this.chartVu.setPreferredSize(800, 600);

let theFont: QCRTGraphTS.ChartFont;

let pTransform1: QCRTGraphTS.CartesianCoordinates =
QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0.0, 0.0, 1.0, 5000.0);

```

In broad terms, first you create an instance of a `ChartView` object using a static constructor.:

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

```

this.chartVu = QCRTGraphTS.ChartView.newChartView(htmlcanvas);
if (!this.chartVu) return;

this.chartVu.setPreferredSize(800, 600);

```

passing in the required parameters.

Next it defines the `RTProcessVar` objects it plans to plot. It does that in the call to `InitializeProcessVar`, which defines the `RTProcessVar` variable `EngineRPM1`.

```

public InitializeProcessVar() {
    let lowrpmalarm: QCRTGraphTS.RTAlarm =
    QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_LOWERTHAN, 500);
    lowrpmalarm.AlarmMessage = "Low RPM";
    lowrpmalarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.BLUE;
    lowrpmalarm.AlarmTextColor = QCRTGraphTS.ChartColor.BLUE;
    let highrpmalarm: QCRTGraphTS.RTAlarm =
    QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_GREATERTHAN, 4000);
    highrpmalarm.AlarmMessage = "High RPM";
    highrpmalarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.RED;
    highrpmalarm.AlarmTextColor = QCRTGraphTS.ChartColor.RED;
    let shutdownrpmalarm: QCRTGraphTS.RTAlarm =
    QCRTGraphTS.RTAlarm.newRTAlarmAlarmType(QCRTGraphTS.ChartConstants.RT_ALARM_GREATERTHAN, 4900);
    shutdownrpmalarm.AlarmMessage = "Take Cover !!!";
    shutdownrpmalarm.AlarmSymbolColor = QCRTGraphTS.ChartColor.WHITE;
    shutdownrpmalarm.AlarmTextColor = QCRTGraphTS.ChartColor.WHITE;
    this.EngineRPM1 = QCRTGraphTS.RTProcessVar.newRTProcessVar("RPM",
    QCRTGraphTS.ChartAttribute.newChartAttribute(QCRTGraphTS.ChartColor.GREEN, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN));
    this.EngineRPM1.MinimumValue = 0;
    this.EngineRPM1.MaximumValue = 5000;
    this.EngineRPM1.DefaultMinimumDisplayValue = 0;
    this.EngineRPM1.DefaultMaximumDisplayValue = 5000;
    this.EngineRPM1.addAlarm(lowrpmalarm);
    this.EngineRPM1.addAlarm(highrpmalarm);
    this.EngineRPM1.addAlarm(shutdownrpmalarm);
    this.EngineRPM1.setCurrentValueNumber(this.EngineRPM1Value);
}

```

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: QCRTGraphTS.CartesianCoordinates =
    QCRTGraphTS.CartesianCoordinates.newCartesianCoordinates(0.0, 0.0, 1.0, 5000.0);

pTransform1.setGraphBorderDiagonal(0.3, 0.15, 0.8, 0.6);

```

Next you define the graphical objects you want displayed in the graph. This would include background objects, axes, axes labels, panel meters, 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: QCRTGraphTS.Background = QCRTGraphTS.Background.newBackground(pTransform1,
    QCRTGraphTS.ChartConstants.PLOT_BACKGROUND, QCRTGraphTS.ChartColor.GRAY);
    this.chartVu.addChartObject(background);

    let attrib1: QCRTGraphTS.ChartAttribute =
    QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.GREEN, 1,
    QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.GREEN);

```



```

let barwidth: number = 1.0, barbase = 0.0;
let barjust: number = QCRTGraphTS.ChartConstants.JUSTIFY_MIN;
let barorient: number = QCRTGraphTS.ChartConstants.VERT_DIR;

let baraxis: QCRTGraphTS.LinearAxis = QCRTGraphTS.LinearAxis.newLinearAxis(pTransform1,
QCRTGraphTS.ChartConstants.Y_AXIS);
baraxis.calcAutoAxis();
this.chartVu.addChartObject(baraxis);

let barAxisLab: QCRTGraphTS.NumericAxisLabels =
QCRTGraphTS.NumericAxisLabels.newNumericAxisLabels(baraxis);
this.chartVu.addChartObject(barAxisLab);

this.InitializeProcessVar();

let barplot: QCRTGraphTS.RTBarIndicator =
QCRTGraphTS.RTBarIndicator.newRTBarIndicatorCoordsProcessVarAttrib(pTransform1,
    this.EngineRPML, barwidth, barbase,
    attrib1, barjust, barorient);
barplot.SegmentSpacing = 400;
barplot.SegmentWidth = 250;
barplot.IndicatorBackground.setColor(QCRTGraphTS.ChartColor.GRAY); // = new
ChartAttribute(ChartColor.Black, 1, DashStyle.Solid, ChartColor.Black);
barplot.SegmentValueRoundMode = QCRTGraphTS.ChartConstants.RT_CEILING_VALUE;
barplot.SegmentCornerRadius = 0;
barplot.IndicatorSubType = QCRTGraphTS.ChartConstants.RT_BAR_SEGMENTED_SUBTYPE;

let baralarms: QCRTGraphTS.RTAlarmIndicator =
QCRTGraphTS.RTAlarmIndicator.newRTAlarmIndicatorAxisRTPlot(baraxis, barplot);
this.chartVu.addChartObject(baralarms);

let panelmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.BLACK);
let paneltagmeterattrib: QCRTGraphTS.ChartAttribute =
QCRTGraphTS.ChartAttribute.newChartAttribute4(QCRTGraphTS.ChartColor.STEELBLUE, 3,
QCRTGraphTS.ChartConstants.LS_SOLID, QCRTGraphTS.ChartColor.WHITE);

let panelmeter: QCRTGraphTS.RTNumericPanelMeter =
QCRTGraphTS.RTNumericPanelMeter.newRTNumericPanelMeterCoordsAttrib(pTransform1,
panelmeterattrib);
panelmeter.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MIN;
panelmeter.NumericTemplate.TextBgMode = true;
panelmeter.NumericTemplate.LineColor = QCRTGraphTS.ChartColor.SPRINGGREEN;
panelmeter.NumericTemplate.TextFont = QCRTGraphTS.ChartFont.newChartFont("Arial",
QCRTGraphTS.ChartFont.REGULAR, 18);
panelmeter.NumericTemplate.DecimalPos = 0;
panelmeter.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
barplot.addPanelMeter(panelmeter);

let panelmeter2: QCRTGraphTS.RTAlarmPanelMeter =
QCRTGraphTS.RTAlarmPanelMeter.newRTAlarmPanelMeterCoordsAttrib(pTransform1, panelmeterattrib);
panelmeter2.PanelMeterPosition = QCRTGraphTS.ChartConstants.BELOW_REFERENCED_TEXT;
panelmeter2.AlarmTemplate.TextBgMode = true;
panelmeter2.AlarmTemplate.LineColor = QCRTGraphTS.ChartColor.SPRINGGREEN;
panelmeter2.AlarmTemplate.TextFont = QCRTGraphTS.ChartFont.newChartFont("Arial",
QCRTGraphTS.ChartFont.BOLD, 18);
panelmeter2.setPositionReference(panelmeter);
panelmeter2.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_TEXT_BACKGROUND_COLOR_CHANGE_ON_ALARM;
barplot.addPanelMeter(panelmeter2);

let panelmeter3: QCRTGraphTS.RTStringPanelMeter =
QCRTGraphTS.RTStringPanelMeter.newRTStringPanelMeterCoordsAttribStringType(pTransform1,
paneltagmeterattrib, QCRTGraphTS.ChartConstants.RT_TAG_STRING);
panelmeter3.PanelMeterPosition = QCRTGraphTS.ChartConstants.OUTSIDE_PLOTAREA_MAX;
panelmeter3.StringTemplate.TextBgMode = true;

```

```

    panelmeter3.StringTemplate.LineColor = QCRTGraphTS.ChartColor.BLACK;
    panelmeter3.StringTemplate.TextFont = QCRTGraphTS.ChartFont.newChartFont("Arial",
QCRTGraphTS.ChartFont.REGULAR, 18);
    panelmeter3.PanelMeterNudge = QCRTGraphTS.ChartPoint2D.newChartPoint2D(0, -6);
    panelmeter3.AlarmIndicatorColorMode =
QCRTGraphTS.ChartConstants.RT_INDICATOR_COLOR_NO_ALARM_CHANGE;
    barplot.addPanelMeter(panelmeter3);

    this.chartVu.addChartObject(barplot);

    this.chartVu.updateDraw();

```

Last is a call to a simple timer used to update the real-time bar chart using a 1 second interval.

```
this.StartTimer(1000, 500);
```

The timer is implemented at the top of the HelloRTBarPlot.js file using the functions:

```

TimerTickProc() {
    if (this.chartVu) {
        this.EngineRPM1Value = 5000 * QCRTGraphTS.ChartSupport.getRandomDouble();
        this.EngineRPM1Value = Math.max(this.EngineRPM1Value, 100);
        this.EngineRPM1Value = Math.min(this.EngineRPM1Value, 5000);

        this.EngineRPM1.setCurrentValueNumber(this.EngineRPM1Value);
        this.chartVu.updateDraw();
    }

    this.timercount++;
    if (this.timercount > this.timercountlimit)
        this.StopTimer();
}

public async StartTimer(milliseconds: number, count: number) {
    if (this.timerId != 0)
        this.StopTimer();
    this.timercountlimit = count;
    this.timerId = setInterval(() =>
        this.TimerTickProc(),
        milliseconds);
}

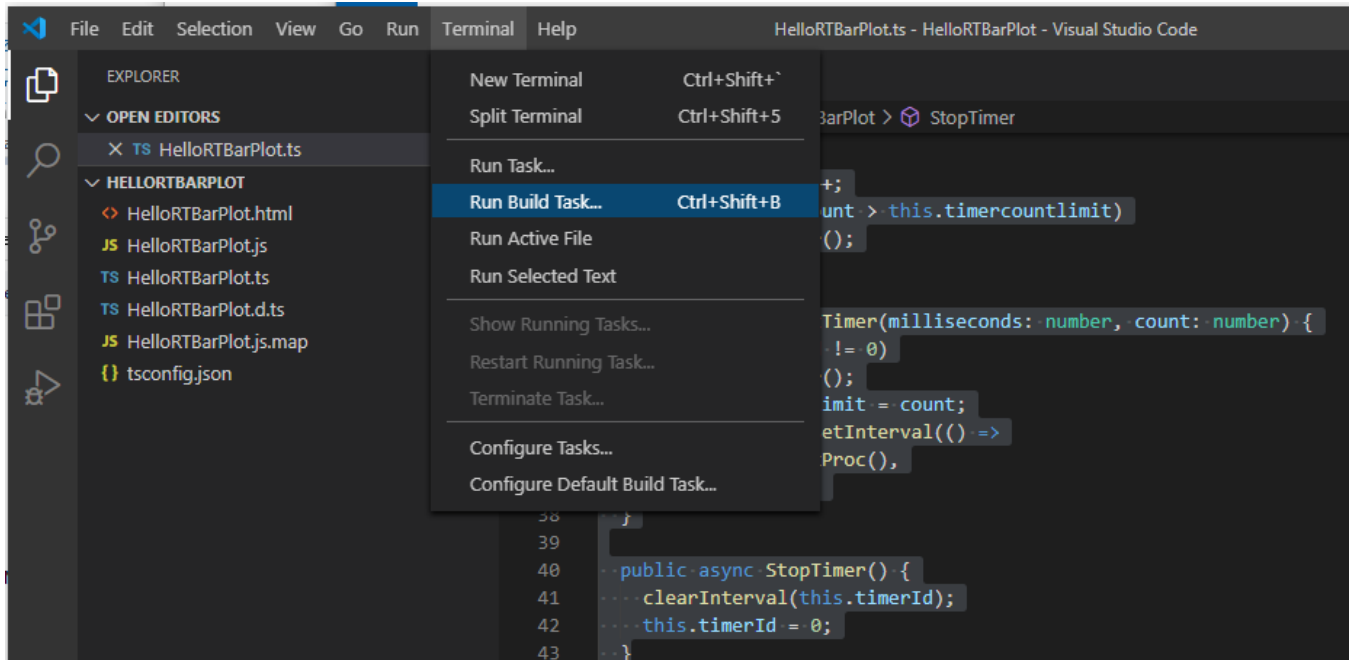
public async StopTimer() {
    clearInterval(this.timerId);
    this.timerId = 0;
}

```

The call to `StartTimer(1000, 500)` starts the timer with a 1000 ms interval (1 second) and stops the timer after a count of 500 updates has been reached. In the timer update function (`TimerTickProc`), the `EngineRPM1` `RTPProcessVar` object is updated with a new value using the call to the function `EngineRPM1.setCurrentValueNumber`. The real-time bar plots is updated to reflect this new value using the call to `chartVu.updateDraw()`. No matter how many bar plots (or any other real-time objects such as meters, scrolling graphics, annunciators, or panelmeter) you have in the chart, you first update all of the associated `RTPProcessVar` objects first, then call the `chartVu.updateDraw` function just once. All of the real-time graph objects will be updated simultaneously with a single call.

Unlike the JavaScript version of the same program, once the `HelloRTBarPlot.ts` TypeScript file is setup correctly you will need to compile it in order to generate the `HelloRTBarPlot.js` file. **In order to compile the project you need a `tsconfig.json` file in the project folder.** Just copy the one from the

TypescriptExamplesQCRTGraph/Bargraphs example and place it in the TypescriptExamplesQCRTGraph/HelloRTBarPlot folder. See Chapter 1 for more information about the options found in the tsconfig.json file.



Once that is in place, compile the HelloRTBarPlot.ts TypeScript file by selecting Terminal | Run Build Task.

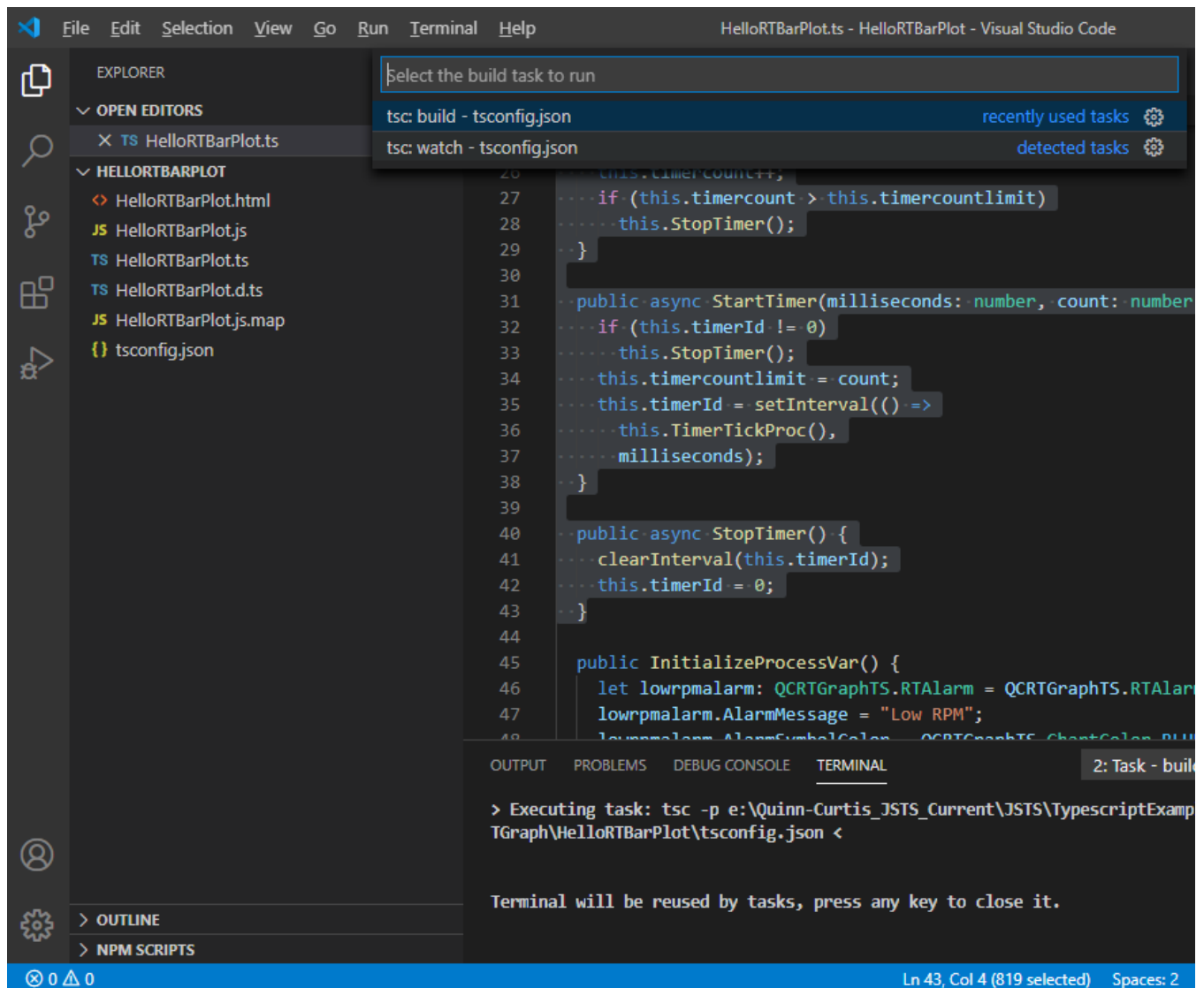
Then you need to select tsc: build – tsconfig.json from the drop down list (its hard to see).

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 HelloRTBarPlot.js and located in the HelloRTBarPlot folder, next to the other files.

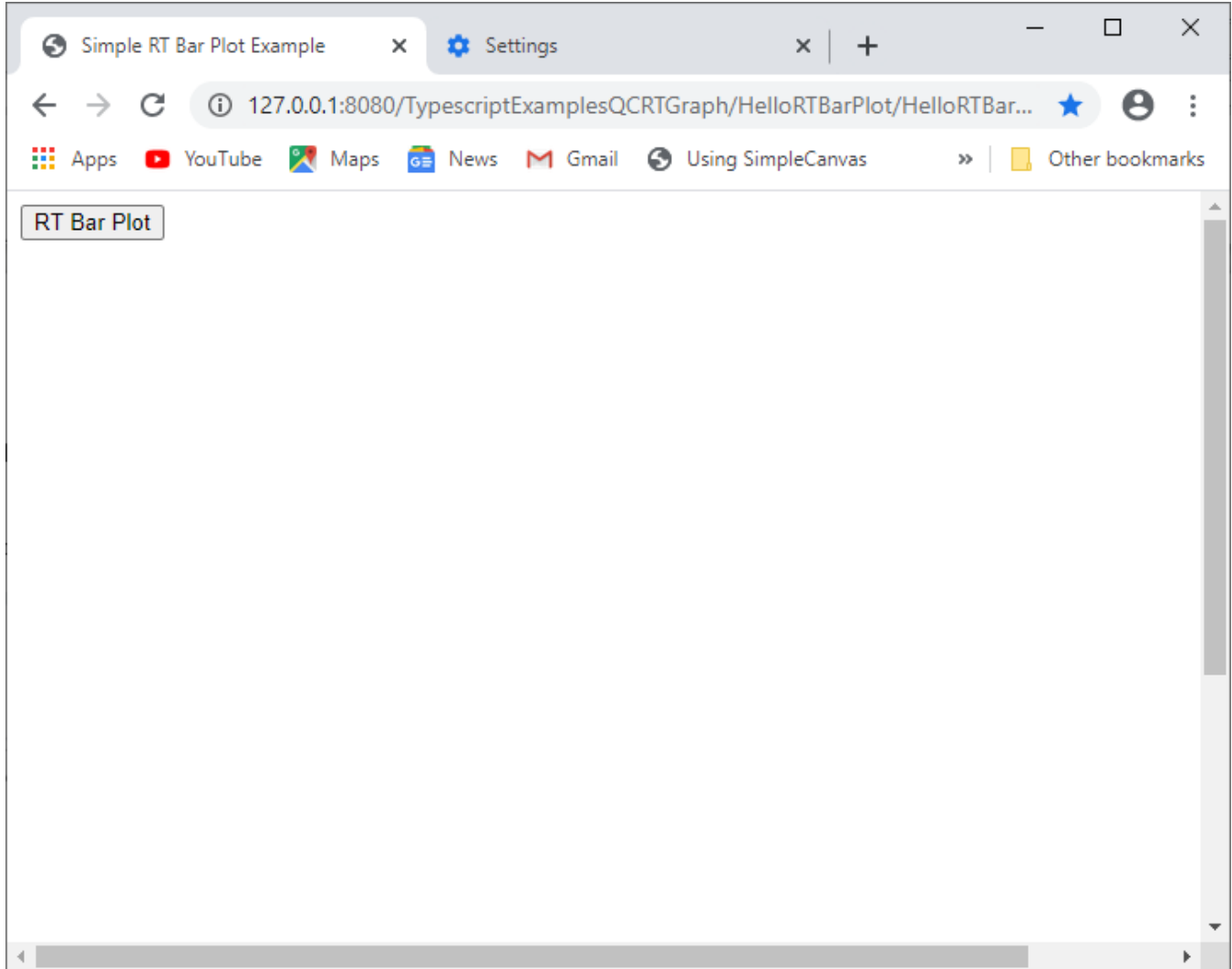
Other files also created during compilation are the HelloRTBarPlot.d.ts file (contains TypeScript type declarations for the HelloRTBarPlot.js file), and the HelloRTBarPlot.js.map (contains a mapping of the HelloRTBarPlot.js file to the HelloRTBarPlot.ts file) which is used in debugging the HelloRTBarPlot.ts source when you are actually executing the associated HelloRTBarPlot.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:

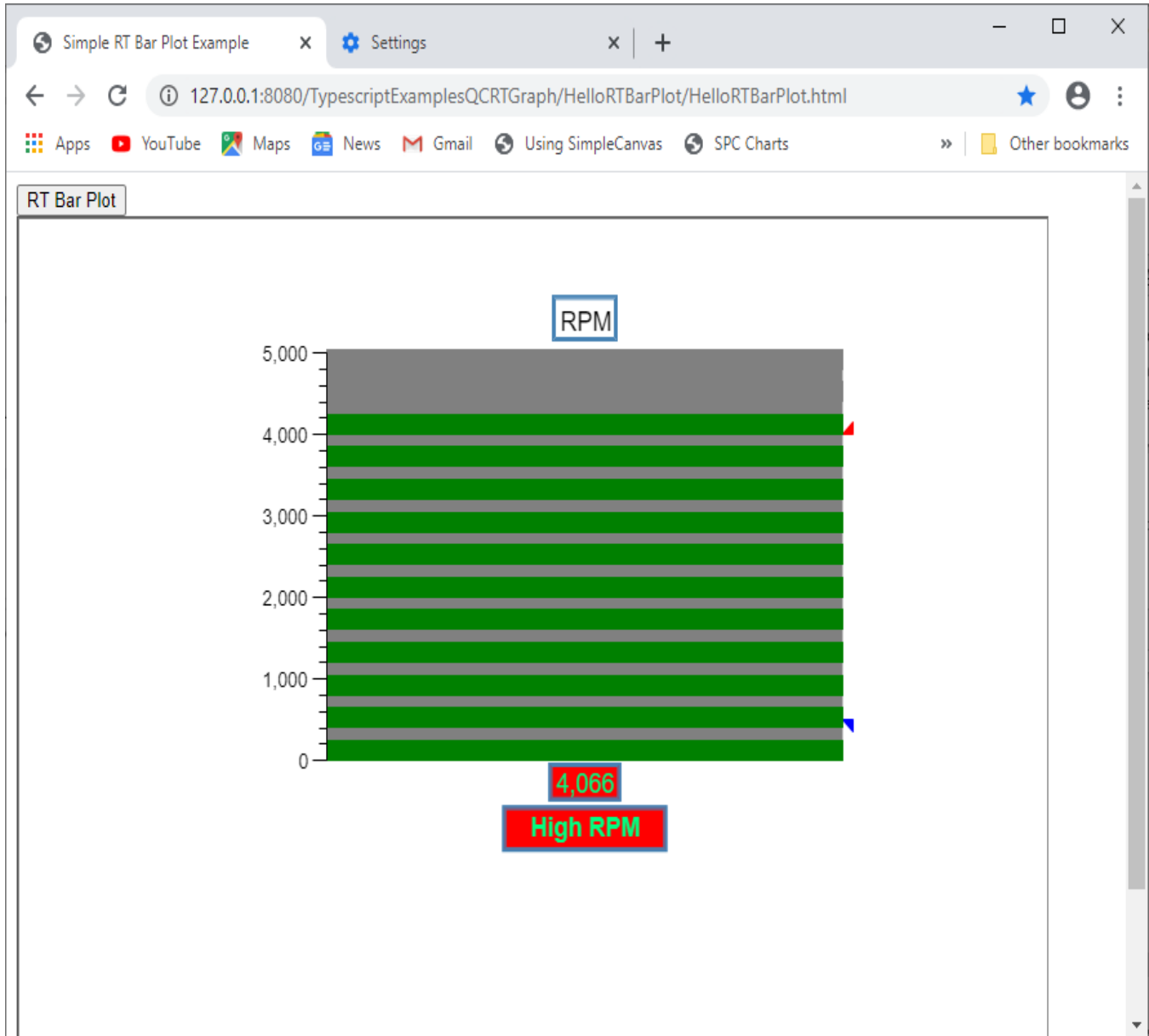
<http://127.0.0.1:8080/TypescriptExamplesQCRTGraph/HelloRTBarPlot/HelloRTBarPlot.html>



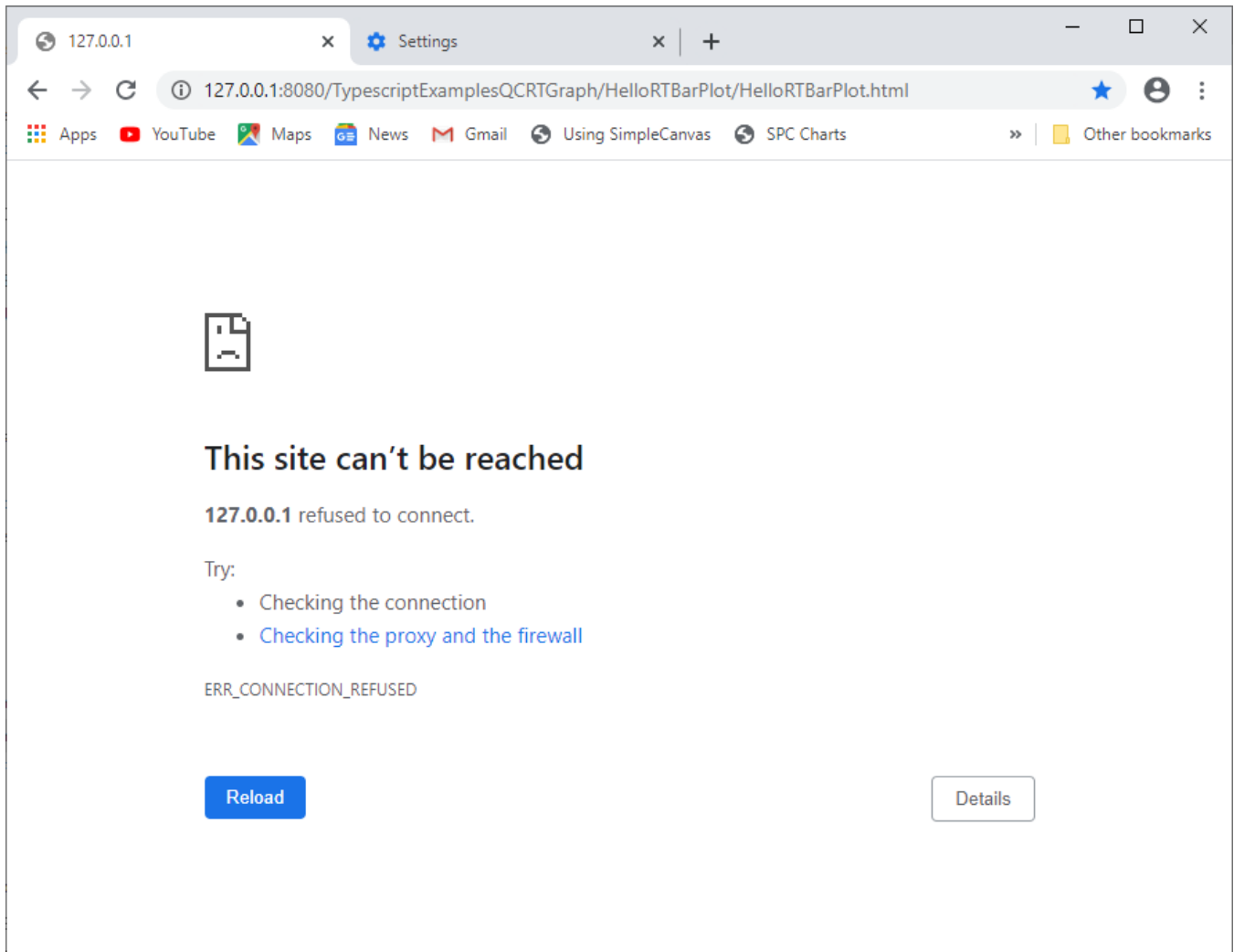
The result will be the following display:



showing just the **RT Bar Plot**. Click the button and the chart will appear.



If you get a screen that looks like:



you probably did not start the http-server local server

Start the npm “http-server” server using the batch file: startlocalserver.bat

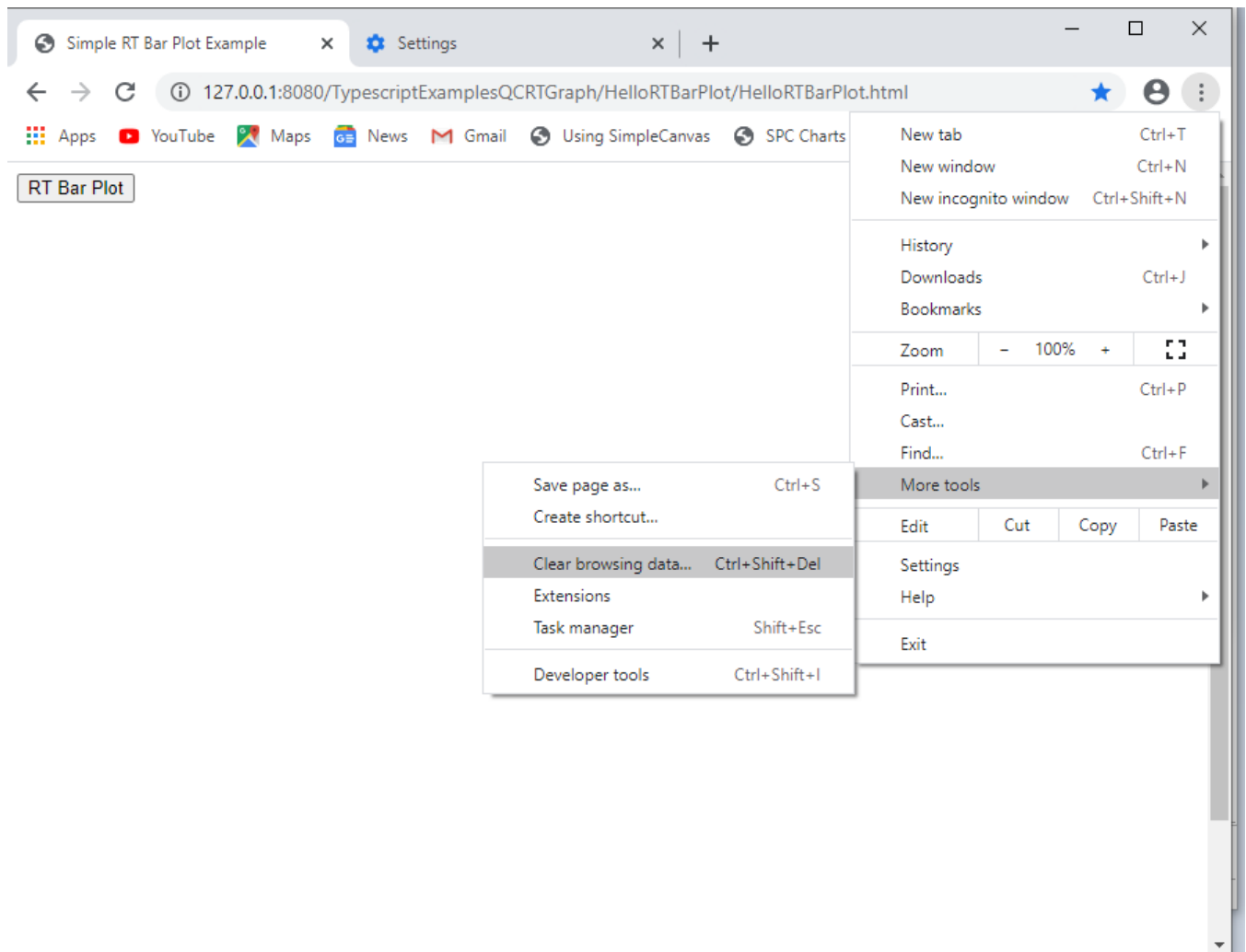
The main folder containing both the JavaScript and TypeScript examples is Quinn-Curtis/JSTS. In that folder is a batch file (**startlocalserver.bat**) which will start the npm “http-server” local server, setting the directory it is located in as the root directory of the test server. All the batch file 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:



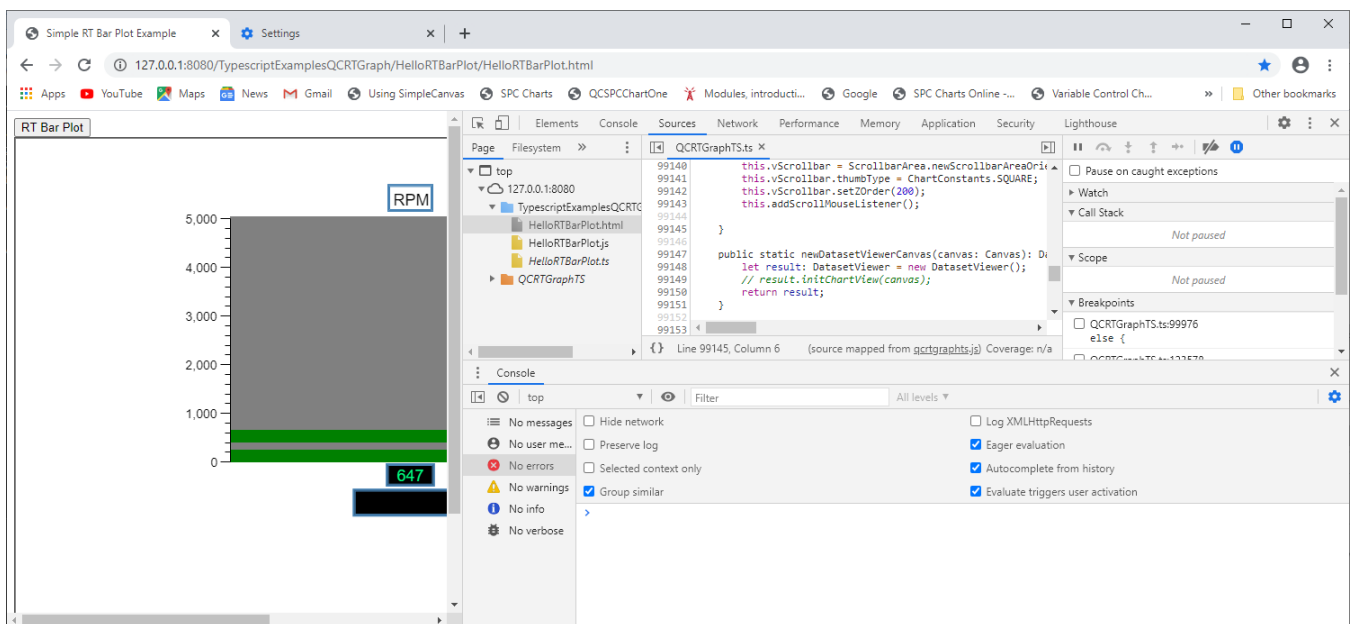
Or, the preferred method, is that you can turn off caching from within the browser Development Mode. Here is a good discussion of that option:

<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

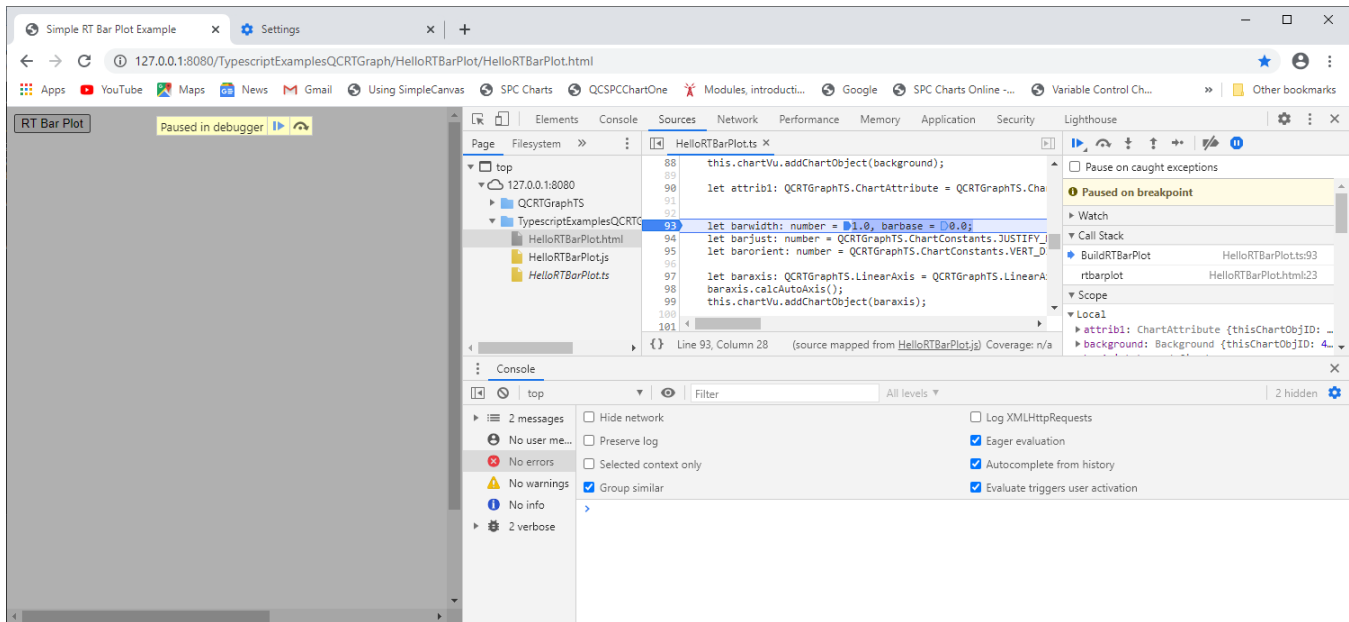
If for some reason the chart does not appear you will have start debugging.

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 HelloRTBarPlot.html file you should see the RT Bar Plot. Press F12 and the browsers integrated debugger will appear on the right. The debugger portion of the window will look something like:



You should be able to select either the HelloRTBarPlot.ts file, HelloRTBarPlot.js file, or the HelloRTBarPlot.html file to view in the main window. It is the presence of the HelloRTBarPlot.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 HelloRTBarPlot.ts file, and then click the RT Bar Plot in the left hand side of the browser, program execution will stop in the code window when the breakpoint is hit.



Using QCRTGraph 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 QCRTGraphTS is spelled in lower case: qcrtgraphs.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/TypescriptExamplesQCRTGraph/HelloRTBarPlot for the HelloRTBarPlot example.

Using QCRTGraph 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 QCRTGraph 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 *qcrgraphts.js* library (found in each project root, under the *QCRTGraphTS* 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 *qcrgraphts.js* file you have in your Quinn-Curtis\JSTS\QCRTGraphTS 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 (QCRTGraphTS and HelloRTBarPlot) which hold the *qcrgraphts.js* library, and the HelloRTBarPlot 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 QCRTGraphTS and HelloRTBarPlot folders are exactly the same. The HelloRTBarPlot.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 HelloRTBarPlot.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 QCRTGraphTS and HelloRTBarPlot 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 QCRTGraphTS library underneath the example program folder, so that the qcrtgraphts.js file is accessible to the JavaScript which displays the chart. So you will find the QCRTGraphTS and HelloRTBarPlot folders duplicated under all of the Asp.Net example program. In order to point the example program HelloRTBarPlot.js to the qcrtgraphts.js library, it uses the import statement:

```
import * as QCRTGraphTS from '../QCRTGraphTS/qcrtgraphts.js';
```

Rather than place the QCRTGraphTS 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 QCRTGraphTS from '../Scripts/QCRTGraphTS/qcrtgraphts.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 QCRTGraphTS and HelloRTBarPlot folders underneath the *wwwroot* folder. We placed the HelloRTBarPlot folder directly under the *wwroot* folder, and the QCRTGraphTS 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 HelloRTBarPlot.js to the qcrtgraphts.js library, it uses the import statement:

```
import * as QCRTGraphTS from '../lib/QCRTGraphTS/qcrtgraphts.js';
```

at the top of the HelloRTBarPlot.js. This is the relative path location, from the HelloRTBarPlot folder, to the qcrtgraphts.js file, in the QCRTGraphTS 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 HelloRTBarPlot.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 { BuildRTBarPlot } from './HelloRTBarPlot/HelloRTBarPlot.js';
    BuildRTBarPlot("chartCanvas1");
  </script>
```

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.

Index

Alarm Panel Meters.....	
Alarm Panel Meters. .	119, 121, 139, 209, 214, 320, 323
Alarms.....	
Alarms.	8, 10, 11, 13, 17, 19, 20, 35, 36, 37, 38, 45, 49, 50, 59, 97, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 118, 119, 120, 121, 137, 138, 139, 140, 150, 152, 209, 210, 214, 215, 305, 306, 307, 308, 319, 320, 322, 323
Angular.....	
Angular.....	23, 24
Angular framework.....	23
Angular Framework.....	
Angular.....	23
Annunciators.....	
Annunciators.....	35, 37, 40, 41, 109, 207, 208, 209, 210
AntennaAnnotation.....	83, 84, 95
AntennaAxes.....	67, 72, 74, 95
AntennaAxesLabels.....	72, 74, 95
AntennaCoordinates.....	64, 65, 95
AntennaGrid.....	89, 90, 95
AntennaLineMarkerPlot.....	83, 84, 95
AntennaLinePlot.....	83, 84, 95
AntennaPlot.....	74, 83, 84, 95
AntennaScatterPlot.....	83, 84, 95
Arrows.....	95
Asp.Net.....	
Asp.Net.....	vi, 26, 333, 334, 335
asynchronous.....	
async.....	
asynchronous.....	2, 15, 22, 304, 318, 336
Auto-scaling classes.....	65, 66, 95
AutoScale.....	
AutoScale7, 12, 19, 65, 66, 95, 218, 219, 220, 222, 223, 225, 226, 227, 228, 229, 230, 235, 236, 237, 238, 275, 276, 278	
Axis 33, 50, 67, 68, 72, 73, 90, 95, 158, 162, 163, 164, 167, 171	
axis.....	32, 273
Axis....iv, v, 7, 8, 12, 13, 18, 19, 28, 31, 32, 33, 36, 41, 43, 44, 49, 50, 51, 52, 53, 61, 66, 67, 68, 69, 70, 71, 72, 73, 74, 89, 90, 91, 92, 93, 94, 95, 115, 118, 119, 134, 137, 138, 139, 142, 143, 144, 145, 147, 150, 151, 152, 154, 155, 158, 159, 160, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 177, 197, 200, 201, 202, 203, 204, 206, 219, 220, 221, 222, 223, 227, 228, 231, 232, 236, 239, 240, 242, 256, 257, 258, 272, 273, 276, 279, 305, 308, 320, 323, 334	
Axis label classes.....	167, 168
AxisLabels.....	12
AxisLabels.....	72, 73, 95, 163, 167, 168
AxisLabels....7, 12, 19, 36, 50, 52, 72, 73, 95, 138, 139, 142, 143, 144, 145, 151, 152, 154, 155, 158, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 197, 201, 202, 204, 206, 305, 308, 320, 323	
AxisTitle.....	90, 95
AxisTitle.....	90, 95
Background.....	
background.....	32
Background...1, iv, vi, 7, 12, 18, 32, 37, 38, 41, 46, 57, 61, 67, 96, 108, 111, 118, 119, 121, 136, 138, 139, 140, 141, 142, 143, 144, 145, 148, 151, 152, 154, 155, 168, 176, 180, 181, 182, 183, 184, 185, 188, 189, 190, 191, 193, 194, 195, 196, 204, 205, 207, 209, 210, 214, 215, 280, 281, 282, 291, 292, 293, 305, 306, 307, 308, 320, 321, 322, 323	
Backgrounds.....	1, 67, 96, 280
background.....	12, 32, 307, 322
Background.....	12
BarDatapointValue.....	
BarDatapointValue.....	95
Bargraphs.....	
Bargraphs.....	35, 37, 40, 42, 47, 97, 109, 118, 119, 120, 121, 123, 133, 134, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 151, 153, 154, 305, 308, 320, 323
BoxWhiskerPlot.....	75, 76, 95
BubblePlot.....	75, 76, 95, 234
BubblePlotLegend.....	88, 89, 95
BubblePlotLegendItem.....	88, 89, 95
Buffered images.....	
BufferedImage.....	296, 297, 298
BufferedImage.....	93, 95, 296, 297
BufferedImage.....	93, 95, 296, 297, 298
Button Controls.....	
Button Controls33, 53, 54, 242, 244, 245, 246, 275, 278	
cache.....	
cache.....	313, 330
Cache.....	313, 330
Calendar utilities.....	
ChartCalendar.....	279
CandlestickPlot.....	75, 77, 95
Canvas.....	
Canvas1, 6, 7, 10, 14, 15, 18, 21, 22, 23, 24, 25, 30, 31, 33, 53, 241, 242, 243, 296, 297, 300, 301, 303, 305, 306, 307, 317, 319, 321, 322, 335	
Cartesian coordinates.....	
CartesianCoordinates.....	10
CartesianCoordinates.....	64, 65, 95
CartesianCoordinates 10, 64, 65, 95, 117, 126, 128, 129, 131, 132, 138, 139, 142, 143, 144, 145, 151, 152, 154, 155, 209, 210, 212, 214, 215,	

246, 247, 248, 251, 252, 253, 257, 260, 262, 282, 283, 305, 306, 307, 319, 321, 322	ChartSymbol.....91, 96
case sensitive.....4, 5, 332	ChartText.....168
case sensitive.....4, 5, 332	ChartText.....57, 59, 90, 95, 168
CellPlot.....75, 77, 95, 234	ChartTitle.....90, 95
Chart object attributes.....32, 61, 65	ChartTitle.....8, 19, 20, 90, 95
attribute.....32, 244	ChartView 31, 34, 37, 55, 62, 66, 92, 93, 95, 103, 108, 118, 135, 137, 148, 150, 164, 168, 177, 218, 219, 222, 225, 226, 273, 277, 285, 287, 290, 291, 292, 297
Attribute.....32	ChartView. 7, 10, 11, 12, 13, 17, 18, 19, 25, 31, 34, 37, 55, 62, 66, 92, 93, 95, 103, 108, 118, 135, 137, 148, 150, 164, 168, 177, 218, 219, 222, 225, 226, 227, 228, 244, 273, 276, 277, 285, 287, 290, 291, 292, 297, 301, 305, 306, 307, 308, 318, 319, 321, 322
ChartAttribute.....65, 66, 95, 135, 148, 164, 168, 176, 233, 255, 259, 281, 287	ContourDataset.....62, 95
ChartAttribute...7, 8, 11, 13, 19, 20, 65, 66, 95, 98, 100, 101, 103, 104, 106, 107, 115, 116, 117, 118, 119, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 134, 135, 138, 139, 141, 142, 143, 144, 145, 147, 148, 151, 152, 153, 154, 155, 161, 162, 164, 165, 166, 168, 169, 170, 172, 173, 176, 178, 179, 180, 181, 182, 183, 184, 185, 186, 188, 189, 190, 191, 193, 194, 195, 196, 198, 200, 201, 202, 203, 204, 205, 206, 208, 209, 210, 211, 212, 213, 214, 215, 227, 228, 233, 234, 236, 237, 244, 246, 247, 248, 250, 252, 253, 255, 257, 258, 259, 260, 262, 280, 281, 285, 286, 287, 288, 305, 306, 307, 308, 319, 320, 322, 323	CSV.....93, 94, 95
ChartCalendar...62, 63, 66, 70, 90, 93, 94, 95, 98, 279	CSV.....93, 94, 95
ChartCalendar 7, 18, 62, 63, 66, 70, 93, 94, 95, 225, 226, 236, 237, 279	DataCursor.....91, 92, 95
ChartColor.....7, 8, 10, 11, 12, 13, 18, 19, 20, 100, 101, 103, 104, 106, 107, 115, 117, 118, 119, 121, 123, 124, 126, 127, 128, 129, 131, 132, 138, 139, 140, 141, 142, 143, 144, 145, 151, 152, 154, 155, 161, 162, 165, 166, 169, 170, 172, 173, 174, 179, 180, 181, 182, 183, 184, 185, 188, 189, 190, 191, 193, 194, 195, 196, 200, 201, 202, 203, 204, 205, 206, 209, 210, 212, 213, 214, 215, 227, 228, 236, 237, 246, 247, 248, 252, 253, 254, 257, 260, 261, 262, 285, 286, 288, 305, 306, 307, 308, 319, 320, 321, 322, 323, 324	DataCursor.....91, 92, 95
ChartColor.....7, 8, 10, 11, 12, 13, 18, 19, 20, 100, 101, 103, 104, 106, 107, 115, 117, 118, 119, 121, 123, 124, 126, 127, 128, 129, 131, 132, 138, 139, 140, 141, 142, 143, 144, 145, 151, 152, 154, 155, 161, 162, 165, 166, 169, 170, 172, 173, 174, 179, 180, 181, 182, 183, 184, 185, 188, 189, 190, 191, 193, 194, 195, 196, 200, 201, 202, 203, 204, 205, 206, 209, 210, 212, 213, 214, 215, 227, 228, 236, 237, 246, 247, 248, 252, 253, 254, 257, 260, 261, 262, 285, 286, 288, 305, 306, 307, 308, 319, 320, 321, 322, 323, 324	Dataset.....31, 62, 74, 95, 97, 98
ChartFont.....6, 17, 18, 19, 109, 110, 117, 131, 161, 162, 165, 169, 173, 180, 181, 183, 184, 188, 190, 193, 195, 200, 204, 246, 251, 260, 282, 306, 308, 319, 320, 321, 323, 324	ChartDataset.....31, 62, 74, 95, 97, 98
ChartImage.....91, 96	Dataset classes.....31, 62, 74, 95, 97, 98
ChartImage.....91, 96	DatasetViewer.....289, 290, 291, 292, 293
ChartLabel.....90, 91, 95, 256	DataToolTip.....91, 92, 95
ChartLabel.....90, 91, 95, 256	DataToolTip.....91, 92, 95
ChartObj.....35, 36, 57, 95, 97, 101, 104, 272, 297	debugging.....3, 16, 314, 325, 331
ChartObj.....297, 308, 322	Debugging.....314, 331
.ChartConstants.....274	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.
ChartPlot.....135, 136, 137, 148, 149, 150	329
ChartPlot...31, 35, 37, 39, 45, 54, 74, 95, 110, 115, 119, 122, 125, 127, 129, 134, 135, 136, 137, 147, 148, 149, 150, 176, 177, 186, 191, 208, 210, 218, 221, 224, 234	313, 331
ChartScale.....63, 64, 95	Developer License.....ii
ChartScale.....63, 64, 95	Developer License.....ii
ChartShape.....91, 96	Dimension.....93, 94, 95, 279
ChartShape.....91, 96	dimension.....32, 274
ChartSymbol.....91, 96	Dimension.....279
	Elapsed Time scale.....35, 37, 39, 108, 127, 128, 129
	Elapsed Time scale.....35, 37, 39, 108, 127, 128, 129
	ElapsedTimeAutoScale.....65, 66, 95
	ElapsedTimeAxis.....67, 71, 73
	ElapsedTimeAxisLabels.....72, 73, 95
	ElapsedTimeGroupDataset.....62, 63, 66, 95
	ElapsedTimeLabel.....39, 90, 91, 95, 127
	ElapsedTimeScale.....63
	ElapsedTimeSimpleDataset.....62, 66, 95
	ErrorBarPlot.....75, 77, 95
	FindObj.....91, 92, 95
	FindObj.....91, 92, 95
	FloatingBarPlot.....75, 78, 95, 234
	FloatingBarPlot.....75, 78, 95, 234
	Form Control Panel Meters.....

Form Control Panel Meters. .35, 37, 39, 52, 53, 54, 108, 129, 130, 131, 132, 241, 252, 253, 254, 255, 256, 257	LogScale.....63, 95
Graph object class. .135, 136, 137, 148, 149, 150, 164, 168, 176, 177, 255, 256, 259, 281, 284, 287	LogScale.....63, 95
GraphObj.....37, 57, 65, 66, 83, 95, 108, 135, 136, 137, 148, 149, 150, 164, 168, 176, 177, 255, 256, 259, 273, 280, 281, 284, 286, 287	MagniView.....92, 93, 95
GraphObj 37, 57, 65, 66, 67, 83, 95, 108, 135, 136, 137, 148, 149, 150, 164, 168, 176, 177, 255, 256, 259, 273, 280, 281, 284, 286, 287	Markers.....91, 92, 96, 224, 234
Grid.....	marker.....30
grid.....30, 32	Marker29, 30, 74, 83, 84, 86, 87, 88, 91, 92, 95, 96, 224, 228, 230, 231, 234, 235, 239
Grid. v, 7, 12, 19, 30, 32, 33, 35, 45, 46, 48, 53, 54, 61, 89, 90, 95, 97, 210, 212, 231, 239, 241, 246, 247, 248, 249, 254, 258, 259, 260, 261, 262, 290, 291, 292, 293, 294	Meters, Clocks and Dials.....178, 187, 192
Grids.....89, 95, 258	Meters, Clocks and Dials. .v, 35, 36, 37, 40, 42, 43, 44, 50, 97, 109, 158, 159, 160, 161, 162, 163, 164, 165, 166, 169, 170, 172, 173, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206
grid.....12, 30, 32	MouseListener.....
Grid.....12	MouseListener. .91, 92, 93, 95, 130, 131, 132, 243, 246, 247, 248, 249, 252, 253, 254, 255, 257, 258, 261, 262, 273, 275, 276, 277, 278
GroupBarPlot.....75, 79, 95, 233	MouseListeners.....91, 92, 93, 95
GroupBarPlot.....75, 79, 95, 233	MouseListener.....243, 273
GroupDataset.....62, 63, 66, 95	MoveCoordinates.....92, 93, 95
GroupDataset.....35, 62, 63, 66, 95	MoveData.....
GroupPlot29, 31, 33, 46, 47, 74, 75, 76, 77, 78, 79, 80, 81, 82, 95, 97, 233, 234	MoveData.....91, 92, 95
GroupPlot29, 31, 33, 46, 47, 74, 75, 76, 77, 78, 79, 80, 81, 82, 95, 97, 233, 234, 235, 236, 237, 291	MoveObj.....
GroupVersaPlot.....75, 79, 95	MoveObj.....91, 92, 95
HistogramPlot.....75, 80, 95, 234	Moving chart data.....91, 92, 95
HistogramPlot.....75, 80, 95, 234	Moving graph objects.....91, 92, 95
HTML5.....2, 3	MultiLinePlot.....47, 75, 81, 95, 233
HTML5 1, 2, 3, iv, 14, 21, 30, 31, 33, 53, 241, 296, 297, 300, 335	MultiLinePlot.....47, 75, 81, 95, 233, 235, 236
http-server.....	MVC.....
http-server.....5, 9, 20, 21, 300, 312, 313, 329, 332, 333	MVC.....334, 335
9	NearestPointData.....93, 94, 95
Image objects.....91, 96	npm.....
Image Rendering.....	npm. .4, 5, 9, 20, 21, 300, 301, 312, 313, 329, 332, 333
BufferedImage.....93, 95, 296, 297, 298	sudo npm.....5, 333
Legend.....	NPM Server.....
legend.....30	npm server.....300, 312, 329
Legend.....30, 88, 89, 92, 95, 307, 322	Numeric axis labels.....167
Legend classes.....88, 89, 95	NumericAxisLabels.....12
legend.....30, 307, 322	Numeric data point labels.....95
LegendItem.....88, 95	Numeric Panel Meters.....
LegendItem.....88, 89, 95	Numeric Panel Meters35, 37, 38, 45, 108, 115, 116, 117, 118, 119, 121, 124, 131, 132, 137, 138, 139, 150, 181, 182, 185, 186, 188, 189, 190, 191, 194, 195, 196, 205, 206, 209, 210, 214, 215, 251, 252, 253, 254, 306, 308, 320, 323
Line plots.....	NumericAxisLabels.....36, 50, 72, 73, 95, 167
Bargraphs.....325	NumericLabel.....38, 90, 95, 112, 116, 136
Linear axis.....164	NumericLabel.....38, 90, 95, 112, 116, 136
LinearAxis.....12	OHLCPlot.....
LinearAutoScale.....65, 66, 95	OHLCPlot.....75, 81, 95, 234, 236, 237
LinearAxis.....36, 50, 67, 68, 73, 89, 95, 163, 164	Open-High-Low-Close plots.....75, 81, 95, 234
LinearScale.....63, 95	Panel Meters.....111, 256
LinearScale.....63, 95	Panel Meters.....
LineGapPlot.....75, 80, 95	Panel Meters. 33, 35, 37, 40, 42, 45, 52, 57, 59, 97, 108, 110, 111, 112, 115, 116, 119, 122, 125, 127, 129, 147, 158, 175, 208, 255, 256, 280
LineGapPlot.....75, 80, 95	Physical coordinates.....
Linux.....	PhysicalCoordinates.....273
Linux.....4, 300, 332	
LogAutoScale.....65, 66, 95	
LogAxis.....67, 69, 73, 89, 95	

- PhysicalCoordinates.63, 64, 66, 67, 95, 135, 148, 176, 255, 259, 273, 277
 PhysicalCoordinates....63, 64, 66, 67, 95, 116, 119, 122, 125, 127, 130, 134, 135, 141, 147, 148, 153, 176, 208, 211, 218, 222, 224, 234, 255, 258, 259, 273, 276, 277, 280, 284, 286, 290
- PID Control.....
 PID Control.....35, 57, 59, 255, 266, 268, 269, 270
- PieChart.....74, 85, 95
- Plot object classes. .31, 35, 37, 39, 45, 54, 74, 95, 110, 115, 119, 122, 125, 127, 129, 134, 135, 136, 137, 147, 148, 149, 150, 176, 177, 186, 191, 208, 210, 218, 221, 224, 234
- Point3D.....95
- PolarAxes.....67, 71, 74, 89, 95
- PolarAxesLabels.....72, 74, 95
- PolarCoordinates.....35, 50, 64, 65, 95, 158
- PolarGrid.....89, 90, 95
- PolarLinePlot.....82, 83, 95
- PolarPlot.....74, 82, 83, 95
- PolarScatterPlot.....82, 83, 95
- Polysurface class.....95
- Printing.....93, 95
- Rectangle2D.....94, 95, 281
- Redistributable License.....
 Redistributable License.....ii
- RingChart.....86
- RT XE.....35, 36, 97, 99, 100, 101, 102, 103, 105, 106
- RT3DFrame.....35, 57, 280, 281, 282
 RT3DFrame.....169, 280, 281, 282, 283
- RTAlarm.....
 RTAlarm....8, 10, 11, 13, 17, 19, 20, 35, 36, 37, 38, 45, 49, 50, 59, 97, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 118, 119, 120, 121, 137, 138, 139, 140, 150, 209, 210, 214, 215, 305, 306, 307, 308, 319, 320, 322, 323
- RTAlarmEventArgs. .35, 36, 37, 97, 103, 104, 105, 106
 RTAlarmEventArgs. .35, 36, 37, 97, 103, 104, 105, 106, 107
- RTAlarmIndicator.....35, 49, 50, 59, 137, 150
 RTAlarmIndicator.....8, 13, 19, 35, 49, 50, 59, 118, 119, 137, 138, 139, 150, 305, 308, 320, 323
- RTAlarmPanelMeter35, 37, 38, 45, 108, 119, 120, 137, 150
- RTAlarmPanelMeter.....
 RTAlarmPanelMeter. 119, 121, 139, 209, 214, 320, 323
- RTAnnunciator.....35, 37, 40, 41, 109, 207, 208
 RTAnnunciator.35, 37, 40, 41, 109, 207, 208, 209, 210
- RTBarIndicator.35, 37, 40, 42, 97, 109, 118, 120, 133, 134, 137, 140, 143, 153
 RTBarIndicator. .35, 37, 40, 42, 97, 109, 118, 119, 120, 121, 133, 134, 137, 138, 139, 140, 141, 142, 143, 144, 145, 153, 305, 308, 320, 323
- RTComboProcessVar.....35, 197, 198, 199, 203
 RTComboProcessVar.....v, 35, 197, 198, 199, 201, 203, 205
- RTControlButton....33, 36, 52, 53, 241, 243, 250, 255, 260
 RTControlButton....33, 53, 54, 242, 244, 245, 246, 275, 278
- RTControlTrackBar.....36, 52, 54, 241, 249, 251, 254, 255, 256
 RTControlTrackBar....33, 36, 52, 53, 54, 130, 131, 132, 241, 242, 243, 249, 250, 251, 252, 253, 254, 255, 256, 257
- RTDatasetTruncateArgs.....35
- RTElapsedTimePanelMeter.....35, 37, 39, 127, 128
 RTElapsedTimePanelMeter....35, 37, 39, 108, 127, 128, 129
- RTFormControl.....36, 39, 48, 54, 129
- RTFormControlGrid. .35, 45, 48, 53, 54, 97, 241, 254, 258, 259, 260
- RTFormControlPanelMeter 35, 37, 39, 52, 53, 54, 108, 129, 130, 241, 255, 256
 RTFormControlPanelMeter. .35, 37, 39, 52, 53, 54, 108, 129, 130, 131, 132, 241, 252, 253, 254, 255, 256, 257
- RTGenShape.....35, 57, 58, 280, 286, 287
 RTGenShape...35, 57, 58, 201, 203, 280, 286, 287, 288
- RTGroupDatasetTruncateArgs.....35
- RTGroupMultiValuePlot 29, 31, 34, 35, 45, 47, 48, 55, 97, 217, 218, 221, 228, 233, 234, 235
 RTGroupMultiValuePlot. 29, 31, 34, 35, 45, 47, 48, 55, 97, 217, 218, 221, 228, 233, 234, 235, 236, 237, 239
- RTMeterArcIndicator...35, 40, 43, 175, 177, 178, 179, 197
 RTMeterArcIndicator....35, 40, 43, 175, 177, 178, 179, 180, 181, 182, 183, 184, 185, 197
- RTMeterAxis....36, 49, 50, 51, 52, 158, 162, 163, 164, 167, 168, 171, 177, 197
- RTMeterAxisLabels.....36, 50, 52, 158, 164, 167, 168, 171, 197
- RTMeterCoordinates. .36, 50, 158, 159, 160, 163, 164, 177, 178, 186, 192, 197
 RTMeterCoordinates....36, 50, 158, 159, 160, 161, 162, 163, 164, 165, 166, 169, 170, 172, 173, 177, 178, 180, 181, 182, 183, 184, 185, 186, 188, 189, 190, 191, 192, 193, 194, 195, 197, 200, 202, 204, 205
- RTMeterIndicator.....35, 37, 40, 42, 97, 109, 163, 164, 175, 176, 177, 178, 186, 187, 191, 192, 197
 RTMeterIndicator 35, 37, 40, 42, 97, 109, 163, 164, 175, 176, 177, 178, 186, 187, 191, 192, 197
- RTMeterNeedleIndicator.....35, 40, 44, 175, 186, 187, 197
 RTMeterNeedleIndicator 35, 40, 44, 165, 166, 169, 170, 172, 173, 175, 186, 187, 188, 189, 190, 191, 197, 200, 202, 204, 205, 206
- RTMeterStringAxisLabels...36, 50, 52, 158, 171, 172, 197
- RTMeterSymbolIndicator....35, 40, 44, 175, 191, 192, 197
- RTMeterSymbolIndicator.....
 RTMeterSymbolIndicator 35, 40, 44, 175, 191, 192, 193, 194, 195, 197
- RTMultiAlarmIndicator.....35, 49, 50
 RTMultiAlarmIndicator.....35, 49, 50, 152
- RTMultiBarIndicator35, 37, 45, 47, 97, 109, 123, 146, 147, 150, 153

- RTMultiBarIndicator. .47, 123, 146, 147, 151, 153, 154
- RTMultiValueAnnunciator.....35, 45, 46, 97, 207, 210, 211
- RTMultiValueAnnunciator 35, 45, 46, 97, 207, 210, 211, 212, 213, 214, 215
- RTMultiValueIndicator....35, 45, 50, 53, 54, 147, 208, 210, 218, 221, 234, 258, 259
- RTNumericPanelMeter...35, 37, 38, 45, 108, 115, 116, 118, 137, 150, 251, 254
- RTNumericPanelMeter.....
- RTNumericPanelMeter.....35, 37, 38, 45, 108, 115, 116, 117, 118, 119, 121, 124, 131, 132, 137, 138, 139, 150, 181, 182, 185, 186, 188, 189, 190, 191, 194, 195, 196, 205, 206, 209, 210, 214, 215, 251, 252, 253, 254, 306, 308, 320, 323
- RTPanelMeter....33, 35, 37, 40, 42, 45, 52, 57, 59, 97, 108, 110, 111, 112, 115, 116, 119, 122, 125, 127, 129, 147, 158, 175, 208, 255, 256, 280
- RTPanelMeter.....
- RTPanelMeter 33, 35, 37, 40, 42, 45, 52, 57, 59, 97, 108, 110, 111, 112, 115, 116, 119, 122, 125, 127, 129, 147, 158, 175, 208, 255, 256, 280
- RTPIDControl.....35, 57, 59, 266, 268
- RTPIDControl...35, 57, 59, 255, 266, 268, 269, 270
- RTPlot 35, 37, 39, 45, 54, 110, 111, 115, 119, 122, 125, 127, 129, 134, 135, 136, 147, 148, 149, 176, 177, 186, 191, 208, 210, 218, 221, 224, 234, 256, 259
- RTPlot. 8, 13, 19, 22, 31, 35, 37, 39, 45, 54, 74, 95, 110, 111, 115, 118, 119, 122, 125, 127, 129, 134, 135, 136, 137, 147, 148, 149, 150, 176, 177, 186, 191, 208, 210, 218, 221, 224, 234, 237, 238, 256, 259, 305, 308, 317, 320, 323
- RTProcessVar 29, 31, 33, 35, 36, 37, 38, 39, 40, 42, 43, 44, 45, 46, 47, 51, 55, 97, 98, 99, 100, 102, 103, 104, 105, 106, 108, 111, 116, 117, 118, 119, 122, 123, 125, 127, 133, 136, 146, 147, 149, 163, 175, 177, 178, 186, 191, 197, 198, 199, 208, 210, 211, 212, 213, 215, 218, 224, 234, 256, 259, 269, 272, 289, 290, 291
- RTProcessVar. .8, 10, 11, 12, 13, 17, 20, 29, 31, 33, 35, 36, 37, 38, 39, 40, 42, 43, 44, 45, 46, 47, 51, 55, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 111, 116, 117, 118, 119, 122, 123, 125, 127, 130, 133, 134, 136, 141, 146, 147, 149, 153, 154, 163, 175, 177, 178, 186, 191, 197, 198, 199, 201, 203, 205, 208, 210, 211, 212, 213, 215, 218, 222, 224, 232, 234, 236, 240, 255, 256, 258, 259, 269, 272, 289, 290, 291, 293, 294, 304, 305, 307, 309, 318, 319, 322, 324
- RTRoundedRectangle2D.....35, 280, 284, 285
- RTRoundedRectangle2D.....
- RTRoundedRectangle2D.....35, 280, 284, 285, 286
- RTScrollFrame.....
- RTScrollFrame..6, 7, 12, 17, 19, 29, 34, 35, 45, 48, 54, 55, 56, 217, 218, 220, 221, 225, 226, 229, 230, 235, 236, 237, 238, 272
- RTSimpleSingleValuePlot....29, 31, 34, 35, 40, 45, 48, 55, 97, 217, 218, 221, 224, 225, 226, 228, 229, 230, 231, 235, 237, 238, 239
- RTSimpleSingleValuePlot 8, 13, 19, 29, 31, 34, 35, 40, 45, 48, 55, 97, 217, 218, 221, 224, 225, 226, 227, 228, 229, 230, 231, 235, 237, 238, 239
- RTSingleValueIndicator.....35, 37, 39, 40, 45, 50, 110, 111, 116, 119, 122, 125, 127, 129, 134, 135, 136, 148, 149, 176, 177, 186, 191, 224, 256
- RTSingleValueIndicator 35, 37, 39, 40, 45, 50, 110, 111, 116, 119, 122, 125, 127, 129, 134, 135, 136, 148, 149, 176, 177, 186, 191, 224, 256
- RTStringPanelMeter.....
- RTStringPanelMeter. 115, 122, 124, 126, 128, 131, 139, 188, 209, 212, 214, 252, 253, 321, 323
- RTTextFrame.....35, 57, 59
- RTTextFrame.....35, 57, 59
- RTTimePanelMeter.....
- RTTimePanelMeter.....125, 126
- RTVerticalScrollFrame.....
- RTVerticalScrollFrame..29, 34, 35, 45, 48, 54, 56, 217, 218, 221, 222, 223, 227, 228
- Scale classes.....63, 64, 95
- Scatter Plots.....
- scatter plots.....32
- Scatter Plots.....32, 61, 74, 82, 83, 86
- Scrolling graph.....29, 34, 35, 56, 217, 221, 223
- Scrolling graph 29, 34, 35, 45, 48, 54, 56, 217, 218, 221, 222, 223, 227, 228
- Scrolling Graphs....29, 34, 35, 45, 48, 54, 55, 56, 217, 218, 220, 221, 225, 226, 272
- Scrolling Graphs 6, 7, 12, 17, 19, 29, 34, 35, 45, 48, 54, 55, 56, 217, 218, 220, 221, 225, 226, 229, 230, 235, 236, 237, 238, 272
- Shapes.....91, 96
- SimpleBarPlot.....29, 86, 95
- SimpleBarPlot.....29, 86, 95
- SimpleDataset.....62, 66, 95
- SimpleDataset.....62, 66, 95, 98, 198, 229
- SimpleLineMarkerPlot.....29, 86, 87, 95
- SimpleLineMarkerPlot.....29, 86, 87, 95
- SimpleLinePlot.....29, 86, 87, 95
- SimpleLinePlot....7, 8, 13, 19, 29, 86, 87, 95, 225, 226, 227, 228, 229, 231, 237, 238, 239
- SimplePlot....31, 33, 40, 45, 74, 86, 87, 88, 91, 92, 95, 97, 224, 234
- SimplePlot 31, 33, 40, 45, 74, 86, 87, 88, 91, 92, 95, 97, 224, 234
- SimpleScatterPlot.....29, 86, 88, 95
- SimpleScatterPlot.....29, 86, 88, 95
- SimpleVersaPlot.....88, 95
- StackedBarPlot.....75, 79, 95, 233
- StackedBarPlot.....75, 78, 79, 95, 233
- StackedLinePlot.....75, 82, 95, 233
- StackedLinePlot.....75, 82, 95, 233
- StandardLegend.....88, 89, 95
- String axis labels.....172
- String Panel Meters 35, 37, 39, 45, 108, 122, 123, 137, 150
- String Panel Meters.....
- String Panel Meters...115, 122, 124, 126, 128, 131, 139, 188, 209, 212, 214, 252, 253, 321, 323
- StringAxisLabels.....36, 50, 72, 73, 95, 171, 172
- StringLabel.....38, 39, 90, 91, 95, 112, 119, 122
- StringLabel.....38, 39, 90, 91, 95, 112, 119, 122

Symbols.....	91, 96
Templates.....	
templates.....	23, 334
Text classes.....	57, 59, 90, 95, 168
TickMark.....	93, 94, 95
TickMark.....	53, 93, 94, 95, 242
Time Panel Meters...35, 37, 39, 45, 108, 124, 125, 127	
Time Panel Meters.....	
Time Panel Meters.....	125, 126
TimeAutoScale.....	65, 66, 95
TimeAxis.....	67, 70, 73, 89, 95
TimeAxisLabels.....	72, 73, 95
TimeCoordinates.....	64, 95
TimeCoordinates.....	7, 12, 18, 64, 95, 225, 226, 277
TimeGroupDataset. 35, 36, 62, 63, 66, 95, 97, 99, 100,	
101, 102, 103, 105, 106	
TimeLabel.....	39, 90, 95, 125
TimeLabel.....	39, 90, 91, 95, 125, 127
TimeScale.....	63, 95
TimeScale.....	63, 95
TimeSimpleDataset.....	62, 66, 95
ToolTips.....	91, 92, 95
Trial License.....	
Trial License.....	ii
tsconfig.json.....	
tsconfig.json.....	15, 16, 300, 315, 324, 325
tutorial.....	
tutorial.....	2
Ubuntu.....	
Ubuntu.....	4, 5, vi, 300, 332, 333
UserControl.....	36
UserCoordinates.....	63, 64, 95
UserCoordinates.....	63, 64, 95
Vertical Scrolling.....	
Vertical Scrolling...29, 34, 35, 45, 48, 54, 56, 217,	
218, 221, 222, 223, 227, 228	
Visual Basic.....29, 276, 278, 279, 293, 298	
279, 298	
Visual Studio.....	
Visual Studio....4, 5, 6, 15, 29, 301, 302, 314, 316,	
331, 332, 333, 334	
Visual Studio Code.....	
Visual Studio Code...4, 5, 6, 15, 29, 301, 302, 314,	
316, 331, 332, 333	
301	
Web Applications.....	
web applications.....	23, 333, 334
WordPress.....	
Wordpress.....	23
WordPress.....	iv, 23
WorkingCoordinates.....	63, 64, 95
WorkingCoordinates.....	63, 64, 95
WorldCoordinates.....	63, 64, 95
WorldCoordinates.....	63, 64, 95
Zoom.....	279
Zooming.....92, 95, 96, 272, 273, 275, 276, 277, 279	
ChartZoom.....	273, 274
zoom.....	273, 274
Zoom....1, v, vi, 29, 92, 95, 96, 272, 273, 274, 275,	
276, 277, 278, 279	
ZOOM.....	274
"	ii, iii

