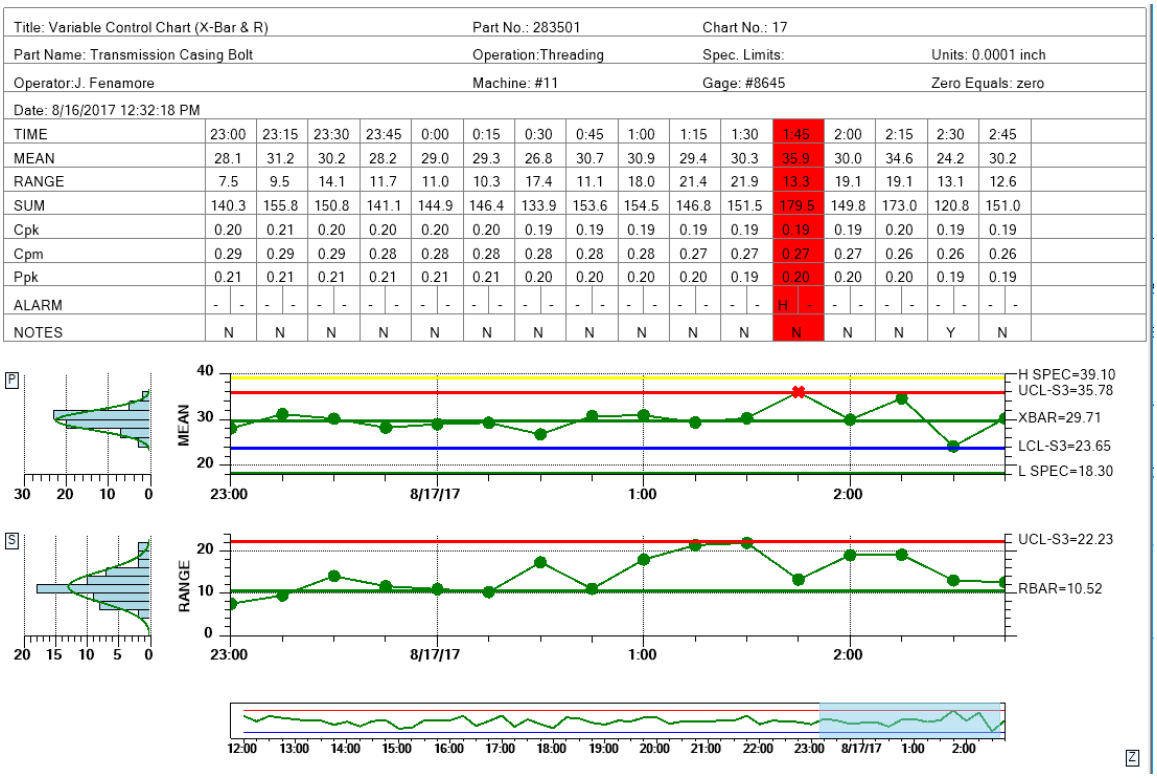
QCSPCChart SPC Control Chart Tools for JavaScript/TypeScript



Title: Variable Control Chart (X-Bar & R)					Part No.: 283501				Chart No.: 17							
Part Name: Transmission Casing Bolt					Operation:Threading				Spec. Limits:					Units: 0.0001 inch		
Operator:J. Fenamore					Machine: #11				Gage: #8645					Zero Equals: zero		
Date: 8/16/2017 12:32:18 PM																
TIME	23:00	23:15	23:30	23:45	0:00	0:15	0:30	0:45	1:00	1:15	1:30	1:45	2:00	2:15	2:30	2:45
MEAN	28.1	31.2	30.2	28.2	29.0	29.3	26.8	30.7	30.9	29.4	30.3	35.9	30.0	34.6	24.2	30.2
RANGE	7.5	9.5	14.1	11.7	11.0	10.3	17.4	11.1	18.0	21.4	21.9	13.3	19.1	19.1	13.1	12.6
SUM	140.3	155.8	150.8	141.1	144.9	146.4	133.9	153.6	154.5	146.8	151.5	179.5	149.8	173.0	120.8	151.0
Cpk	0.20	0.21	0.20	0.20	0.20	0.20	0.19	0.19	0.19	0.19	0.19	0.19	0.19	0.20	0.19	0.19
Cpm	0.29	0.29	0.29	0.28	0.28	0.28	0.28	0.28	0.28	0.27	0.27	0.27	0.27	0.26	0.26	0.26
Ppk	0.21	0.21	0.21	0.21	0.21	0.21	0.20	0.20	0.20	0.20	0.19	0.20	0.20	0.20	0.19	0.19
ALARM	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	H -	- -	- -	- -	- -
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N

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 7/12/2021 Rev. 3.01

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 and 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. polices 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

Chapter 1. Introduction

QCSPCChart for JavaScript/TypeScript (JSTS-SPC2-DEVR)

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

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

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

Relation to QCSPCChart for JavaScript (JS-SPC2-DEVR)
We (Quinn-Curtis) have a related product which is also written in JavaScript. It is called the QCSPCChart for JavaScript. It utilizes a framework available from Google called GWT (Google Web Toolkit), where you can write your source code in the Java programming language, and the resulting source code is transpiled into JavaScript. In addition, GWT also includes many additional support libraries necessary to transpile Java code into JavaScript. The downside of this approach for us is that due to name mangling and optimization, the functions in the resulting JavaScript library are not callable from JavaScript running in a browser, or from an external JavaScript file. Instead, the creation of charts using the QCSPCChart library takes place using a JSON template. This limits the overall programmability of web applications using the QCSPCChart library. After 5 years we feel that this product, along with GWT, has run its course, and we are replacing it with, this, the QCSPCChart for JavaScript/TypeScript software. Since QCSPCChart

for JavaScript/TypeScript is written entirely in TypeScript, with no dependencies on external libraries, it is a better choice for developers going forward.

Transpiling vs Compiling

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

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

Tutorials

Chapter 11 is a tutorial that describes how to define a simple SPC chart and deploy it to a web page. Read the first couple of chapters, and then the tutorial.

HTML

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

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

HTML5

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

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

JavaScript

JavaScript is an interpreted programming language created to make web browsers more dynamic and interactive. It was originally developed by Netscape in 1995 as a feature of their web browser. While it includes "Java" as the root of its name, that was a 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 QCSPCChart 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 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, for reasons already mentioned.

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 add true object-oriented classes, and strong type checking to JavaScript, critical for developing large applications. It also supports source level debugging of both TypeScript source code, and the transpiled JavaScript source code. We encourage you read about TypeScript on the web : https://www.typescriptlang.org/.

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

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

Visual Studio Code

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

```
npm install -g typescript
```

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

npm -v

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

Using QCSPCChart 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 QCSPCChartTS is spelled in lower case: qcspcchartts.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/TypescriptExamples/XBarRChart for the XbarRChart example.

Programming QCSPCChart directly from JavaScript

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

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';


export async function BuildXBarRChart(canvasid) {

    var htmlcanvas = document.getElementById(canvasid);
    var spccharttype = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    var subgroupsize = 5;
    var numberpointsinview = 12;
    var charttitle = "XBar-R Example";


    var xbarrchart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);

    xbarrchart.setPreferredSize(800, 600);
```

```
    xbarrchart.setGraphStopPosX(0.825);
    xbarrchart.setGraphStartPosX(0.18);

xbarrchart.setChartAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_SYM
BOL);
    xbarrchart.setEnableScrollBar(true);

    xbarrchart.setEnableCategoryValues(true);
    xbarrchart.setEnableCalculatedValues(true);
    xbarrchart.setEnableAlarmStatusValues(false);
    xbarrchart.setEnableChartToggles(true);

xbarrchart.setTableAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_BAR
);

xbarrchart.setHeaderStringsLevel( QCSPCChartTS.SPCControlChartData.HEADER_STRINGS_
LEVEL1);

    xbarrchart.getChartData().setTitle (charttitle);
    xbarrchart.getChartData().setChartDescriptior("XBar-R");
    xbarrchart.setEnableDisplayOptionToggles(true);

    var numssampleintervals = 100;
    var chartmean = 30;
    var chartsigma = 5;

    SimulateData(xbarrchart, numssampleintervals, chartmean, chartsigma);


    // Calculate the SPC control limits for both graphs of the current SPC chart
(X-Bar R)
    xbarrchart.autoCalculateControlLimits();


    // Scale the y-axis of the X-Bar chart to display all data and control limits
    xbarrchart.autoScalePrimaryChartYRange();
    // Scale the y-axis of the Range chart to display all data and control limits
    xbarrchart.autoScaleSecondaryChartYRange();
    // Rebuild the chart using the current data and settings
    xbarrchart.rebuildChartUsingCurrentData();

}


function SimulateData(spcchart, count, mean, sigma) {
    // batch number for a given sample subgroup
    var batchCounter = 0;
    var i;
    var timestamp = new Date();
    if (!spcchart) return;
    if (!spcchart.getChartData()) return;
    for (i = 0; i < count; i++) {
        // Simulate a sample subgroup record
        var samples =
spcchart.getChartData().simulateMeasurementRecordMeanRange(mean, sigma);
        // Update chart data using i as the batch number
        batchCounter = i;
        var note = "";
        if ((i % 5) == 0) note = "This is a note";
        // Add a new sample record to the chart data

spcchart.getChartData().addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter
, timestamp, samples, note);
        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }

}
```

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

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
```

The "../../QCSPCChartTS" folder specification means to back up two directory levels and go to the QCSPCChartTS folder, where the qcspcchartts.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/JavascriptExamples/XBarRChart/XBarRChart.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/JavascriptExamples/XBarRChart/XBarRChart.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 QCSPCChartTS** statement, all JavaScript objects found in the library are located in the namespace QCSPCChartTS, which prevents them from conflicting with any other libraries you may be using which have similar named objects. The file qcspcchartts.js contains the entire QCSPCChart library.

A new SPC Chart is created inside the BuildXBarRChart function using the call:

```
    var xbarrchart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
```

passing in the HTML canvas you have reserved for the chart, the other basic SPC chart setup information, such as chart type, subgroup size and number of points in the initial view.

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

```
SPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartTypeSu
bgroupSize
```

which is the static SPCBatchVariableControlChart class constructor which specifies the chart type, and subgroup size.

The programmer customizes the charts by making function calls into the resulting xbarrchart objects, which is of type SPCBatchVariableControlChart.

```
    xbarrchart.setPreferredSize(800, 600);

    xbarrchart.setGraphStopPosX(0.825);
    xbarrchart.setGraphStartPosX(0.18);

xbarrchart.setChartAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_SYM
BOL);
    xbarrchart.setEnableScrollBar(true);

    xbarrchart.setEnableCategoryValues(true);
    xbarrchart.setEnableCalculatedValues(true);
    xbarrchart.setEnableAlarmStatusValues(false);
    xbarrchart.setEnableChartToggles(true);
```

Note that all constants used in the setup, and all QCSPCChart types in general, need to be prefixed by the QCSPCChartTS namespace. **Note:** The identifier QCSPCChartTS is not fixed, it is just the namespace specified in the **import** statement at the top of the file. 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 all calls into the QCSPCChart objects are function calls (set., get..), similar to Java, and unlike C# which makes extensive use of getter/setter properties. 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.

In this example the BuildXBarRChart function creates the initial graph. Since a graph without data is not of much use, simulated data is added to the chart using the call to the SimulateData method, also found in the XBarRChart.js file.

```
function SimulateData(spcchart, count, mean, sigma) {
    // batch number for a given sample subgroup
    var batchCounter = 0;
```

```
    var i;
    var timestamp = new Date();
    if (!spcchart) return;
    if (!spcchart.getChartData()) return;
    for (i = 0; i < count; i++) {
        // Simulate a sample subgroup record
        var samples =
spcchart.getChartData().simulateMeasurementRecordMeanRange(mean, sigma);
        // Update chart data using i as the batch number
        batchCounter = i;
        var note = "";
        if ((i % 5) == 0) note = "This is a note";
        // Add a new sample record to the chart data

spcchart.getChartData().addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter
, timestamp, samples, note);
        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }

}
```

In it you will find that simulated sample records are created using the spcchart.getChartData().simulateMeasurementRecordMeanRange function call. The resulting sample array is passed to spcchart.getChartData().addNewSampleRecordBatchNumberDateSamplesNotes function.

So most of the work is done in the BuildXBarRChart function of the XBarRChart.js file. But it needs to be invoked from the HTML file you want to place the chart in. Look at the XBarRChart/XBarRChart.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 SPC chart in.

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

    <div id="buttondiv">
        <button type="button" id="xbarr_menuitem">XBar-R</button>
    </div>

    <div id="canvasdiv">
        <canvas id="spcChartCanvas1" width="800" height="600"></canvas>E
     </div>
        <script type="module">

          import { BuildXBarRChart} from './XBarRChart.js';


          function xbarrchart() {
              BuildXBarRChart("spcChartCanvas1");
          }
```

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

        </script>

        <br>
        <br>

</body>
</html>
```

In this example the button click for the Xbar-R button invokes the JavaScript *xbarrchart* function call inside the script section. That function in turn calls the BuildXBarRChart in the imported XbarRChart.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 placed.

```
    <div id="canvasdiv">
        <canvas id="spcChartCanvas1" width="800" height="600"></canvas>
     </div>
```

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

[JavaScript]

```
let htmlcanvas = document.getElementById(canvasid);
```

[TypeScript]

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

Note: The **import** statement found in the script uses type=”module”. This is part of the 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 BuildXBarRChart function from the XbarRChart.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 xbarrchart function in the script block has not yet been loaded. Instead is it assigned explicitly in the javascript code, after the XbarRChart.js module has been loaded.

```
          document.getElementById("xbarr_menuitem").onclick = xbarrchart;
```

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

Programing QCSPCChart 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 Aug. 2019) 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/TypescriptExamples folder. Each example folder contains an HTML file to load into a browser to test the example, a *.ts file which loads the qcspcchartts.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 (XbarRChart) using your compiler (Visual Studio Code in this case). This makes the folder the root location of the current TypeScript project. It looks to the folder for a tsconfig.json file for the compiler options it will use to compile the TypeScript code within the project. There are a many compiler options and you will need to study those for advanced projects. Here is one link: https://www.typescriptlang.org/docs/handbook/compiler-options.html to start learning from. Our examples use a very simplified tsconfig.json file which basically tells the compiler to compile all *.ts files it finds in the folder, and place the resulting *.js files in the same folder.

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

}
```

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

compilerOptions:

target: es6 - Specify ECMAScript target version. Permitted values are 'es3', 'es5', 'es6', 'es2015', 'es2016', 'es2017', 'es2018', 'es2019', 'es2020' or 'esnext'. We use es6 in all of our tsconfig.json files, but es5 to esnext all seem to work.

lib: [es6, dom] - Specify library file to be included in the compilation. Requires TypeScript version 2.0 or later.

sourceMap: true - Generates corresponding '.map' file., used in debugging

outDir: "./" - send the compiler output (*.js, *.map, *.d.ts) to to this location), the root directory of the project in this case. Specifying "./" places the compiler output files (*.js, *.map, *.d.ts) in the project folder, next to the source *.ts files. In many cases you will want to put them somewhere else. Here are some other examples:
"./built" - send files to the current folders /built folder.
"../" - up one directory level from the current folder.
"../built" - up one directory level from the current folder into the /built folder there.
Note: Where ever you send them, that is where you should access them from (unless you copy them somewhere else). So the HTML file you use must reference the imported *.js file from the correct folder, which is not necessarily the project folder.

declaration: true – generate the *.d.ts map file, which enables TypeScript debugging in the browser

module: es6 - Specify module code generation: 'none', 'commonjs', 'amd', 'system', 'umd', 'es2015' or 'esnext'.

strict: true - Enable all strict type checking options. Requires TypeScript version 2.3 or later. Not easy to work with, but it does catch countless bad programming errors involving use of null or uninitialized objects. The underlying qcspcchartts.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 (XbarRChart) is defined with a BuildXBarRChart member function.

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';


export class XBarRChart {

    public constructor() {

    }



    public async  BuildXBarRChart(canvasid: string) {

        let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
        let spccharttype: number =
QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
        let subgroupsize: number = 5;
        let numberpointsinview: number = 12;
        let charttitle: string = "XBar-R Example";


        let xbarrchart: QCSPCChartTS.SPCChartBase | null =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
        if (!xbarrchart) return;

        xbarrchart.setPreferredSize(800, 600);

        xbarrchart.setGraphStopPosX(0.825);
        xbarrchart.setGraphStartPosX(0.18);

xbarrchart.setChartAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_SYM
BOL);
        xbarrchart.setEnableScrollBar(true);

        xbarrchart.setEnableCategoryValues(true);
        xbarrchart.setEnableCalculatedValues(true);
        xbarrchart.setEnableAlarmStatusValues(false);
        xbarrchart.setEnableChartToggles(true);

xbarrchart.setTableAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_BAR
);

xbarrchart.setHeaderStringsLevel(QCSPCChartTS.SPCControlChartData.HEADER_STRINGS_L
EVEL1);

        xbarrchart.setEnableDisplayOptionToggles(true);
        let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarrchart.getChartData();
        if (chartdata) {
            chartdata.setTitle(charttitle);
            chartdata.setChartDescriptior("XBar-R");
        }
        let numssampleintervals: number = 100;
        let chartmean: number = 30;
        let chartsigma: number = 5;

        this.SimulateData(xbarrchart, numssampleintervals, chartmean, chartsigma);
```

```
        // Calculate the SPC control limits for both graphs of the current SPC
chart (X-Bar R)
        xbarrchart.autoCalculateControlLimits();


        // Scale the y-axis of the X-Bar chart to display all data and control
limits
        xbarrchart.autoScalePrimaryChartYRange();
        // Scale the y-axis of the Range chart to display all data and control
limits
        xbarrchart.autoScaleSecondaryChartYRange();
        // Rebuild the chart using the current data and settings
        xbarrchart.rebuildChartUsingCurrentData();

    }



    SimulateData(spcchart: QCSPCChartTS.SPCChartBase | null, count: number, mean:
number, sigma: number) {
        // batch number for a given sample subgroup
        let batchCounter = 0;
        let i = 0;
        let timestamp = new Date();
        if (!spcchart) return;
        let chartdata: QCSPCChartTS.SPCControlChartData | null =
spcchart.getChartData();
        if (!chartdata) return;
        for (i = 0; i < count; i++) {
            // Simulate a sample subgroup record
            let samples = chartdata.simulateMeasurementRecordMeanRange(mean,
sigma);
            // Update chart data using i as the batch number
            batchCounter = i;
            let note = "";
            if ((i % 5) == 0) note = "This is a note";
            // Add a new sample record to the chart data
            chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, note);
            // Simulate passage of timeincrementminutes minutes
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }

    }

}
```

The "../../QCSPCChartTS" folder specification means to back up two directory levels and go to the QCSPCChartTS folder, where the qcspcchartts.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/JavascriptExamples/XBarRChart/XBarRChart.html

You can usually replace 127.0.0.1 with *localhost*.

http://localhost:8080/JavascriptExamples/XBarRChart/XBarRChart.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 QCSPCChartTS statement, all JavaScript objects found in the library are located in the namespace QCSPCChartTS, which prevents them from conflicting with any other libraries you may be using which have similar named objects. The file qcspcchartts.js contains the entire QCSPCChart library

In it you will find that simulated sample records are created using the xbarrchart.getChartData().simulateMeasurementRecordMeanRange function call. The resulting sample array is passed to chartdata.addNewSampleRecordBatchNumberDateSamplesNotes function.

So most of the work is done in the BuildXBarRChart member function of the XbarRChart clsss. But it needs to be invoked from the HTML file you want to place the chart in. Look at the XbarRChart/XbarRChart.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 SPC chart in. The example references the XbarRChart.js file found in the JavascriptExamples/XBarRChart folder.

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

    <div id="buttondiv">
        <button type="button" id="xbarr_menuitem">XBar-R</button>
    </div>

    <div id="canvasdiv">
        <canvas id="spcChartCanvas1" width="800" height="600"></canvas>
     </div>
        <script type="module">
```

```
    import { XBarRChart} from './XBarRChart.js';

     var spcchart = new XBarRChart();


          function xbarrchart() {
            spcchart.BuildXBarRChart("spcChartCanvas1");
          }


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

        </script>

        <br>
        <br>

</body>
</html>
```

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

```
          document.getElementById("xbarr_menuitem").onclick = xbarrchart;
```

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

Programing QCSPCChart within Wordpress

Using QCSPCChart 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 (**BuildChart** in the example below) inside that file to build the chart. In that case the embedded code might look something like:

```
<canvas id="spcCanvas1" width="600" height="600"></canvas>
    <script type="module">
      import { spc_setupparams, BuildChart} from
'http://spcchartsonline.com/QCSPCChartWebApp/src/BasicBuildChart1.js';
    spc_setupparams.numberpointsinview = 25;
       spc_setupparams.detaildisplaymode = 0;
       spc_setupparams.canvas_id = "spcCanvas1";
       spc_setupparams.initialdata = [
[1.3235,    1.4128, 1.6744, 1.4573, 1.6914],
[1.4314,    1.3592, 1.6075, 1.4666, 1.6109],
[1.4284,    1.4871, 1.4932, 1.4324, 1.5674],
[1.5028,    1.6352, 1.3841, 1.2831, 1.5507],
[1.5604,    1.2735, 1.5265, 1.4363, 1.6441],
[1.5955,    1.5451, 1.3574, 1.3281, 1.4198],
[1.6274,    1.5064, 1.8366, 1.4177, 1.5144],
[1.419,     1.4303, 1.6637, 1.6067, 1.5519],
[1.3884,    1.7277, 1.5355, 1.5176, 1.3688],
[1.4039,    1.6697, 1.5089, 1.4627, 1.522],
[1.4158,    1.7667, 1.4278, 1.5928, 1.4181],
[1.5821,    1.3355, 1.5777, 1.3908, 1.7559],
[1.2856,    1.4106, 1.4447, 1.6398, 1.1928],
[1.4951,    1.4036, 1.5893, 1.6458, 1.4969],
[1.3589,    1.2863, 1.5996, 1.2497, 1.5471],
[1.5747,    1.5301, 1.5171, 1.1839, 1.8662],
[1.368,     1.7269, 1.3957, 1.5014, 1.4449],
[1.4163,    1.3864, 1.3057, 1.621,  1.5573],
[1.5796,    1.4185, 1.6541, 1.5116, 1.7247],
[1.7106,    1.4412, 1.2361, 1.382,  1.7601],
[1.4371,    1.5051, 1.3485, 1.567,  1.488],
[1.4738,    1.5936, 1.6583, 1.4973, 1.472],
[1.5917,    1.4333, 1.5551, 1.5295, 1.6866],
[1.6399,    1.5243, 1.5705, 1.5563, 1.553],
[1.5797,    1.3663, 1.624,  1.3732, 1.6887]];
BuildChart();
    </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://spcchartsonline.com/QCSPCChartWebApp/XBarRInlinePage.html"
id="iframe_id" width="100%" height = "600"]
```

If your intended use is a simple SPC 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://spcchartsonline.com .

Programing QCSPCChart using Angular

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. QCSPCChart 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 SPC chart to appear. You can add as many canvas elements as you want and put unique SPC chart in each. Just keep the variable name (#chartCanvas1 in the example below) of the canvas elements unique. Something like this:

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

<div id="canvasdiv">

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

In 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. The **BuildXBarRChart** and **SimulateData** functions are exactly the same as similar function in all of our example programs.

```
import { Component, ViewChild, ElementRef, AfterViewInit } from '@angular/core';
import * as  QCSPCChartTS from '../../../QCSPCChartTS/qcspcchartts.js';


@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<HTMLCanva
sElement>;
  title = 'Hello World!';

  private spcchart: QCSPCChartTS.SPCChartBase | null = null;

  constructor() {

  }
```

```
  buttonClickFunction() {
    this.BuildXBarRChart();
  }

  ngAfterViewInit() {

  }

  public BuildXBarRChart() {

    let htmlcanvas: HTMLCanvasElement | null = <HTMLCanvasElement>this.chartCanvas
1.nativeElement;
    let spccharttype: number = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    let subgroupsize: number = 5;
    let numberpointsinview: number = 12;
    let charttitle: string = "XBar-R Example";

  let xbarrchart: QCSPCChartTS.SPCChartBase = QCSPCChartTS.SPCBatchVariableControl
Chart.newSPCBatchVariableControlChartChartTypeSubgroupSize(htmlcanvas, spccharttyp
e, subgroupsize, numberpointsinview);

.
.
.

    // Rebuild the chart using the current data and settings
    xbarrchart.rebuildChartUsingCurrentData();


  }


  SimulateData(spcchart: QCSPCChartTS.SPCChartBase, count: number, mean: number, s
igma: number) {
    // batch number for a given sample subgroup
    let batchCounter = 0;
.
.
.

  }

}
```

The important Angular specific elements of this module are the following:

Make sure that the import statement points to a directory location where the QCSPCChartTS/qcspcchartts.js library file can be found.

```
import * as  QCSPCChartTS from '../../../QCSPCChartTS/qcspcchartts.js';
```

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<HTMLCanva
sElement>;
```

When you actually build the SPC 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:

```
    let htmlcanvas: HTMLCanvasElement | null = <HTMLCanvasElement>this.chartCanvas
1.nativeElement;
```

The resulting *htmlcanvas* TypeScript variable can then be used to pass the canvas element into the one of the many newSPCBatchVariable... (or other) constructors.

```
 let xbarrchart: QCSPCChartTS.SPCChartBase =

QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
Subgroupsize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
    if (!xbarrchart) return;
```

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 by running "ng serve" from the root of the project folder. Then point your web browser to "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) in the TypeScriptExamplesQCSPCChart/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 the entire project.

SPC Control Chart Tools Background

In a competitive world environment, where there are many vendors selling products and services that *appear* to be the same, **quality**, both real and perceived, is often the critical factor determining which product wins in the marketplace. Products that have a reputation for higher quality command a premium, resulting in greater market share and profit margins for the manufacturer. Low quality products not only take a big margin hit at the time of sale, but also taint the manufacturer with a reputation that will hurt future sales, regardless of the quality of future products. Users have a short memory. A company's quality reputation is only as good as the quality of its most recent product.

The measurement, control and gradual improvement of quality is the goal of all quality systems, no matter what the name. Some of the more common systems are known as **SCC** (Statistical Quality Control) **Quality Engineering**, **Six-Sigma**, **TQM** (Total Quality Management), **TQC** (Total Quality Control), **TQA** (Total Quality Assurance) and **CWQC** (Company- Wide Quality Control). These systems work on the principle that management must integrate quality into the basic structure of the company, and not relegate it to a Quality Control group within the company. Historically, most of the innovations in quality systems took place in the 20th century, with pioneering work carried out by Frederick W. Taylor (Principles of Scientific Management), Henry Ford (Ford Motor), W. A. Shewhart (Bell Labs), W. E. Deming (Department of Agriculture, War department, Census Bureau), Dr. Joseph M. Juran (Bell Labs), and Dr. Armand V.

Feigenbaum among others. Most quality control engineers are familiar with the story of how the statistical quality control pioneer, W. E. Deming, frustrated that US manufactures only gave lip service to quality, took the quality system he developed to Japan, where it was embraced with almost religious zeal. Japanese industry considers Deming a national hero, where his quality system played a major role in the postwar expansion of the Japanese economy. Twenty to thirty years after Japan embraced his methods, Deming found a new audience for his ideas at US companies that wanted to learn *Japanese* methods of quality control.

All quality systems use Statistical Process Control (**SPC**) to one degree or another. SPC is a family of statistical techniques used to track and adjust the manufacturing process in order to produce gradual improvements in quality. While it is based on sophisticated mathematical analysis involving sampling theory, probability distributions, and statistical inferences, SPC results can usually be summarized using simple charts that even management can understand. SPC charts can show how product quality varies with respect to critical factors that include things like batch number, time of day, work shift personal, production machine, and input materials. These charts have odd names like X-Bar R, Median Range, Individual Range, Fraction Number Non-Conforming, and NP. The charts plot some critical process variable that is a measurement of product quality and compares it to predetermined limits that signify whether or not the process is working properly.

Initially, quality control engineers create all SPC charts by hand. Data points were painstakingly gathered, massaged, summed, averaged and plotted by hand on graph paper. It is still done this way in many cases. Often times it is done by the same factory floor personal who control the process being measured, allowing them to "close the loop" as quickly as possible, correcting potential problems in the process before it goes out of control. Just as important, SPC charts tell the operator when to leave the process alone. Trying to micro-adjust a process, when the process is just exhibiting normal random fluctuations in quality, will often drive the process out of control faster than leaving it alone.

The modern tendency is to automate as much of the SPC chart creation process as possible. Electronic measuring devices can often measure quality in real-time, as items are coming off the line. Usually some form of sampling will be used, where one of every N items is measured. The sampled values form the raw the data used in the SPC chart making process. The values can be entered by hand into a SPC chart making program, or they can be entered directly from a file or database connection, removing the potential for transcription errors. The program displays the sampled data in a SPC chart and/table where the operator or quality engineer can make a judgment about whether or not the process is operating in or out of control.

Usually the SPC engineer tasked with automating an existing SPC charting application has to make a decision about the amount of programming he wants to do. Does he purchase an application package that implements standard SPC charts and then go about defining the charts using some sort of menu driven interface or wizard. This is probably the most expensive in terms of up front costs, and the least flexible, but the cheapest in

development costs since a programmer does not have to get involved creating the displays. Another choice is to use a general purpose spreadsheet package with charting capability to record, calculate, and display the charts. This is probably a good choice if your charting needs are simple, and you are prepared to write complicated formulas as spreadsheet entries, and your data input is not automated. Another choice is writing the software from scratch, using a charting toolkit like our ***QCChart2D*** software as the base, and creating custom SPC charts using the primitives in the toolkit. This is cheaper up front, but may be expensive in terms of development costs. Often times the third option is the only one available because the end-user has some unique requirement that the pre-packaged software can't handle, hence everything needs to programmed from scratch.

The current SPC trend is for data to be centralized on a server in a database, and the display to be localized on the client computer. The local display on the client can be a desktop application, or a web-based application. We have several versions of QCSPCChart for the display of SPC data in a desktop application: specifically for .Net, WPF (Windows Presentation Foundation), and Java. For mobile applications, we have QCSPCChart for Android. For web based applications we have this software (QCSPCChart for JavaScript/TypeScript) which is meant to replace an earlier version we had named (QCSPCChart for JavaScript). The older version created SPC charts using JSON templates, while this version interacts with the underlying QCSPSChart JavaScript directly.

Quinn-Curtis SPC (Statistical Process Control) Software

We have created a library of SPC routines that represents an intermediate solution. Our SPC software still requires an intermediate level programmer, but it does not require advanced knowledge of SPC or of charting. Built on top our ***QCChart2D,*** it implements templates and support classes for the following SPC charts and control limit calculations.

Variable Control Charts Templates

Fixed sample size subgroup control charts

- X-Bar R – (Mean and Range Chart)
- X-Bar Sigma (Mean and Sigma Chart)
- Median and Range (Median and Range Chart)
- X-R (Individual Range Chart)
- EWMA (Exponentially Weighted Moving Average Chart)
- MA (Moving Average Chart)
- MAMR (Moving Average / Moving Range Chart)
- MAMS (Moving Average / Moving Sigma Chart)
- CuSum (Tabular Cumulative Sum Chart)
- Xbar-R-MR (Mean-Range-Moving Range Chart)

Variable sample size subgroup control charts

- X-Bar Sigma (Mean and Sigma Chart)

Attribute Control Charts Templates

Fixed sample size subgroup control charts
- p-Chart (Fraction or Percent of Defective Parts)
- np-Chart (Number of Defective Parts)
- c-Chart (Number of Defects)
- u-Chart (Number of Defects per Unit)
- DPMO-Chart (Number Defects per Million)

Variable sample size subgroup control charts
- p-Chart (Fraction or Percent of Defective Parts)
- u-Chart (Number of Defects per Unit)

Analysis Chart Templates
Frequency Histograms

SPC Support Calculations
Array statistics (sum, mean, median, range, standard deviation, variance, sorting)

SPC Control Limit Calculations
High and low limit control calculations for X-Bar R, X-Bar Sigma, Median and Range, X-R, p, np, c and u charts

SPC Process Capability Calculations
Variable Control Charts include Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppl, Ppu, and Ppk process capability statistics

SPC Control Named Rule Sets
Western Electric (WECO) Runtime and Supplemental Rules
Nelson
AIAG
Juran
Hughes
Gitlow
Westgard
Duncan

Design Considerations

- Minimal programming required – create SPC charts with a few lines of JavaScript or TypeScript code
- Integrated frequency histograms support – Display frequency histograms of sampled data, displayed side-by-side, sharing the same y-axis, with the SPC chart.
- Charts Header Information – Customize the chart display with job specific information, for example: Title, Operator, Part Number, Specification Limits, Machine, etc.

- Table display of SPC data – Display the sampled and calculated values for a SPC chart in a table, directly above the associated point in the SPC chart, similar to standardized SPC worksheets.

- Automatic calculation of SPC control limits – Automatically calculate SPC control limits using sampled data, using industry standard SPC control limit algorithms unique to each chart type.

- Automatic y-Axis scaling – Automatically calculated the y-axis scale for SPC charts, taking into account sampled and calculated data points, and any control limit lines added to the graph.

- Alarms – When monitored value exceeds a SPC control limit it can trigger an event that vectors to a user-written alarm processing delegate.

- SPC Process Capability Calculations -Variable Control Charts include Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppl, Ppu, and Ppk process capability statistics

- Notes – The operator can view or enter notes specific to a specific sample subgroup using a special notes tooltip.

- Data tooltips – The operator can view chart data values using a simple drill-down data tooltip display. The Data tooltips can optionally display sample subgroup data values and statistics, including process capability calculations (Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppl, Ppu, and Ppk) and customized using notes that have been entered for the sample subgroup.

- Scrollable view – Enable the scroll bar option and scroll through the chart and table view of the SPC data for an unlimited number of sample subgroups.

- Other, optional features – There are many optional features that SPC charts often use, including:

- Multiple SPC control limits, corresponding to +-1, 2 and 3 sigma limits.

- Support for named control rule sets: WE, Nelson, AIAG, Juran, Hughes, Duncan, Westgard, and Gilow

- Support for custom control rule sets based on our predefined templates.

- Scatter plots of all sampled data values on top of calculated means and medians.

- Data point annotations

- Alarm forcing – An sample interval can be forced into alarm

- Alarm highlighting – Symbols can change color or shape when a sample interval is in an alarm condition.

- Variable control limits and specification limits for all chart types

The **SPC Control Chart Tools for JavaScript/TypeScript** is a library of classes that integrate **a** charting software with tables, data structures and specialized rendering routines used for the static and dynamic display of SPC charts. The SPC charts are pre-programmed classes that create, manage and display the graphs and tables corresponding to major SPC control chart types. Each class can be further customized by calling member functions, allowing tremendous flexibility in the look of the SPC charts.

Web-Based Versions of QCSPCChart

Not counting this, the JavaScript/TypeScript version of QCSPCChart, we have five other variants of QCSPCChart which can be used to display charts in web browsers. Each variant has weaknesses which prevent it from being a true cross-platform, cross-browser method used to implement a truly interactive chart.

Web Browser Application Frameworks which are not JavaScript

.Net – Using the regular .Net version, you can write ASP.Net applications which run on a server in *headless* mode. *Headless* means that custom chart images are created on demand and converted to jpg or png format without being specifically rendered to a server workstation monitor. The converted images are transferred to the client-side workstation and displayed in an HTML (or Asp .Net equivalent) image element. The drawback of this techniques is that the chart is not interactive. Data tooltips, editing, scrolling, and real-time updates are all either limited, or non-existent. Also, the host server is limited to ones supporting Asp .Net program development. This requires Administrator privileges on the host server not granted to many developers using shared servers on third party hosts.

Java – Java has a long history of being used to write desktop, server and client side applications. Using Java, it is possible to write cross platform, client-side web browser applications which render graphics directly on the local workstation. When a Java application is run in a browser, and the browser host does not have a Java run-time installed, a workstation specific subset of the Java run-time environment is downloaded into the client memory, and that run-time is used to execute (interpret) the Java application program. Unfortunately, it contains many security holes which render it unusable in high security applications. Even though Sun (the controlling authority behind Java) seems to issue new revisions of Java weekly to fix old security flaws, new holes seem to appear just as fast. Java is simply to easy for malicious hackers to subvert. Over the last several years, web browsers have carefully reduced the number of Java features they support, making it a moving target to write against, since programs which work under IE 6 may not work under IE8, IE9 and IE10. Modern browsers are turning Java support off as part of their recommended security settings.

WPF – WPF and Silverlight are very similar. While Silverlight is strictly a framework for writing rich internet applications, WPF can be used to write applications for both the desktop and the internet. It has much in common with Silverlight, and both share an XAML-based method of creating the user interface for an application, and both are

normally programmed using a .Net language (C# or VB). They do not share a common run-time though. So WPF requires a different run-time plug-in than Silverlight. Also, WPF browser applications will only run on workstations which have a recent version of the .Net run-time installed. This means it has more limited browser support than Silverlight, with no support of Apple workstations, no support for Linux, and no support for mobile applications (IOS and Android). There are also rumors that Microsoft is placing WPF in the same *no further development* category as Silverlight.

Silverlight – Silverlight was intended to be Microsoft's answer to Java as a client-side programming language for web browser applications. First introduced in 2007, its most recent version is Version 5.0, released in 2011. Silverlight run-time plug-ins are available for browsers running on Windows and OSX (Apple) based workstations. Silverlight was a significant innovation for Microsoft. It does provide a very powerful application framework for writing rich internet applications. Unfortunate, early attempts (Moonlight) to port the run-time to Linux-based browsers have been abandoned. Also, no plug-ins were ever created for browsers running on mobile devices running IOS (Apple) and Android (Google). Even Microsoft's own Mobile phone does not support Silverlight. Microsoft has stopped development work on Silverlight. They will probably support it for years to come, because of the large installed base, but it is a dead end development-wise.

Web Browser Application Frameworks which use JavaScript

JavaScript using the GWT Framework - GWT (Google Web Toolkit) is a web application framework available from Google, where you can write your source code in Java, and the resulting source code is transpiled into JavaScript. In addition, GWT also includes many additional support libraries necessary to transpile Java code into JavaScript. The downside of this approach for us is that due to name mangling and optimization, the functions in the resulting JavaScript library are not callable from JavaScript running in a browser, or from external JavaScript file.

JavaScript using ES6 modules- Import JavaScript libraries, either raw JavaScript code, or JavaScript code transpiled from TypeScript, directly into web applications. That is what this software utilizes.

Directory Structure of QCSPCChart for JavaScript/TypeScript

The **SPC Control Chart Tools for JavaScript/TypeScript** software is located in the Quinn-Curtis/JSTS folder.

Drive:

Quinn-Curtis/ - Root directory

JSTS/ - Root of the JavaScript/TypeScript folder

Docs/ - documentation directory

QCSPCChartJSTSDoc.pdf – This document, the User guide

docs/

QCSPCChartJSTSClassesIndex.html – Class descriptions based on the internal library documentation

QCSPCChartTS/ - A folder containing the qcspcchartts.js library.

qcspccharts.js - the qcspcchartts.js library

qcspccharts.d.ts – a map indexing the original TypeScript source (qcspcchartts.ts) to the resultant qcspcchartts.js file.

qcspccharts.js.map – a source map used for debugging of the qcspcchartts files.

JavascriptExamples/ – constains the JavaScript programming examples

AttributeControlCharts

FrequencyHistogramChart

HelloSPC

MultiDivCharts

MultiLimitCharts

NamedControlRules

RealTimeSPCChartUpdate

ParetoChart

ProbabilityChart

VariableControlCharts

VariableSampleSizeCharts

XbarRChart

TypescriptExamples/ – constains the TypeScript programming examples

AttributeControlCharts

FrequencyHistogramChart

MultiDivCharts

MultiDivCharts

MultiLimitCharts

NamedControlRules

RealTimeSPCChartUpdate

ParetoChart

ProbabilityChart

VariableControlCharts

VariableSampleSizeCharts

XbarRChart

Tutorials

Chapter 11 is a tutorial that describes how to define a simple SPC chart, using both JavaScript and TypeScript, and deploy it to a web page.

Customer Support

Use our forums at http://www.quinn-curtis.com/ForumFrame.htm for customer support. Please, do not post questions on the forum unless you are familiar with this manual and have run the examples programs provided. We try to answer most questions by referring to the manual, or to existing example programs. We will always attempt to answer any question that you may post, but be prepared that we may ask you to create, and send to us, a simple example program. The program should reproduce the problem with no, or minimal interaction, from the user. You should strip out of any code not directly associated with reproducing the problem. You can use either your own example or a modified version of one of our own examples.

Chapter Summary

The remaining chapters of this book discuss the **SPC Control Chart Tools for JavaScript/TypeScript** package.

Chapter 2 - Standard SPC Control Charts - presents a summary of the standard SPC control charts that can be created using the software.

Chapter 3 – Class architecture of QCSPCChartTS library –describes the class hierarchy of the QCSPCChartTS library.

Chapter 4 – Notes on JavaScript / TypeScript programming – describes differences between JavaScript vs TypeScript programming.

Chapter 5 - SPC Control Data and Alarm Classes - describes the classes that hold SPC control chart data and control limit alarms.

Chapter 6 – SPC Variable Control Chart - describes how the SPCBatchVariableControlChart classes create common variable control charts: X-Bar R, Median and Range, X-Bar Sigma and X-R charts**.**

Chapter 7 – SPC Attribute Control Charts - describes how the SPCBatchAttributeControlChart classes create common attribute control charts: p-, np-, c- and u-charts.

Chapter 8 – Named and Custom Control Rule Sets - describes how to implement the named control rules (WECO, Nelson, AIAG, Juran, Hughes, Gitlow, Westgard, and Duncan) control rules. It also describes how to implement custom rules sets, and how to define your own rules using our standardized templates.

Chapter 9 – Frequency Histogram and Pareto Diagram Charts – describes how the FrequencyHistogram, ProbabilityChart and ParetoChart classes create ancillary SPC charts.

Chapter 10 – Regionalization for non-USA English Markets – custom strings for the US-English, and other markets.

Chapter 11 – Using SPC Control Chart Tools for JavaScript/TypeScript to Web Applications - a tutorial that describes how to create web applications using both JavaScript and TypeScript.

Chapter 2 - Standard SPC Control Charts

There are many different types SPC control charts. Normally they fall into one of two major classifications: *Variable Control Charts*, and *Attribute Control Charts*. Within each classification, there are many sub variants. Often times the same SPC chart type has two or even three different names, depending on the software package and/or the industry the chart is used in. We have provided templates for the following SPC control charts:

Variable Control Charts

- Fixed sample size subgroup control charts
  - X-Bar R – (Mean and Range Chart)
  - X-Bar Sigma (Mean and Sigma Chart)
  - Median and Range (Median and Range Chart)
  - X-R (I-R, Individual Range Chart)
  - EWMA (Exponentially Weighted Moving Average Chart)
  - MA (Moving Average Chart)
  - MAMR (Moving Average / Moving Range Chart)
  - MAMS (Moving Average / Moving Sigma Chart)
  - Levey-Jennings with Westgard Rules
- Variable sample size subgroup control charts
  - X-Bar Sigma (Mean and Sigma Chart)

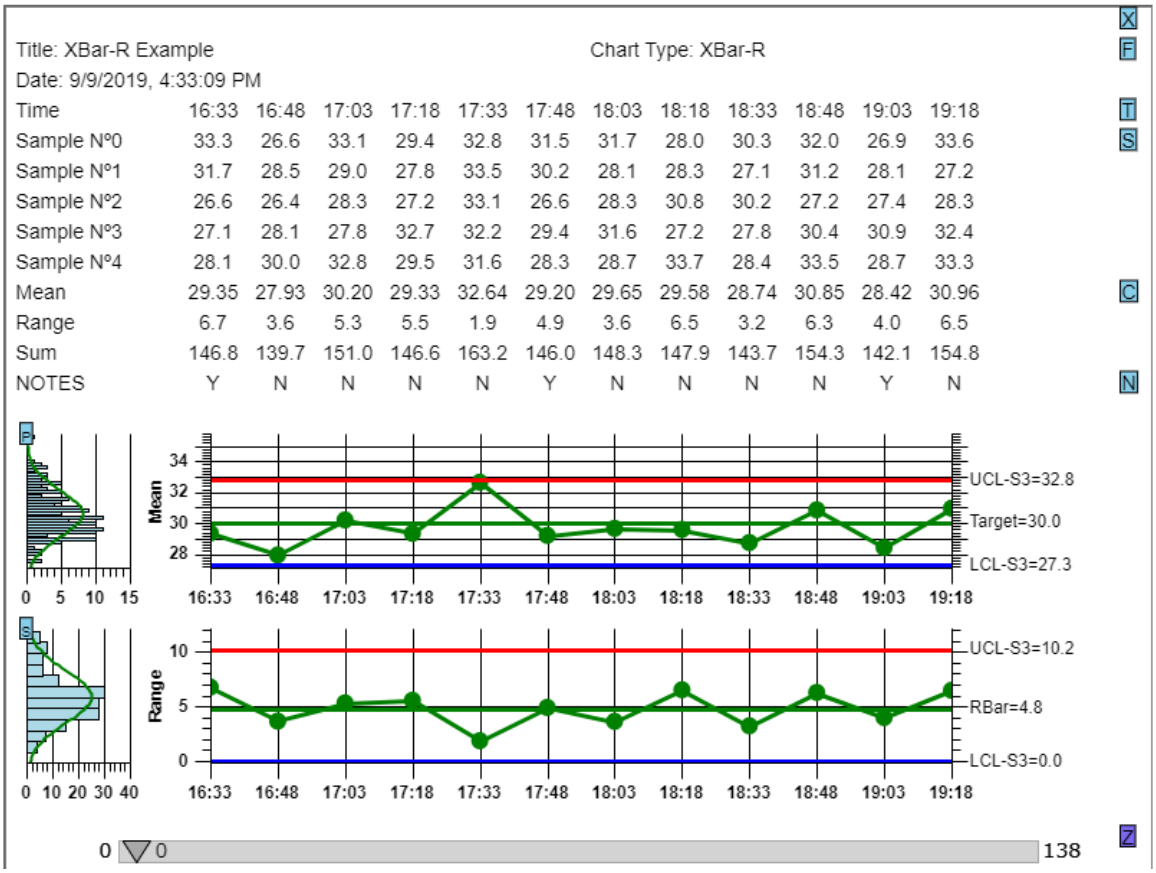
Attribute Control Charts

- Fixed sample size subgroup control charts
  - p-Chart (Fraction or Percent of Defective Parts)
  - np-Chart (Number of Defective Parts)
  - c-Chart (Number of Defects)
  - u-Chart (Number of Defects per Unit)
  - DPMO-chart (Number Defects per Million)
- Variable sample size subgroup control charts
  - p-Chart (Fraction or Percent of Defective Parts)
  - u-Chart (Number of Defects per Unit)

Frequency Histogram Chart
Pareto Chart
Probability Chart

Variable Control Charts

Variable Control Charts are for use with sampled quality data that can be assigned a specific numeric value, other than just 0 or 1. This might include, but is not limited to, the measurement of a critical dimension (height, length, width, radius, etc.), the weight a specific component, or the measurement of an important voltage. Common types of *Variable Control Charts* include X-Bar R (Mean and Range), X-Bar Sigma, Median and Range, X-R (Individual Range), EWMA, MA, MAMR (Moving Average/Moving Range), MAMS (Moving Average/Moving Sigma), Levey-Jennings and CuSum charts.



Variable Control Chart (X-Bar R) with header information and time stamps on the x-axis

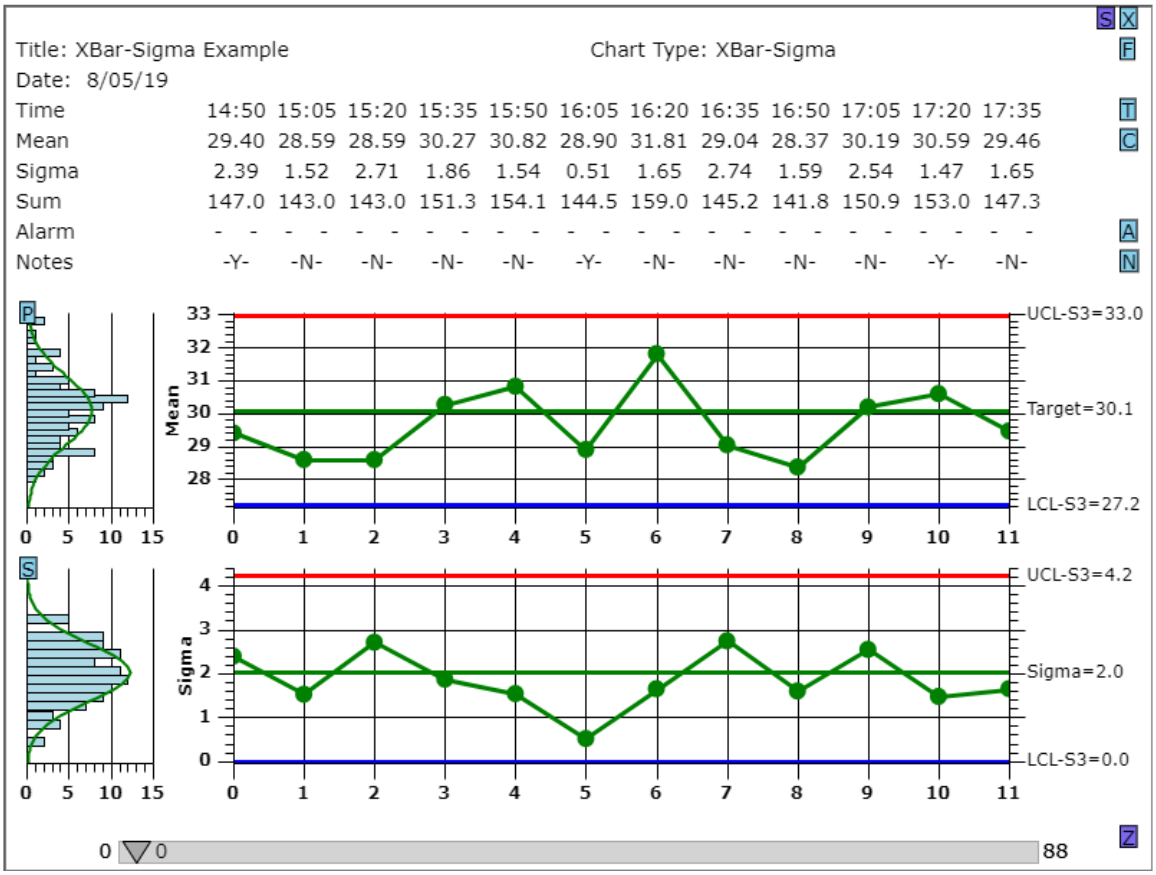
X-Bar R Chart – Also known as the Mean (or Average) and Range Chart

The X-Bar R chart monitors the trend of a critical process variable over time using a statistical sampling method that results in a subgroup of values at each subgroup interval. The X-Bar part of the chart plots the mean of each sample subgroup and the Range part

of the chart monitors the difference between the minimum and maximum value in the subgroup.

X-Bar Sigma Chart

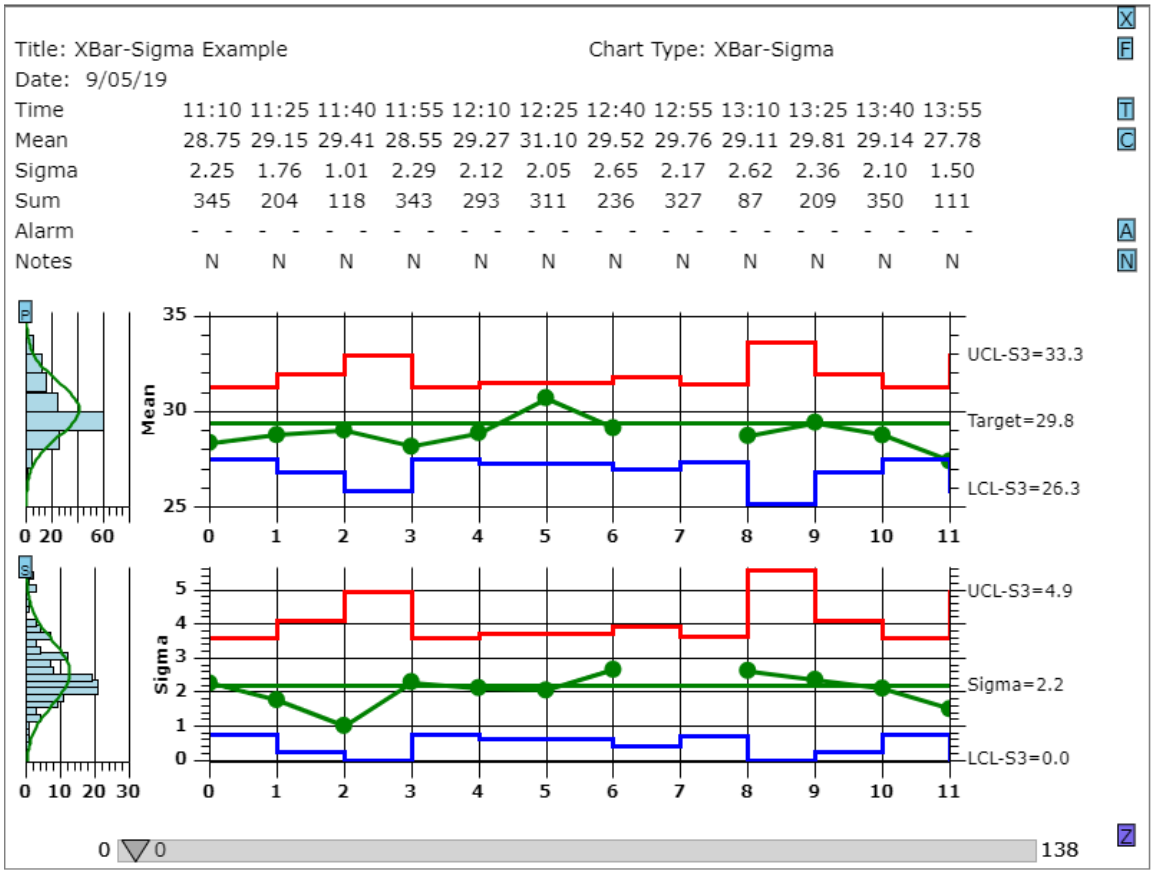
Very similar to the X-Bar R Chart, the X-Bar Sigma chart replaces the Range plot with a Sigma plot based on the standard deviation of the measured values within each subgroup. This is a more accurate way of establishing control limits if the sample size of the subgroup is moderately large (> 10). Though computationally more complicated, the use of a computer makes this a non-issue.



Fixed sample size X-Bar Sigma Control chart with header information and a batch number x-axis

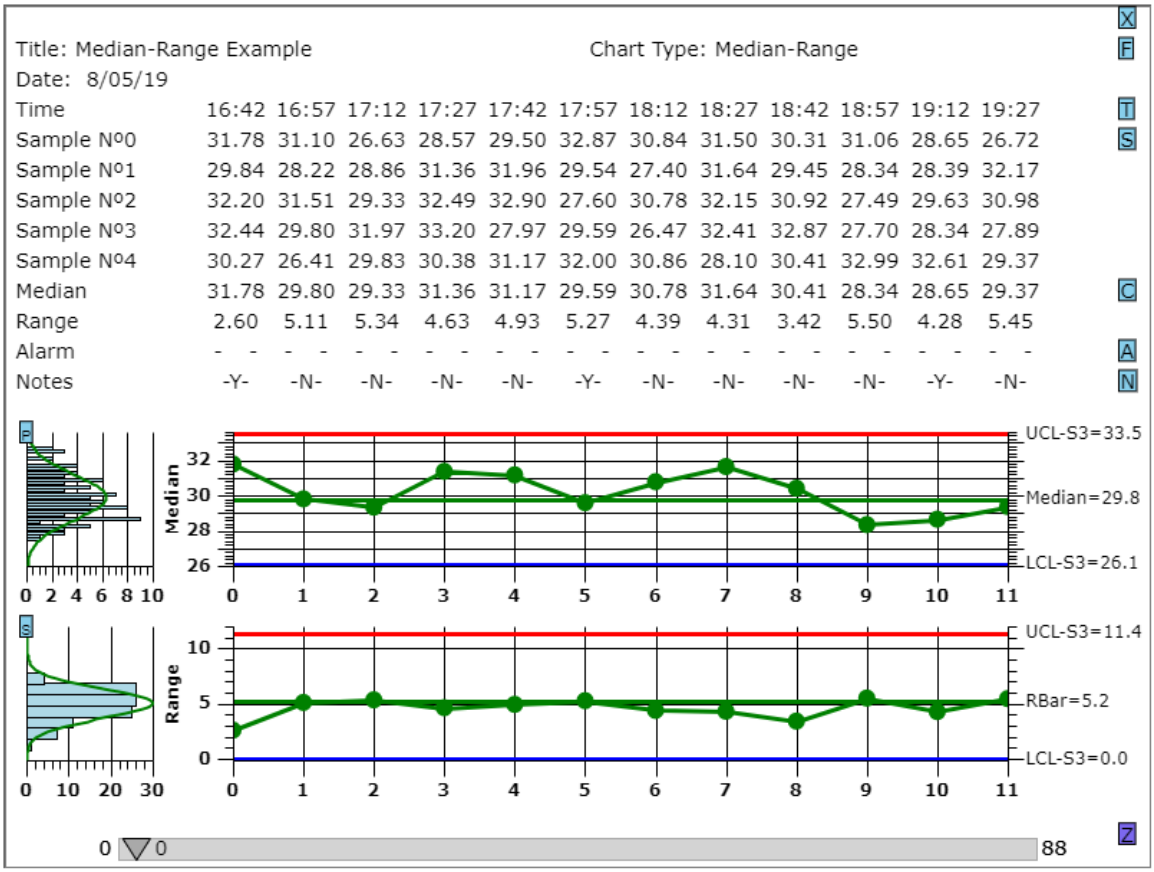
The X-Bar Sigma chart can also be used if the sample subgroup size varies from sampling interval to sampling interval. In this case, the control chart high and low limits vary from sample interval to sample interval, depending on the number of samples in the associated sample subgroup. A low number of samples in the sample subgroup make the band between the high and low limits wider than if a higher number of samples are

available. The X-Bar Sigma chart is the only variable control chart that can be used with a variable sample size.



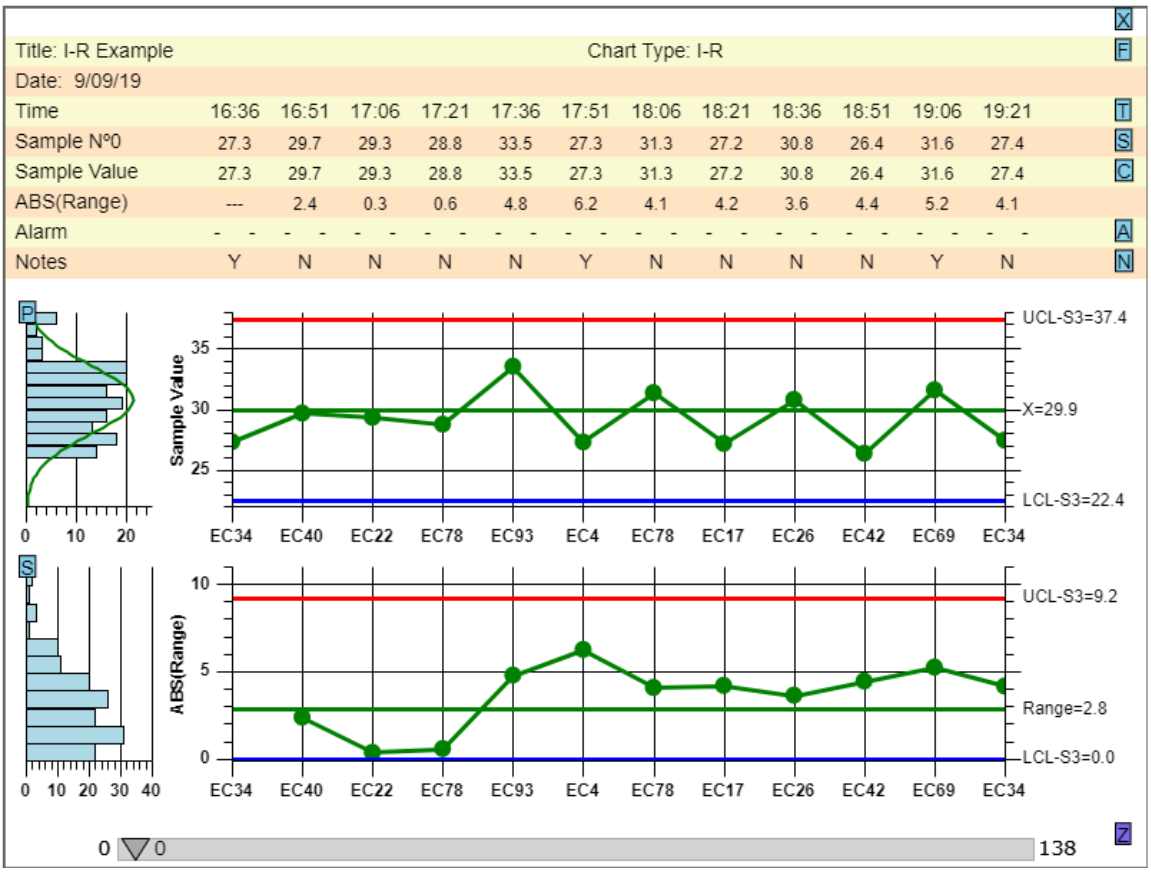
X-Bar Sigma Chart with variable sample size

Median Range – Also known as the Median and Range Chart



Median-Range Chart with batch number x-axis

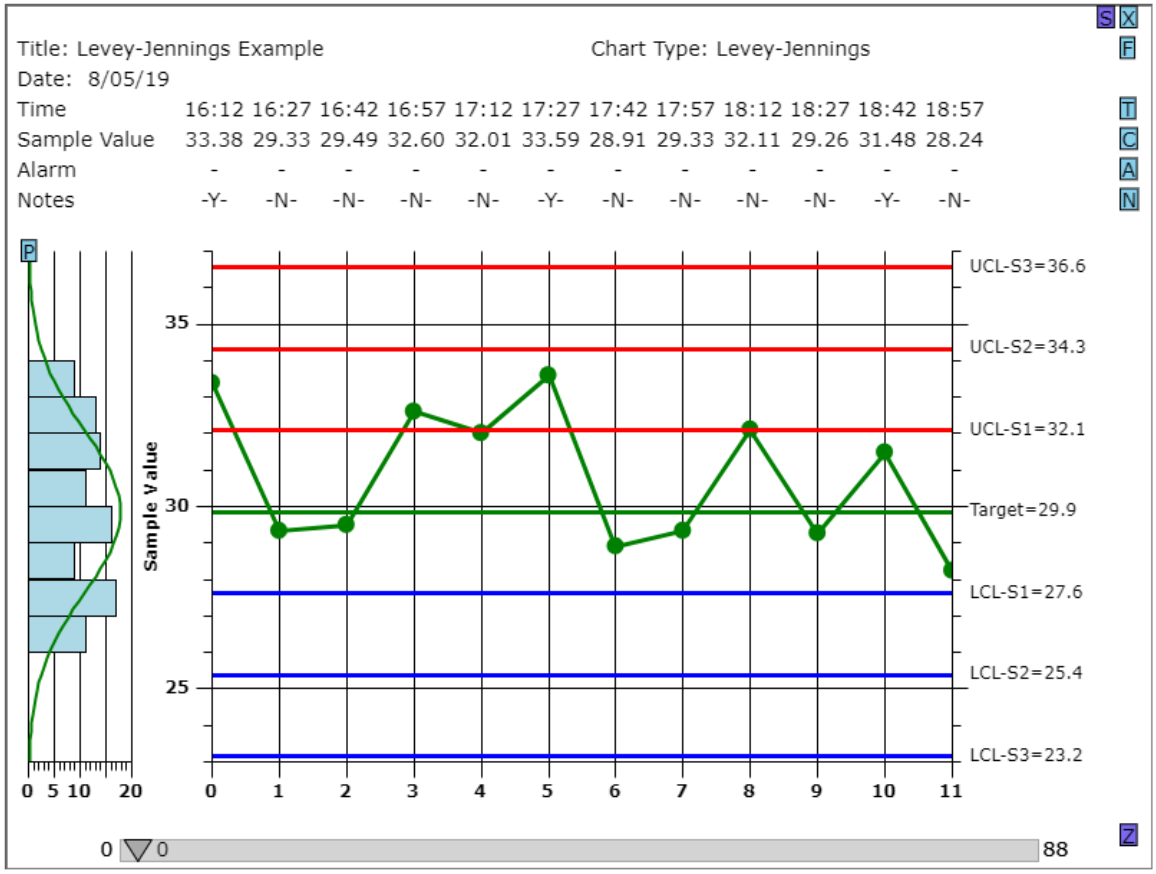
Very similar to the X-Bar R Chart, Median Range chart replaces the Mean plot with a Median plot representing the median of the measured values within each subgroup. The Median Range chart requires that the process be well behaved, where the variation in measured variables are (1) known to be distributed normally, (2) are not very often disturbed by assignable causes, and (3) are easily adjusted.



Typical Individual Range Chart (X-R) with header information and user-defined strings on the x-axis

Individual Range Chart – Also known as the X-R Chart

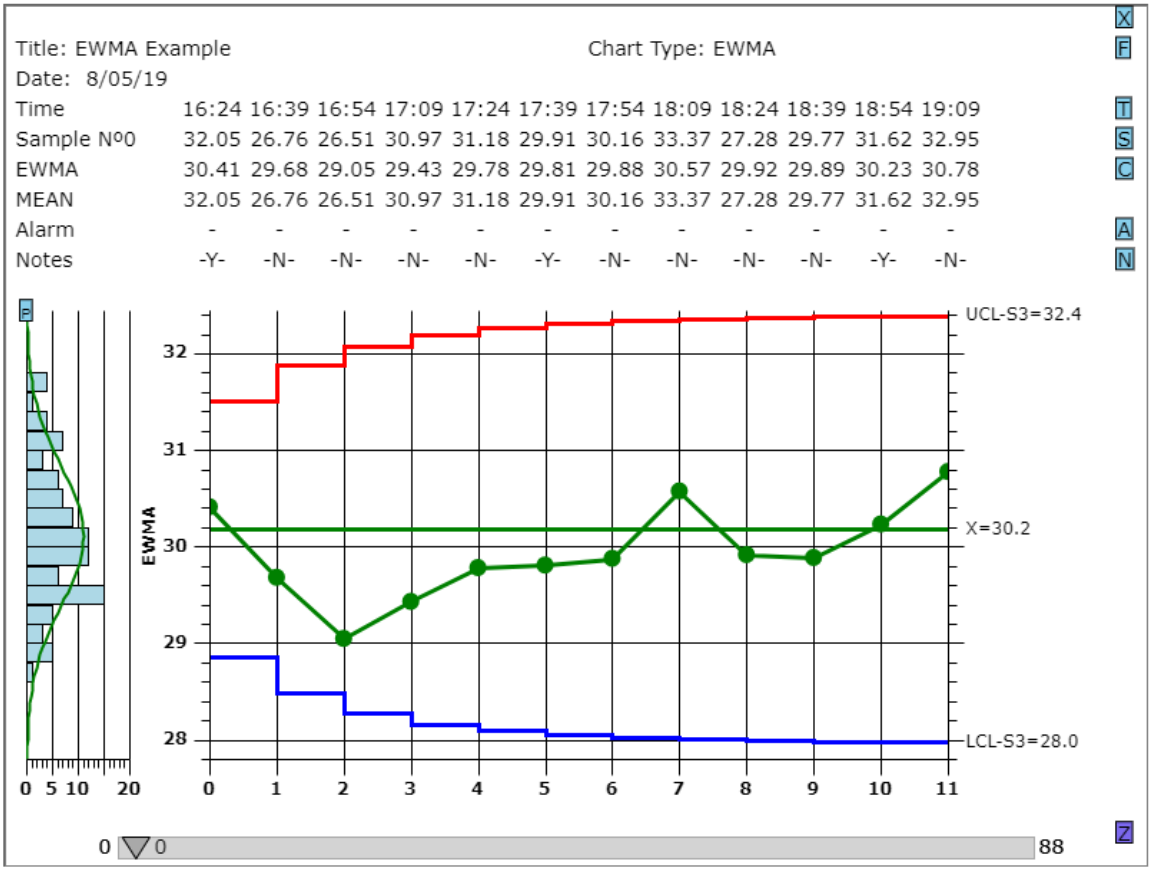
The Individual Range Chart is used when the sample size for a subgroup is 1. This happens frequently when the inspection and collection of data for quality control purposes is automated and 100% of the units manufactured are analyzed. It also happens when the production rate is low and it is inconvenient to have sample sizes other than 1. The X part of the control chart plots the actual sampled value (not a mean or median) for each unit and the R part of the control chart plots a moving range, calculated using the current value of sampled value minus the previous value.



Typical Levey-Jennings Chart using Batch Sampling

Levey-Jennings Chart

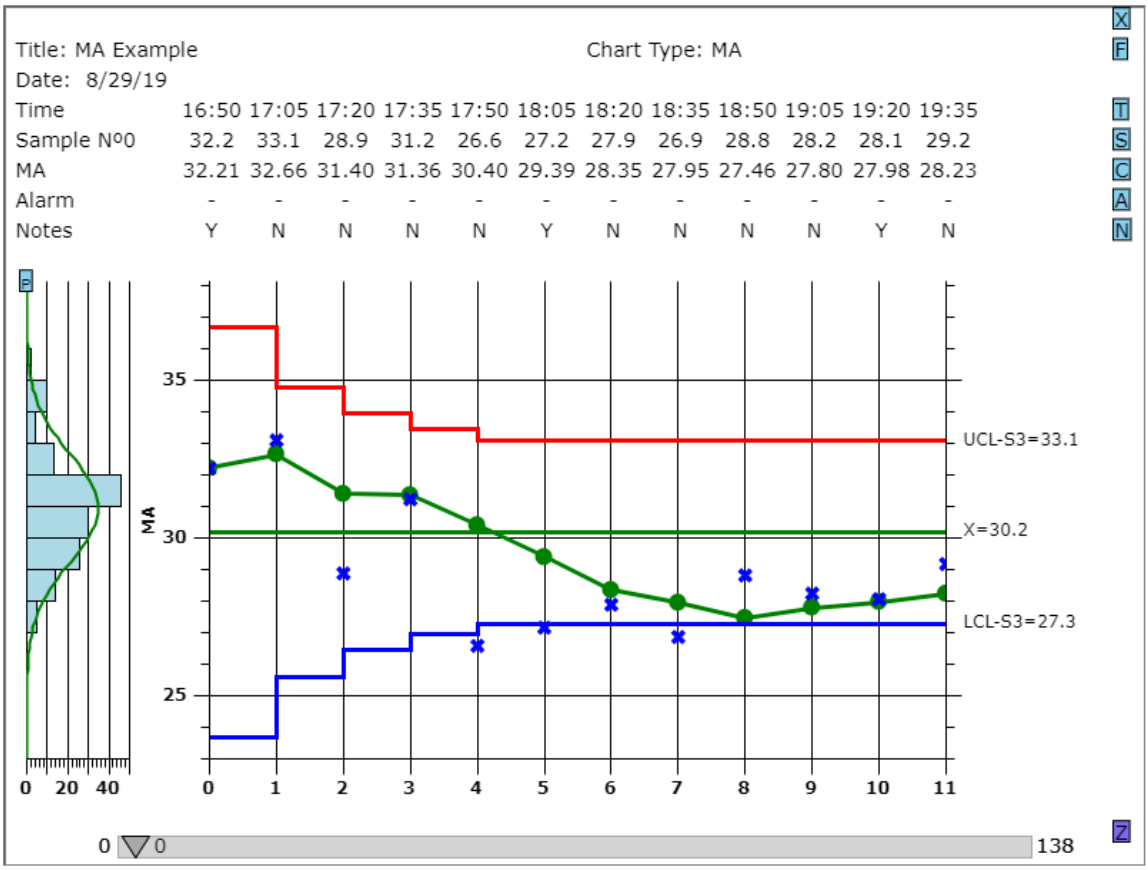
The Levey-Jennings chart is used almost exclusively in laboratory settings. It uses a chart very similar to the Individual Range chart above, the major difference being that it only uses the Primary individual data point graph of the chart and does not include the Secondary range graph. Also, the Levey-Jennings chart uses the Westgard rules which utilizes tests involving 1-, 2- and 3- sigma control limits. The control limit calculations depart from all of the other SPC Chart types in that the target value (mean) and control limit (sigma) calculations use the overall mean and standard deviation values from the entire, charted, sample population. See the links https://en.wikipedia.org/wiki/Laboratory_quality_control and https://www.westgard.com/lesson12.htm for more information about the underlying principles of the Levey-Jennings chart.



Typical EWMA Chart

EWMA Chart – Exponentially Weighted Moving Average

The EWMA chart is an alternative to the preceding Shewhart type control charts (X-Bar R and I-R charts in particular) and is most useful for detecting small shifts in the process mean. It uses a weighted moving average of previous values to “smooth” the incoming data, minimizing the affect of random noise on the process. It weights the current and most recent values more heavily than older values, allowing the control line to react faster than a simple MA (Moving Average) plot to changes in the process. Like the Shewhart charts, if the EWMA value exceeds the calculated control limits, the process is considered out of control. While it is usually used where the process uses 100% inspection and the sample subgroup size is 1 (same is the I-R chart), it can also be used when sample subgroup sizes are greater than one.

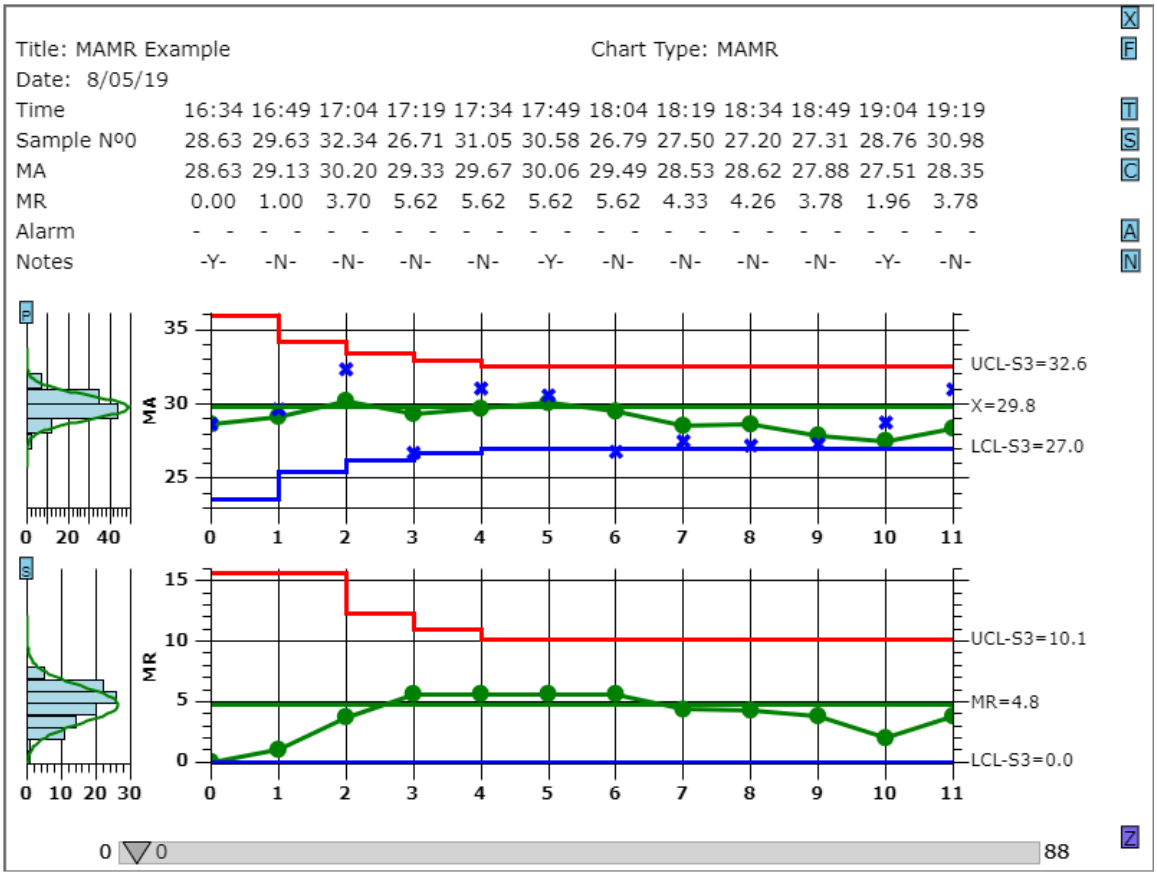


MA (Moving Average) Chart with Sample Values Plotted

MA Chart – Moving Average

The MA chart is another alternative to the preceding Shewhart type control charts (X-Bar R and I-R charts in particular) and is most useful for detecting small shifts in the process mean. It uses a moving average, where the previous (N-1) sample values of the process variable are averaged together along with the process value to produce the current chart value. This helps to "smooth" the incoming data, minimizing the affect of random noise on the process. Unlike the EWMA chart, the MA chart weights the current and previous (N-1) values equally in the average. While the MA chart can often detect small changes in the process mean faster than the Shewhart chart types, it is generally considered inferior to the EWMA chart. Like the Shewhart charts, if the MA value exceeds the calculated control limits, the process is considered out of control.

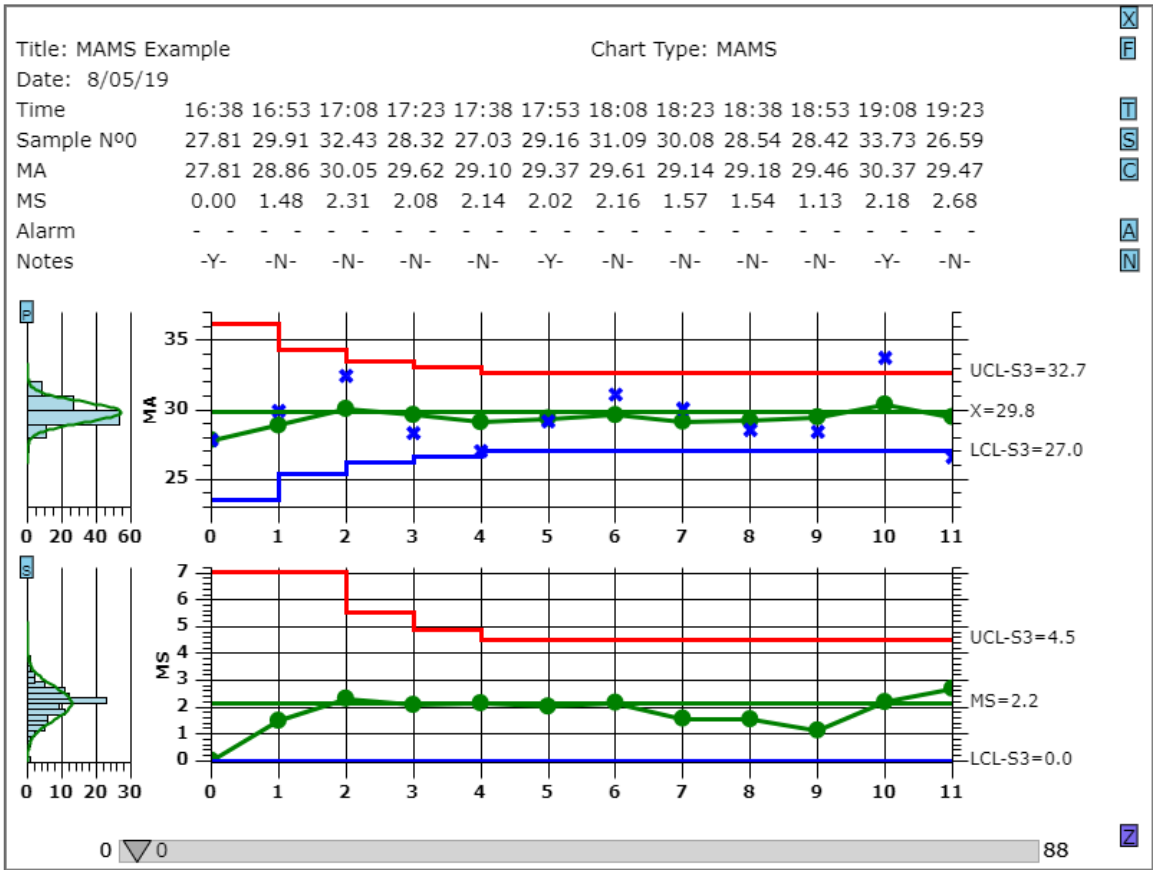
MAMR Chart – Moving Average / Moving Range



MAMR using both raw data and smoothed values

The MAMR chart combines our Moving Average chart with a Moving Range chart. The Moving Average chart is primary (topmost) chart, and the Moving Range chart is the secondary (bottom) chart. It uses a single sample/subgroup, same as our standard [Individual-Range], [Moving Average], [EWMA], and [Moving Average] charts. When calculating the Moving Range, it windows the same data values used in the Moving Average calculation. Note that the limits are variable (wider) at the beginning, taking into account the fewer samples in the start up of any type of SPC chart which uses a sliding window in the calculation of moving average and moving range statistics.

MAMS Chart – Moving Average / Moving Sigma



MAMS (Moving Average/Moving Sigma) Chart with Sample Values Plotted

The MAMS chart combines our Moving Average chart with a Moving Sigma chart. The Moving Average chart is primary (topmost) chart, and the Moving Sigma chart is the secondary (bottom) chart. It uses a single sample/subgroup, same as our standard [Individual-Range], [Moving Average], [EWMA], and [Moving Average] charts. When calculating the Moving Sigma, it windows the same data values used in the Moving Average calculation. Note that the limits are variable (wider) at the beginning, taking into account the fewer samples in the start up of any type of SPC chart which uses a sliding window in the calculation of moving average and moving sigma statistics.

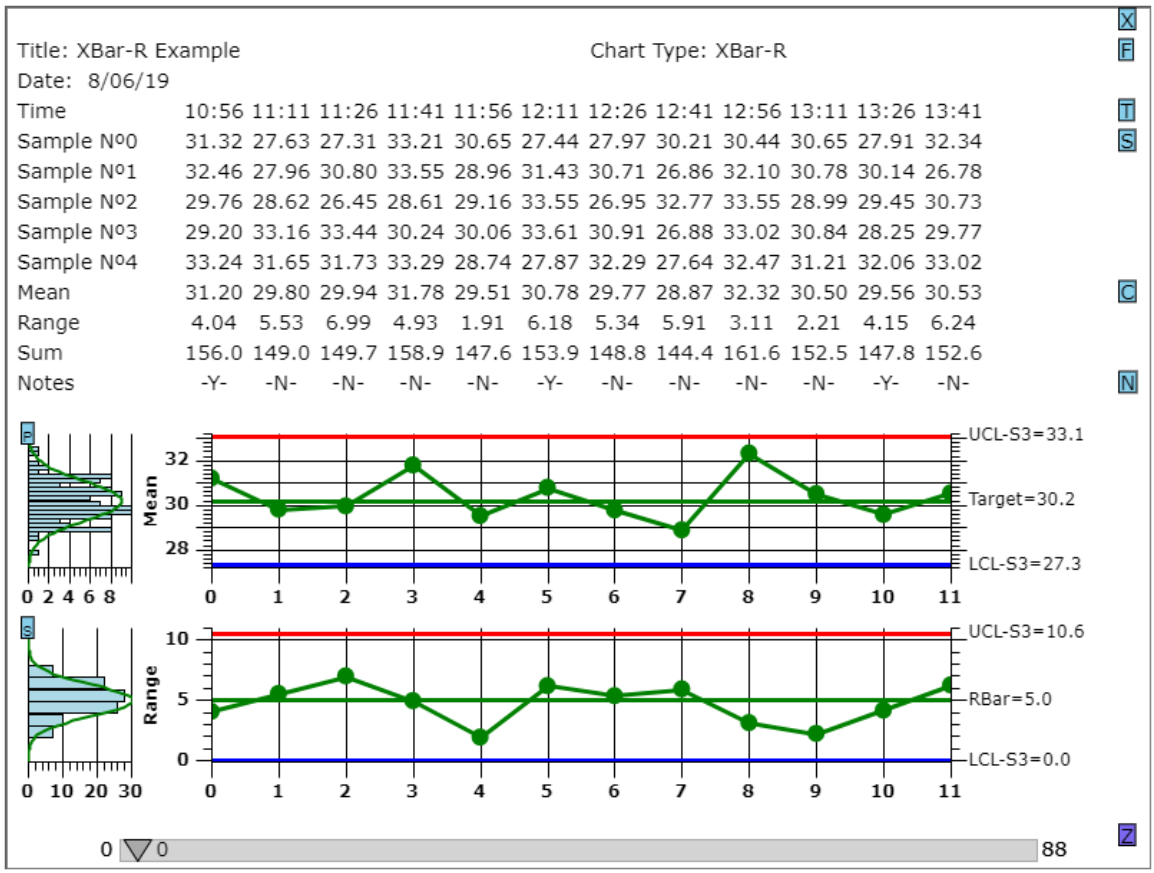
Measured Data and Calculated Value Tables

Standard worksheets used to gather and plot SPC data consist of three main parts.

- The first part is the header section, identifying the title of the chart, the monitored process, the machine operator, part number and other important information specific to the chart.

- The second part is the measurement data recording and calculation section, organized as a table, recording the sampled and calculated data in a neat, readable fashion.
- The third part, the actual SPC chart, plots the calculated SPC values for the sample group

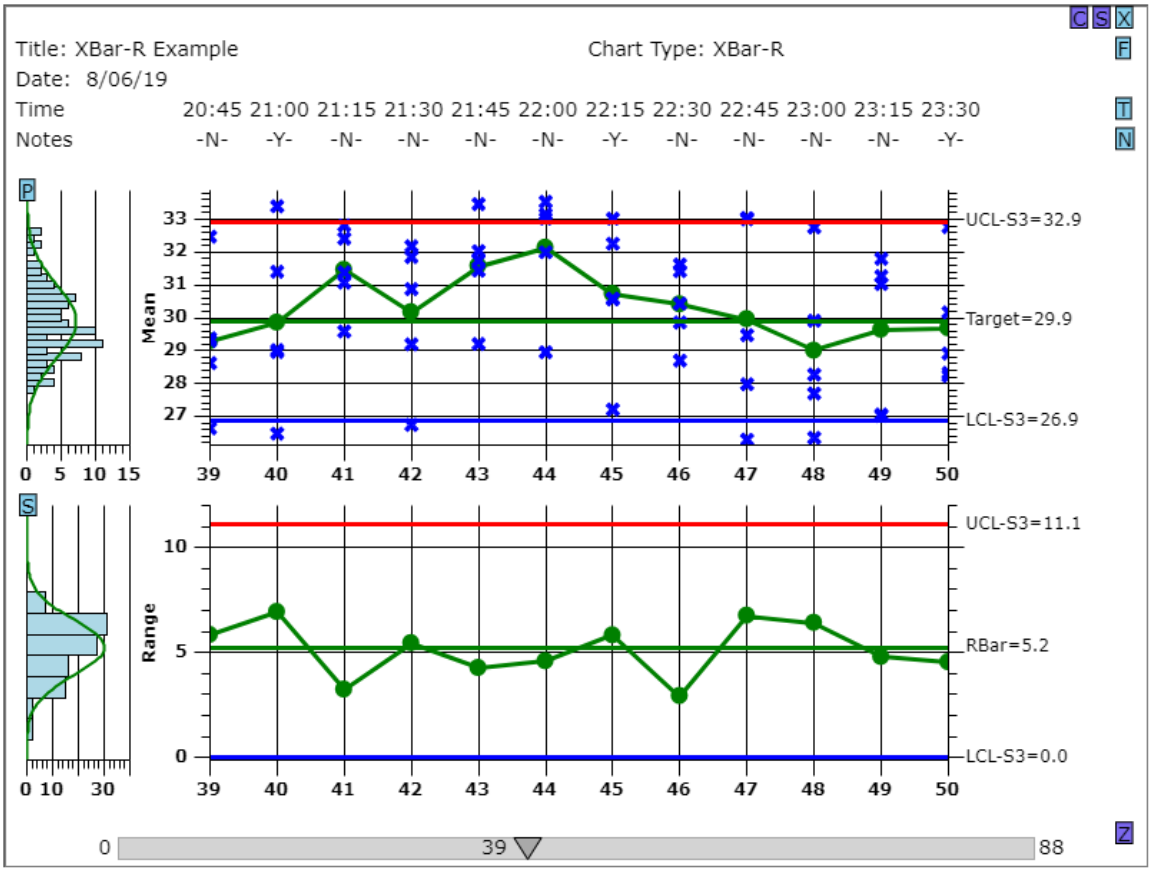
The *Variable Control Chart* templates that we have created have options that enable the programmer to customize and automatically include header information along with a table of the measurement and calculated data, in the SPC chart. Enable the scrollbar option and you can display the tabular measurement data and SPC plots for a window of 8-20 subgroups, from a much larger collection of measurement data represented hundreds or even thousands of subgroups, and use the scrollbar to move through the data, similar to paging through a spreadsheet.



Scrollable Bar-R Chart with frequency histograms, header, measurement and calculated value information

Scatter Plots of the Actual Sampled Data

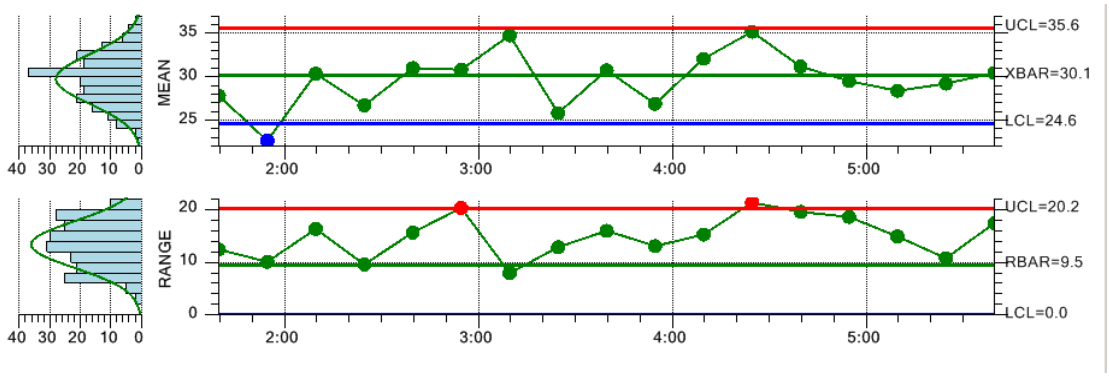
In some cases it useful to plot the actual values of a sample subgroup along with the sample subgroup mean or median. Plot these samples in the SPC chart using additional scatter plots.



Scrollable XBar-R Chart with Scatter Plot of Actual Sampled Data

Alarm Notification

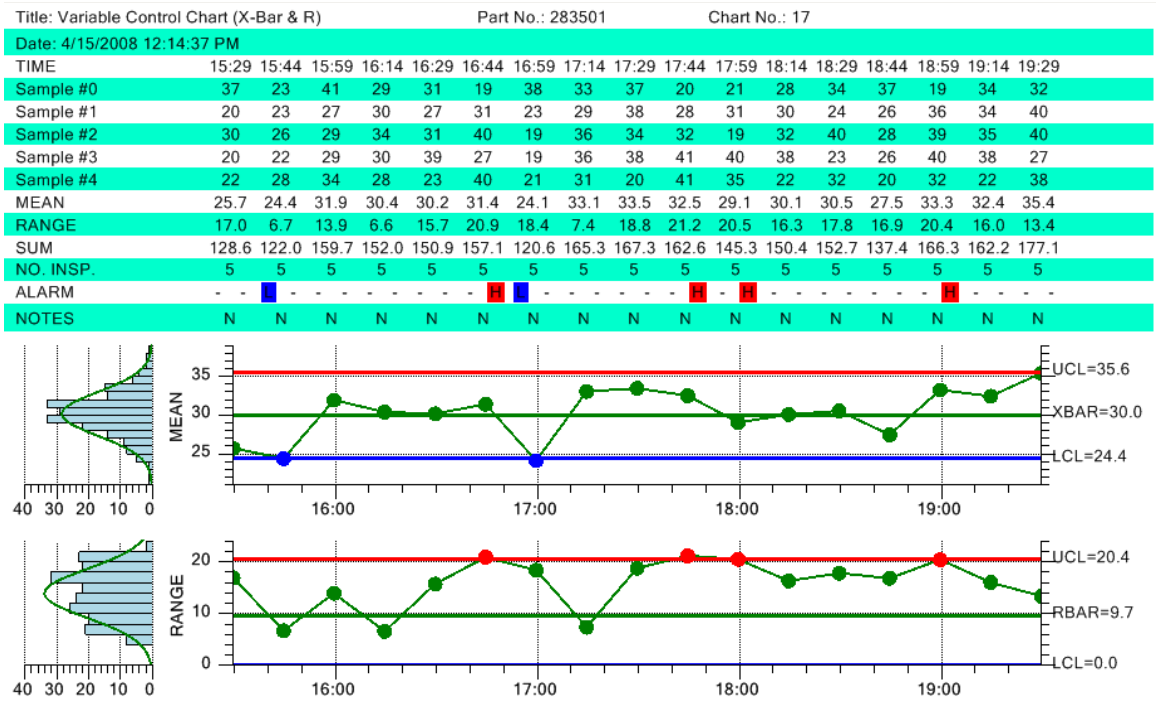
Typically, when a process value exceeds a control limit, an alarm condition exists. In order to make sure that the program user identifies an alarm you can emphasize the alarm in several different ways. You can trap the alarm condition using an event delegate, log the alarm to the notes log, highlight the data point symbol in the chart where the alarm occurs, display an alarm status line in the data table, or highlight the entire column of the sample interval where the alarm occurs.



Change the color of a data point that falls outside of alarm limits.

Title: Variable Control Chart (X-Bar & R)						Part No.: 283501						Chart No.: 17					
Date: 4/15/2008 12:09:38 PM																	
TIME	1:39	1:54	2:09	2:24	2:39	2:54	3:09	3:24	3:39	3:54	4:09	4:24	4:39	4:54	5:09	5:24	5:39
Sample #0	33	26	23	31	37	29	38	30	31	25	32	40	34	34	30	32	23
Sample #1	21	19	24	32	34	33	30	19	24	25	37	41	41	30	28	26	32
Sample #2	33	19	40	25	21	20	36	23	26	20	22	19	28	23	19	30	29
Sample #3	27	20	36	22	33	32	36	25	32	32	34	38	21	21	31	34	41
Sample #4	25	29	28	24	30	40	34	32	40	33	35	38	32	39	34	23	27
MEAN	27.8	22.6	30.4	26.7	31.0	30.8	34.8	25.8	30.8	26.9	32.1	35.2	31.2	29.5	28.4	29.2	30.4
RANGE	12.4	10.1	16.3	9.6	15.6	20.3	8.0	12.9	16.0	13.1	15.2	21.3	19.6	18.6	14.9	10.8	17.4
SUM	139.1	113.1	151.8	133.5	155.0	154.1	173.8	129.1	153.8	134.4	160.3	175.9	155.9	147.5	141.9	146.0	152.2
NO. INSP.	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
ALARM	- -	L -	- - - - -			- H	- - - - -			- - - - -		- H	- - - - -			- - - - -	
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Highlight the column of the sample interval where the alarm occurs

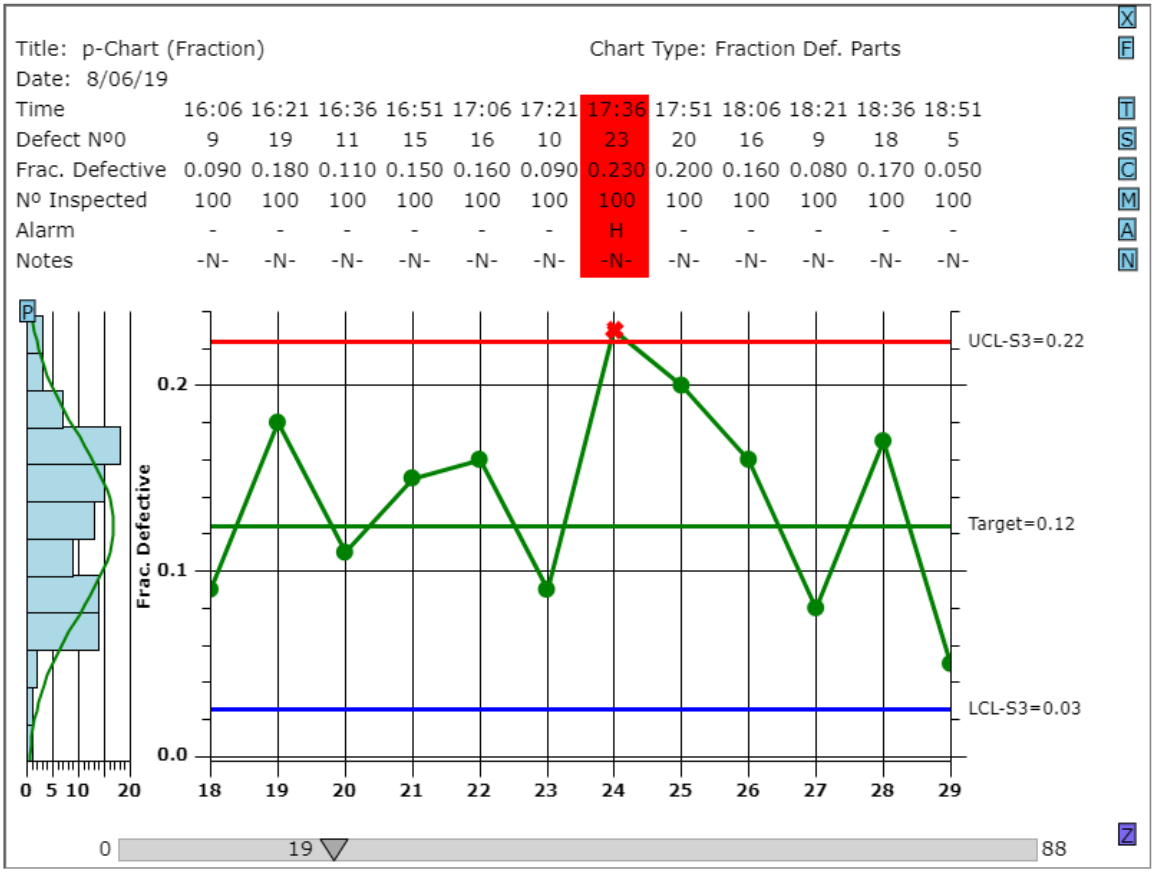


An alarm status line highlights an alarm condition, and lets you know when chart the (primary or secondary) the alarm occurs in.

These alarm highlight features apply to both variable control and attribute control charts.

Attribute Control Charts

Attribute Control Charts are a set of control charts specifically designed for tracking defects (also called non-conformities). These types of defects are binary in nature (yes/no), where a part has one or more defects, or it doesn't. Examples of defects are paint scratches, discolorations, breaks in the weave of a textile, dents, cuts, etc. Think of the last car that you bought. The defects in each sample group are counted and run through some statistical calculations. Depending on the type of *Attribute Control Chart*, the number of defective parts are tracked (p-chart and np-chart), or alternatively, the number of defects are tracked (u-chart, c-chart). The difference in terminology "number of defective parts" and "number of defects" is highly significant, since a single part not only can have multiple defect categories (scratch, color, dent, etc), it can also have multiple defects per category. A single part may have 0 – N defects. So keeping track of the number of defective parts is statistically different from keeping track of the number of defects. This affects the way the control limits for each chart are calculated.

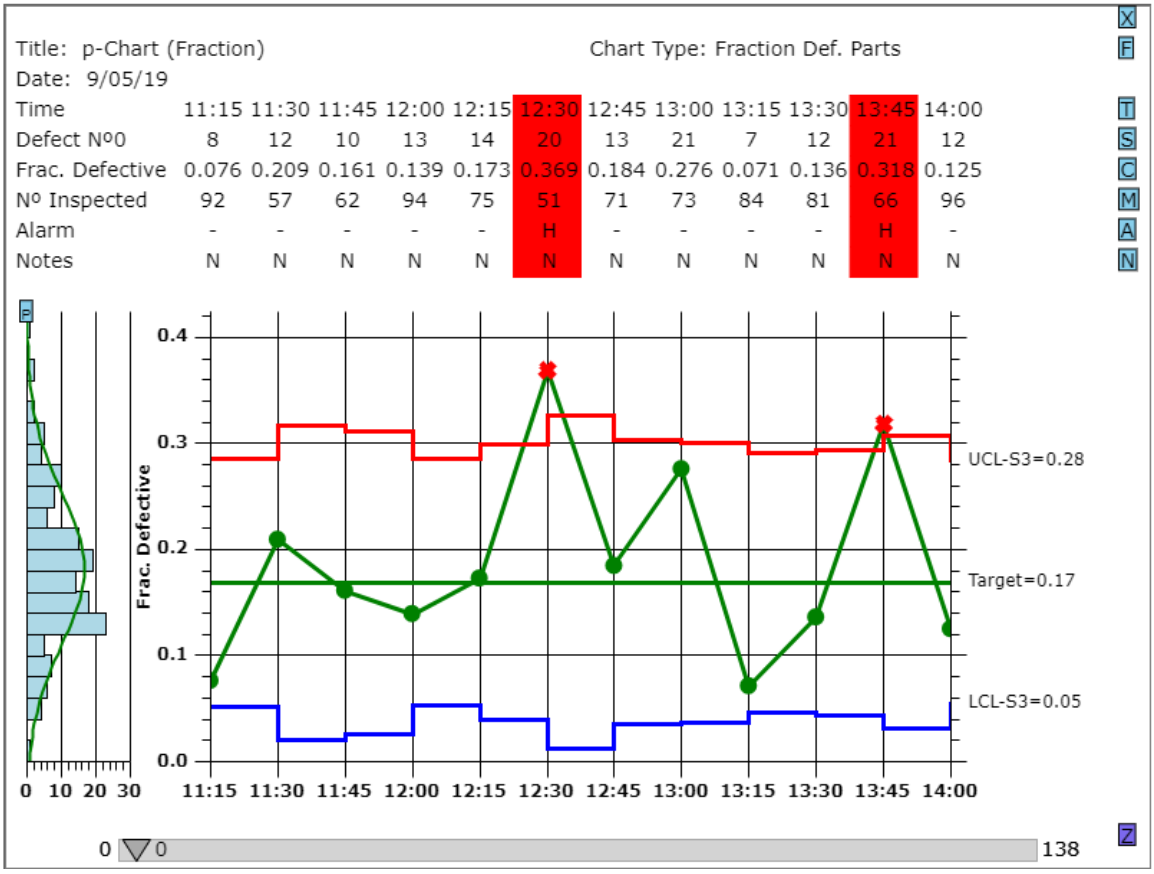


Typical Attribute Control Chart (p-Chart)

p-Chart - Also known as the Percent or Fraction Defective Parts Chart

For a sample subgroup, the number of defective parts is measured and plotted as either a percentage of the total subgroup sample size, or a fraction of the total subgroup sample size. Since the plotted value is a fraction or percent of the sample subgroup size, the size of the sample group can vary without rendering the chart useless.

The p-Chart chart can also be used if the sample subgroup size varies from sampling interval to sampling interval. In this case, the control chart high and low limits vary from sample interval to sample interval, depending on the number of samples in the associated sample subgroup. A low number of samples in the sample subgroup make the band between the high and low limits wider than if a higher number of samples are available. Both the *Fraction Defective Parts and Percent Defective Parts* control charts come in versions that support variable sample sized for a subgroup.

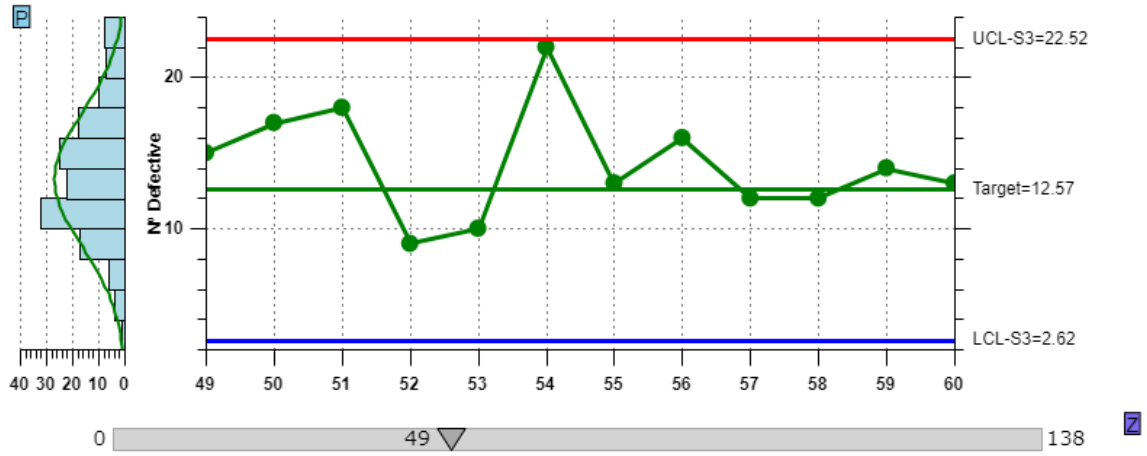


Fraction Defective Parts (p-Chart) with variable sample size

np-Chart – Also known as the Number Defective Parts Chart

For a sample subgroup, the number of defective parts is measured and plotted as a simple count. Statistically, in order to compare number of defective parts for one subgroup with the other subgroups, this type of chart requires that the subgroup sample size is fixed across all subgroups.

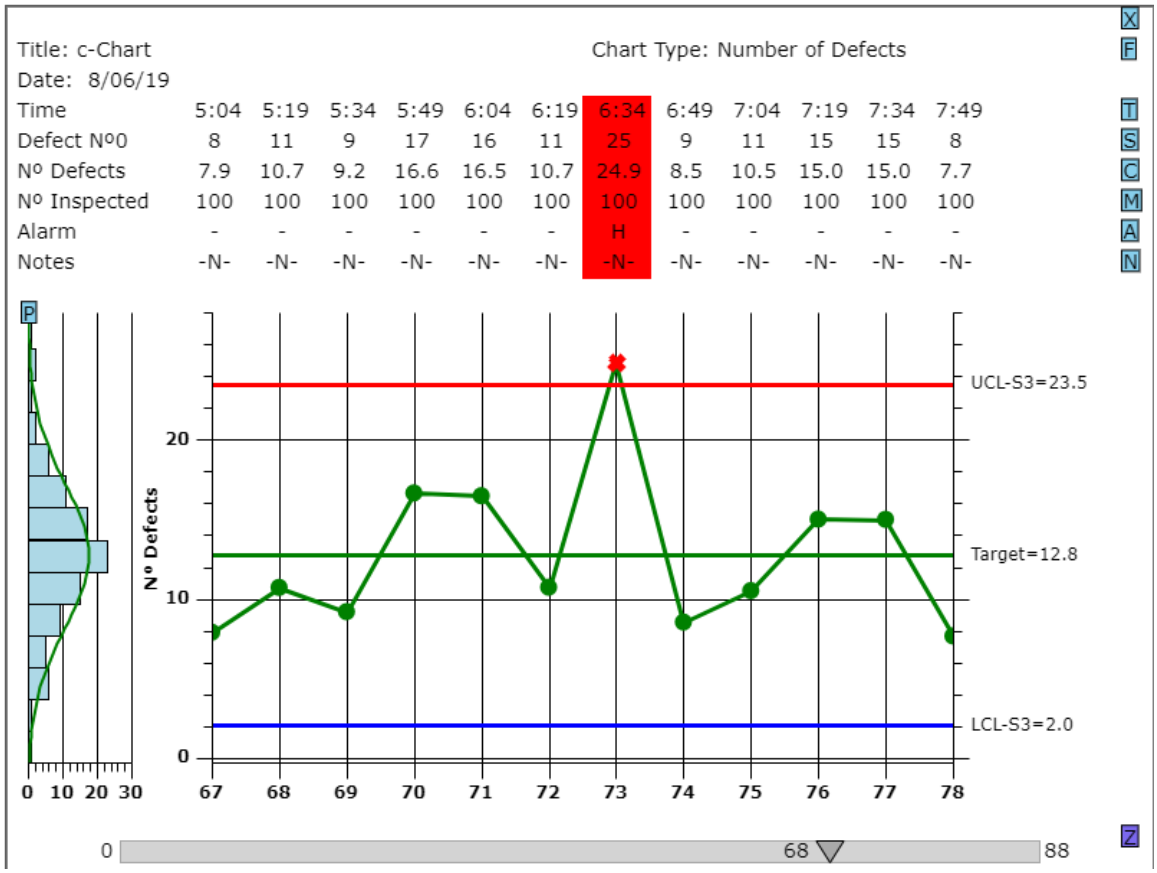
Title: np-Chart						Chart Type: Number Def. Parts						
Date: 10/08/19												
Time	1:31	1:46	2:01	2:16	2:31	2:46	3:01	3:16	3:31	3:46	4:01	4:16
Scratch	1	9	2	3	4	3	2	5	1	1	6	2
Burr	3	8	6	5	1	11	2	8	0	3	5	0
Dent	6	3	2	2	2	2	6	7	0	2	0	3
Seam	1	5	2	1	1	8	1	8	6	5	1	1
Other	5	1	6	1	3	10	2	6	5	1	5	7
Nº Defective	15.0	17.0	18.0	9.0	10.0	22.0	13.0	16.0	12.0	12.0	14.0	13.0
Nº Inspected	100	100	100	100	100	100	100	100	100	100	100	100
Alarm	-	-	-	-	-	-	-	-	-	-	-	-
Notes	N	N	N	N	N	N	N	N	N	N	N	N



Typical np-Chart

c-Chart - Also known as the Number of Defects or Number of Non-Conformities Chart

For a sample subgroup, the number of times a defect occurs is measured and plotted as a simple count. Statistically, in order to compare number of defects for one subgroup with the other subgroups, this type of chart requires that the subgroup sample size is fixed across all subgroups.

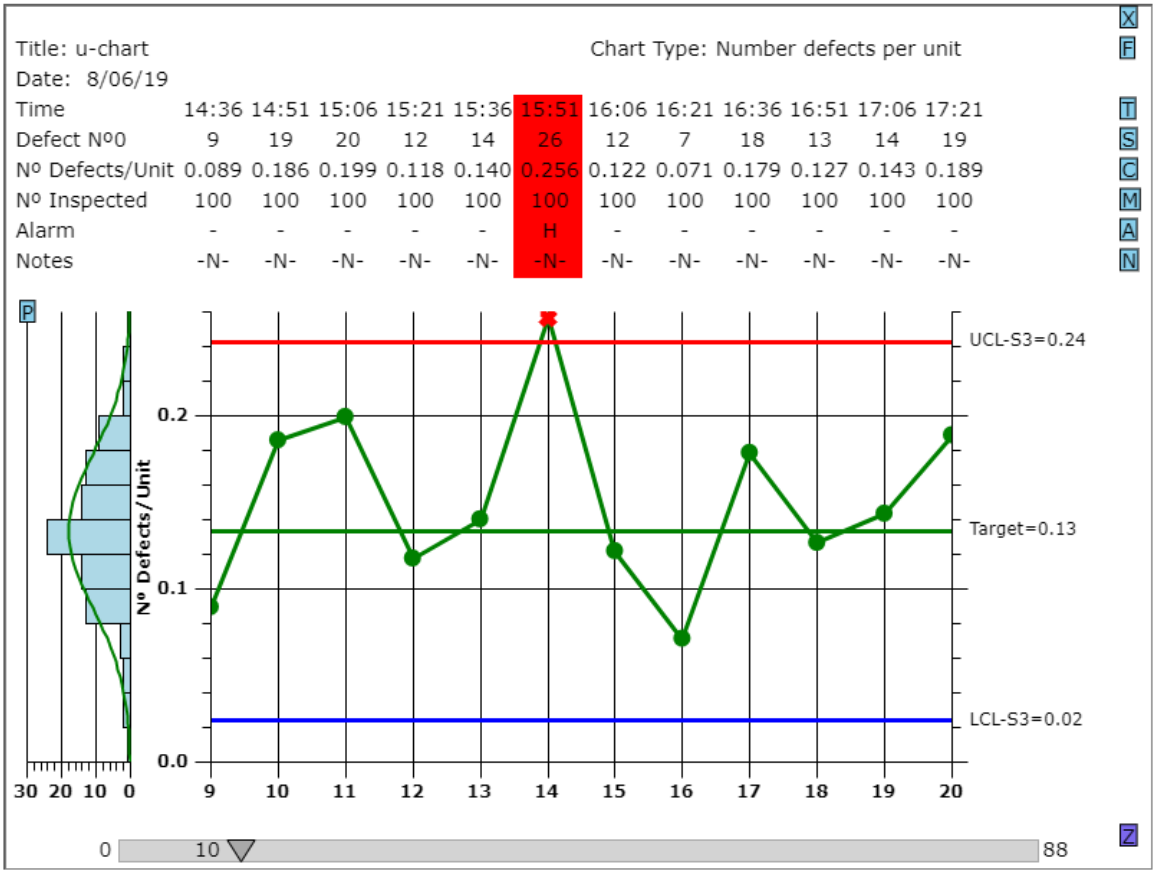


Typical Number of Defects (c)

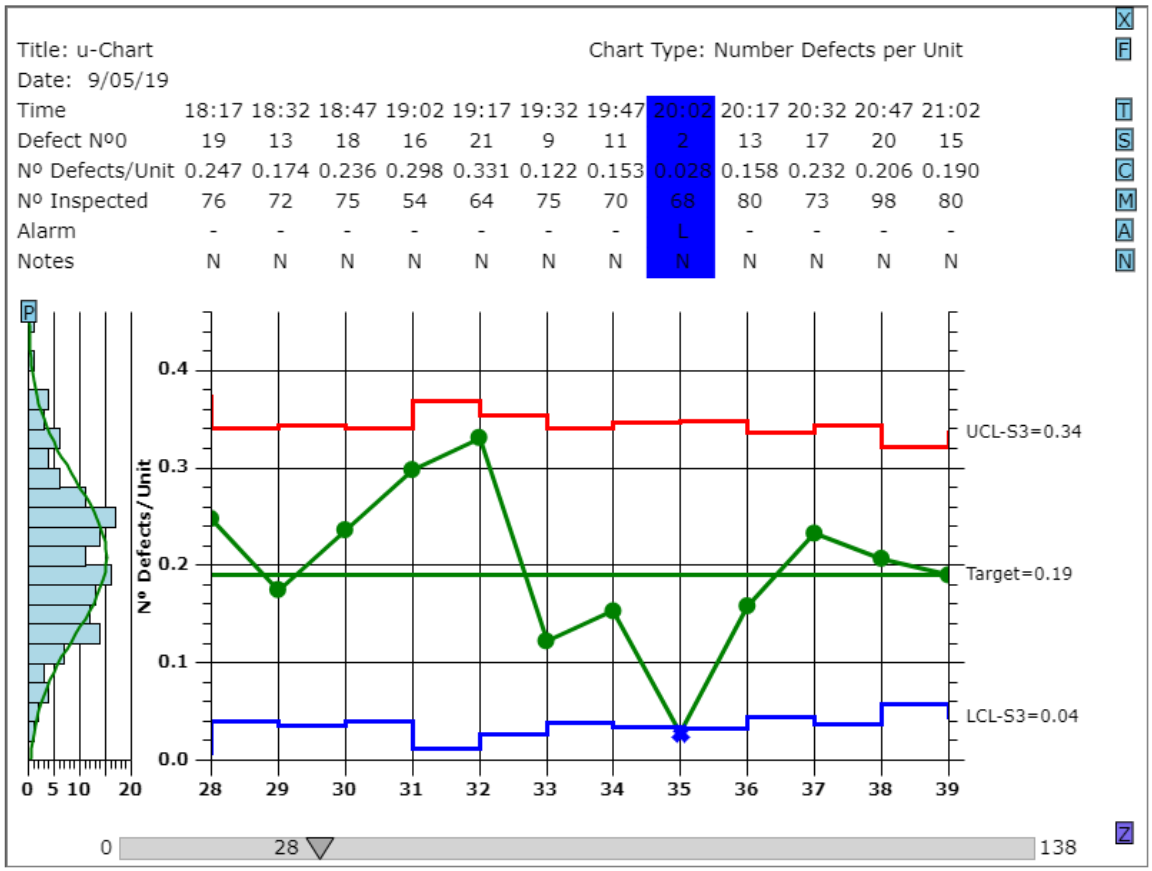
u-Chart – Also known as the Number of Defects per Unit or Number of Non-Conformities per Unit Chart

For a sample subgroup, the number of times a defect occurs is measured and plotted as either a percentage of the total subgroup sample size, or a fraction of the total subgroup sample size. Since the plotted value is a fraction or percent of the sample subgroup size, the size of the sample group can vary without rendering the chart useless.

The u-Chart chart can also be used if the sample subgroup size varies from sampling interval to sampling interval. In this case, the control chart high and low limits vary from sample interval to sample interval, depending on the number of samples in the associated sample subgroup. A low number of samples in the sample subgroup make the band between the high and low limits wider than if a higher number of samples are available.

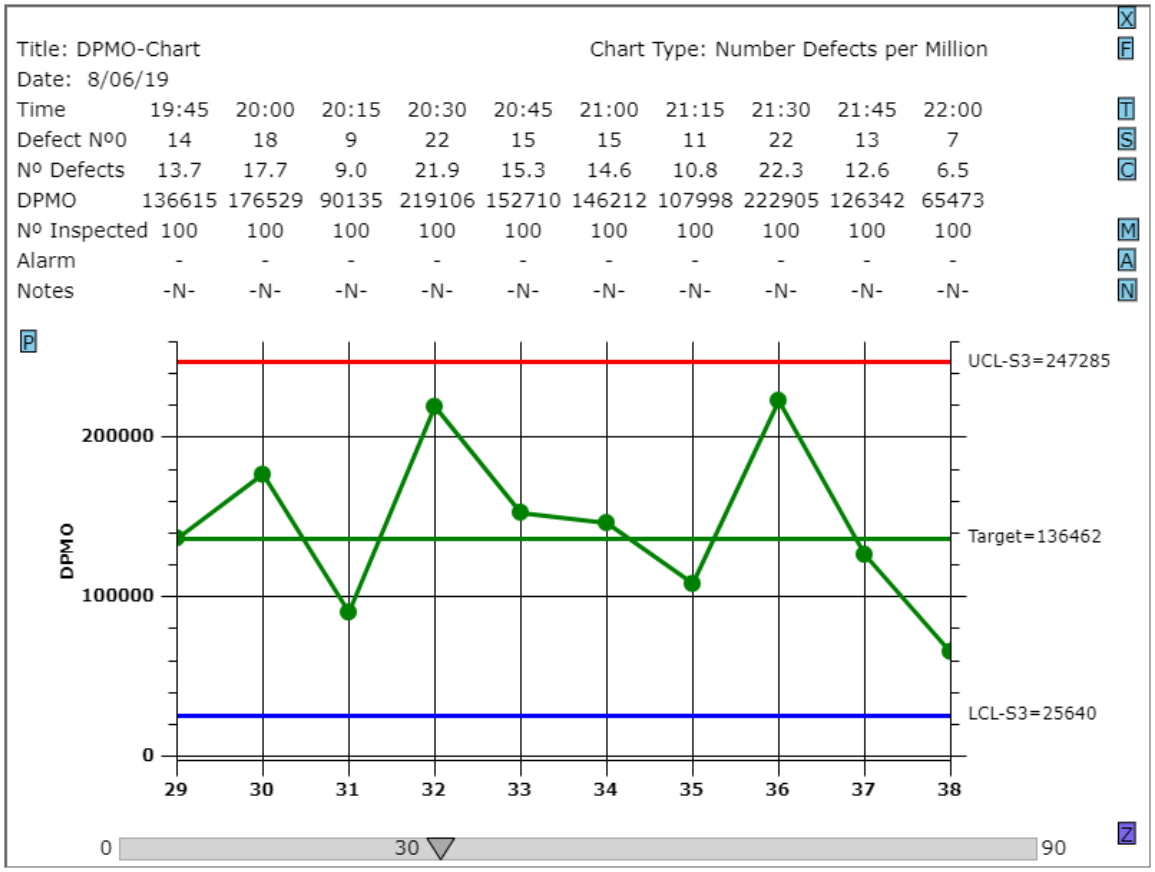


Number of Defects per Unit Chart (u-chart)



Number of Defects per Unit Chart with variable sample size (u-chart)

DPMO-Chart – Also known as Defects per Million Opportunities



DPMO Chart – Also known as the Number of Defects per Million Chart

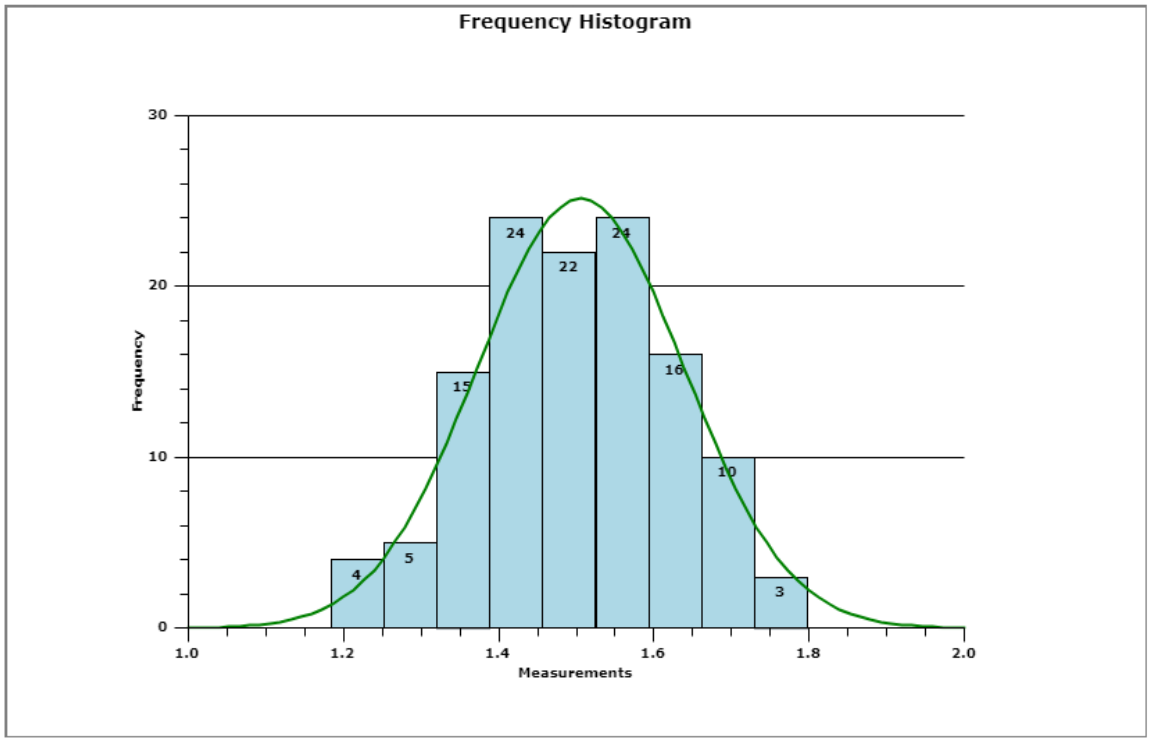
This Attribute Control chart is a combination of the u-chart and the c-chart. The chart normalizes the defect rate, expressing it as defects per million. The chart displays the defect rate as defects per million. The table above gives the defect count in both absolute terms, and in the normalized defects per million used by the chart.

Other Important SPC Charts

Frequency Histogram Chart

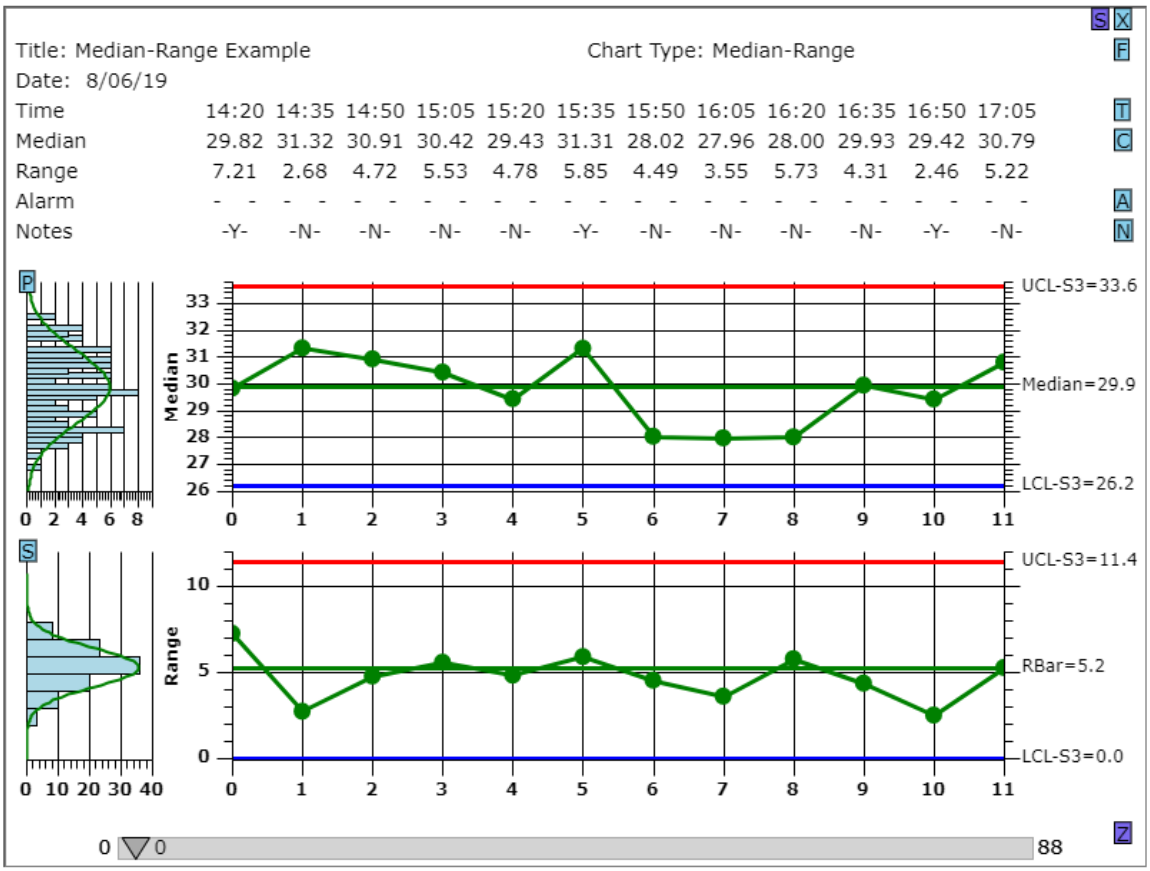
An SPC control chart tacks the trend of critical variables in a production environment. It is important that the production engineer understand whether or not changes or variation in the critical variables are natural variations due to the tolerances inherent to the production machinery, or whether or not the variations are due to some systemic, assignable cause that needs to be addressed. If the changes in critical variables are the result of natural variations, a frequency histogram of the variations will usually follow

one of the common continuous (normal, exponential, gamma, Weibull) or discrete (binomial, Poisson, hypergeometric) distributions. It is the job of the SPC engineer to know what distribution best models his process. Periodically plotting of the variation of critical variables will give SPC engineer important information about the current state of the process. A typical frequency histogram looks like:



Frequency Histogram Chart

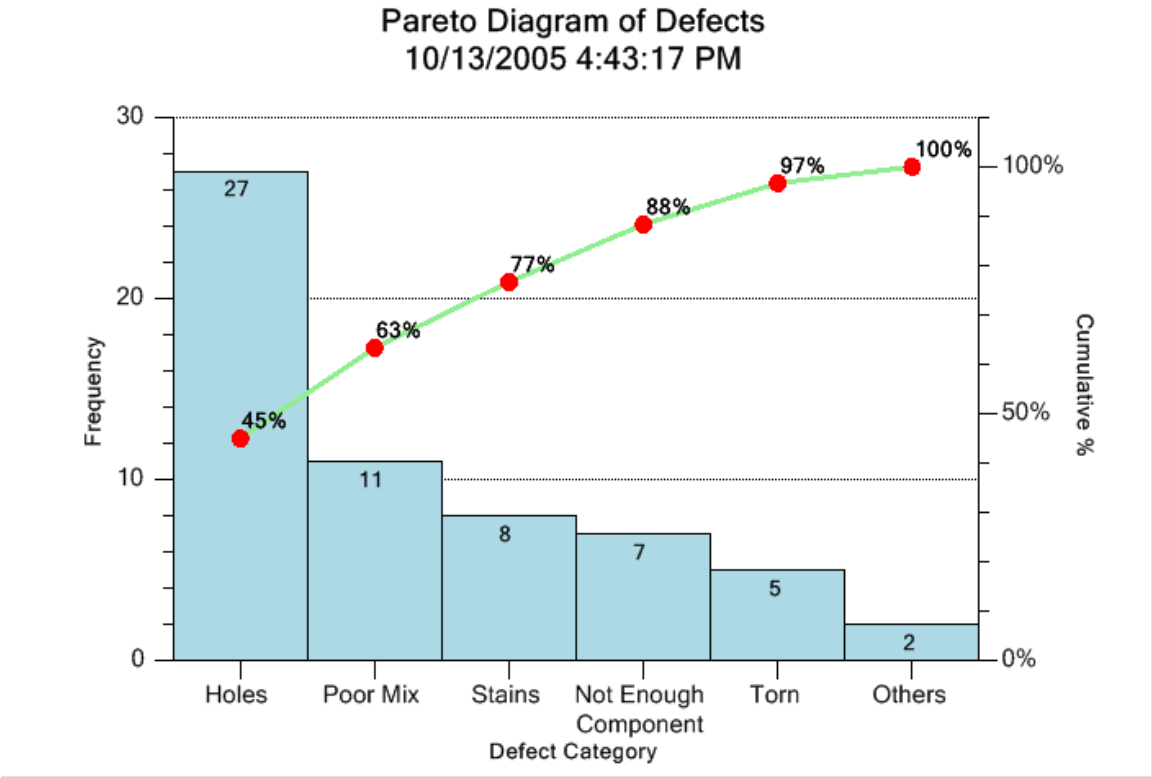
Viewing frequency histograms of both the variation in the primary variable (Mean, Median, count, or actual value), and the secondary variable (Range, Sigma or Moving Range) side-by-side with the SPC control chart makes it even easier to find out whether the variations are the result of natural variations or the result of some systemic change in the process.



XBar-Sigma Chart with Integral Frequency Histograms

Pareto Diagrams

The Pareto diagram is a special type of bar graph combined with a line plot, plotted in a chart that uses two different y-axis scales. The bar graph scale is a frequency scale that measures the number of times a specific problem has occurred. The line plot scale is a cumulative percentage scale.



Pareto Chart

The chart is easy to interpret. The tallest bar, the left-most one in a Pareto diagram, is the problem that has the most frequent occurrence. The shortest bar, the right-most one, is the problem that has the least frequent occurrence. Time spend on fixing the biggest problem will have the greatest affect on the overall problem rate. This is a simplistic view of actual Pareto analysis, which would usually take into account the cost effectiveness of fixing a specific problem. Never less, it is powerful communication tool that the SPC engineer can use in trying to identify and solve production problems.

Chapter 3 - Class Architecture of the SPC Control Chart Tools for JavaScript/TypeScript Class Library

Major Design Considerations

This chapter presents an overview of the **SPC Control Chart Tools for JavaScript/TypeScript** class architecture. Major design consideration specific to **SPC Control Chart Tools for JavaScript/TypeScript** are:

- Direct support for the following SPC chart types:

Variable Control Charts

Fixed sample size subgroup control charts
- X-Bar R – (Mean and Range) chart
- X-Bar Sigma (Mean and Sigma) chart
- Median and Range (Median and Range) chart
- X-R (Individual Range Chart) chart
- EWMA (Exponentially Weighted Moving Average Chart)
- MA (Moving Average Chart)
- MAMR (Moving Average/Moving Range)
- MAMS (Moving Average/Moving Sigma)
- CuSum (Tabular Cumulative Sum Chart)
- Levey-Jennings

Variable sample size subgroup control charts
- X-Bar Sigma (Mean and Sigma Chart)

Attribute Control Charts

Fixed sample size subgroup control charts
- p-Chart (Fraction or Percent of Defective Parts, Fraction or Percent Non-Conforming)
- np-Chart (Number of Defective Parts, Number of Non-Conforming)
- c-Chart (Number of Defects, Number of Non-Conformities)
- u-Chart (Number of Defects per Unit, Number of Non-Conformities Per Unit)
- DPMO (Number of Defects per Million)

Variable sample size subgroup control charts
- p-Chart (Fraction or Percent of Defective Parts)
- u-Chart (Number of Defects per Unit)

SPC Analysis Charts

Frequency Histograms
Pareto Charts

- Minimal programming required – create SPC charts with a few lines of code using our SPC chart templates.
- Integrated frequency histograms support – Display frequency histograms of sampled data, displayed side-by-side, sharing the same y-axis, with the SPC chart.
- Charts Header Information – Customize the chart display with job specific information, for example: Title, Operator, Part Number, Specification Limits, Machine, etc.
- Table display of SPC data – Display the sampled and calculated values for a SPC chart in a table, directly above the associated point in the SPC chart, similar to standardized SPC worksheets.
- Automatic calculation of SPC control limits – Automatically calculate SPC control limits using sampled data, using industry standard SPC control limit algorithms unique to each chart type.
- Automatic y-Axis scaling – Automatically calculated the y-axis scale for SPC charts, taking into account sampled and calculated data points, and any control limit lines added to the graph.
- Alarms – When monitored value exceeds a SPC control limit it can trigger an event that vectors to a user-written alarm processing delegate.
- SPC Process Capability Calculations -Variable Control Charts include Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppl, Ppu, and Ppk process capability statistics
- Notes – The operator can view or enter notes specific to a specific sample subgroup using a special notes tooltip.
- Data tooltips – The operator can view chart data values using a simple drill-down data tooltip display. The Data tooltips can optionally display sample subgroup data values and statistics, including process capability calculations (Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppl, Ppu, and Ppk) and customized using notes that have been entered for the sample subgroup.
- Scrollable view – Enable the scroll bar option and scroll through the chart and table view of the SPC data for an unlimited number of sample subgroups.
- Other, optional features – There are many optional features that SPC charts often use, including:

Multiple SPC control limits, corresponding to +-1, 2 and 3 sigma limits.

Scatter plots of all sampled data values on top of calculated means and medians.

Data point annotations

The chapter also summarizes the classes in the **SPC Control Chart Tools for JavaScript/TypeScript** library.

SPC Control Chart Tools for JavaScript/TypeScript Class Summary

The **SPC Control Chart 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.

The **QCChart2D** classes included in the qcspcchartts.js library file represent a subset of the standalone QCChart2D software. In general, only **QCChart2D** classes necessary for the display of the **QCSPCChart** SPC charts are included, in order minimize the payload of the initial qcspcchartts.js download into the client browser. Most, but not all, of your interaction with the software will take place through QCSPCChart classes. There are many static constants you may need to use, found in the QCChart2D class ChartConstants. Also, there a some basic classes you will also need to know how to use in order to customize the SPC Charts. These classes are covered at the end of this chapter..

QCChart2D Class Summary

Chart view class — The chart view class is the base class for charts which manages a wide range of chart 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 contour 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, directly and indirectly System.EventHandler 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.

Miscellaneous utility classes Other classes use these for data storage, file I/O, and data processing.

The **SPC Control Chart Tools for JavaScript/TypeScript** classes are in addition to the ones above. They are summarized below.

SPC Control Chart Tools for JavaScript/TypeScript Class Hierarchy

The diagram below depicts the class hierarchy of the **SPC Control Chart Tools for JavaScript/TypeScript** library without the additional **QCChart2D** classes

.ChartView
- FrequencyHistogramChart
- ParetoChart
- ProbabilityChart
- SPCChartBase
  - SPCEventAttributeControlChar
    - SPCBatchAttributeControlChart
    - SPCTimeAttributeControlChart
  - SPCTimeVariableControlChart

SPCBatchAttributeControlChart
SPCTimeAttributeControlChart

StringLabel
    NotesLabel
MouseListener
    NotesToolTip
DataToolTip
    SPCDataToolTip

QCSPCChart Classes

SPCControlChartData
SPCControlLimitAlarmArgs
SPCControlLimitRecord
SPCCalculatedValueRecrod
SPCProcessCapabilityRecord
SPCSampledValueRecord
SPCControlParameters
SPCGeneralizedTableDisplay
SPCControlPlotObjects
SPCChartObjects

SPC Control Chart Data

SPCControlChartData

SPC control chart data is stored in the **SPCControlChartData** class. It holds the header information used to customize the chart table, the raw sample data used to prepare the chart, the calculated chart values used in the chart, and the SPC control limits. It contains array lists of **SPCSampledValueRecord**, **SPCControlLimitRecord** and **SPCCalculatedValueRecord** objects.

SPCSampledValueRecord

This class encapsulates a sample data value. It includes a description for the item, the current value of the sampled value, and a history of previous values.

SPCControlLimitRecord

This class holds information specific to a SPC control limit: including the current value of the control limit, a history of control limit values, description, and the hysteresis value for alarm checking.

SPCCalculatedValueRecord

The record class for a calculated SPC statistic. It holds the calculated value type (mean, median, sum, variance, standard deviation, etc.), value, description and historical data.

SPCProcessCapabilityRecord

The record class for storing and calculating process capability statistics: Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppl, Ppu and Ppk.

SPC Charts and Related Chart Objects

SPCChartBase

The **SPCChartBase** forms the base object for all SPC control charts. The variable control chart templates (**SPCBatchVariableControlChart, and SPCTimeVariableControlChart**) are derived from the **SPCEventVariableControlChart** class, which in turn is derived from **SPCChartBase**. The attribute control charts (**SPCBatchAttributeControlChart** and **SPCTimeAttributeControlChart**) are derived from the **SPCEventAttributeControlChart** class, which in turn is derived from SPCChartBase.

SPCEventVariableControlChart

A Variable Control Chart class that uses an EventCoordinate system with an event based X-Axis. This class creates MEAN_RANGE_CHART, MEDIAN_RANGE_CHART, INDIVIDUAL_RANGE_CHART, MEAN_SIGMA_CHART, MEAN_SIGMA_CHART_VSS, EWMA_CHART, TABCUSUM, MA_CHART, MAMR_CHART and MAMS_CHART chart types.

SPCBatchVariableControlChart

A Batch Variable Control Chart class that uses a CartesianCoordinate system with a numeric based X-Axis. This class creates MEAN_RANGE_CHART, MEDIAN_RANGE_CHART, INDIVIDUAL_RANGE_CHART, MEAN_SIGMA_CHART, MEAN_SIGMA_CHART_VSS, EWMA_CHART, TABCUSUM, MA_CHART, MAMR_CHART, LEVEY_JENNINGS_CHART and MAMS_CHART chart types.

SPCTimeVariableControlChart

A Variable Control Chart class that uses a TimeCoordinate system with a time based X-Axis. This class creates MEAN_RANGE_CHART, MEDIAN_RANGE_CHART, INDIVIDUAL_RANGE_CHART, MEAN_SIGMA_CHART, MEAN_SIGMA_CHART_VSS, EWMA_CHART, TABCUSUM, MA_CHART, MAMR_CHART and MAMS_CHART chart types.

SPCEventAttributeControlChart

An Attribute Control Chart class that uses an EventCoordinates system with an Event X-Axis. This class creates PERCENT_DEFECTIVE_PARTS_CHART, FRACTION_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTS_PERUNIT_CHART, NUMBER_DEFECTS_CHART SPC, NUMBER_DEFECTS_PER_MILLION_CHART, PERCENT_DEFECTIVE_PARTS_CHART_VSS, FRACTION_DEFECTIVE_PARTS_CHART_VSS, NUMBER_DEFECTS_PERUNIT_CHART_VSS chart types

SPCBatchAttributeControlChart

A Batch Attribute Control Chart class that uses a CartesianCoordinate system with a numeric X-Axis. This class creates PERCENT_DEFECTIVE_PARTS_CHART, FRACTION_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTS_PERUNIT_CHART, NUMBER_DEFECTS_CHART SPC, NUMBER_DEFECTS_PER_MILLION_CHART,

PERCENT_DEFECTIVE_PARTS_CHART_VSS, FRACTION_DEFECTIVE_PARTS_CHART_VSS, NUMBER_DEFECTS_PERUNIT_CHART_VSS chart types.

SPCTimeAttributeControlChart

An Attribute Control Chart class that uses a TimeCoordinate system with a time based X-Axis This class creates PERCENT_DEFECTIVE_PARTS_CHART, FRACTION_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTS_PERUNIT_CHART, NUMBER_DEFECTS_CHART, NUMBER_DEFECTS_PER_MILLION_CHART, PERCENT_DEFECTIVE_PARTS_CHART_VSS, FRACTION_DEFECTIVE_PARTS_CHART_VSS, NUMBER_DEFECTS_PERUNIT_CHART_VSS chart types.

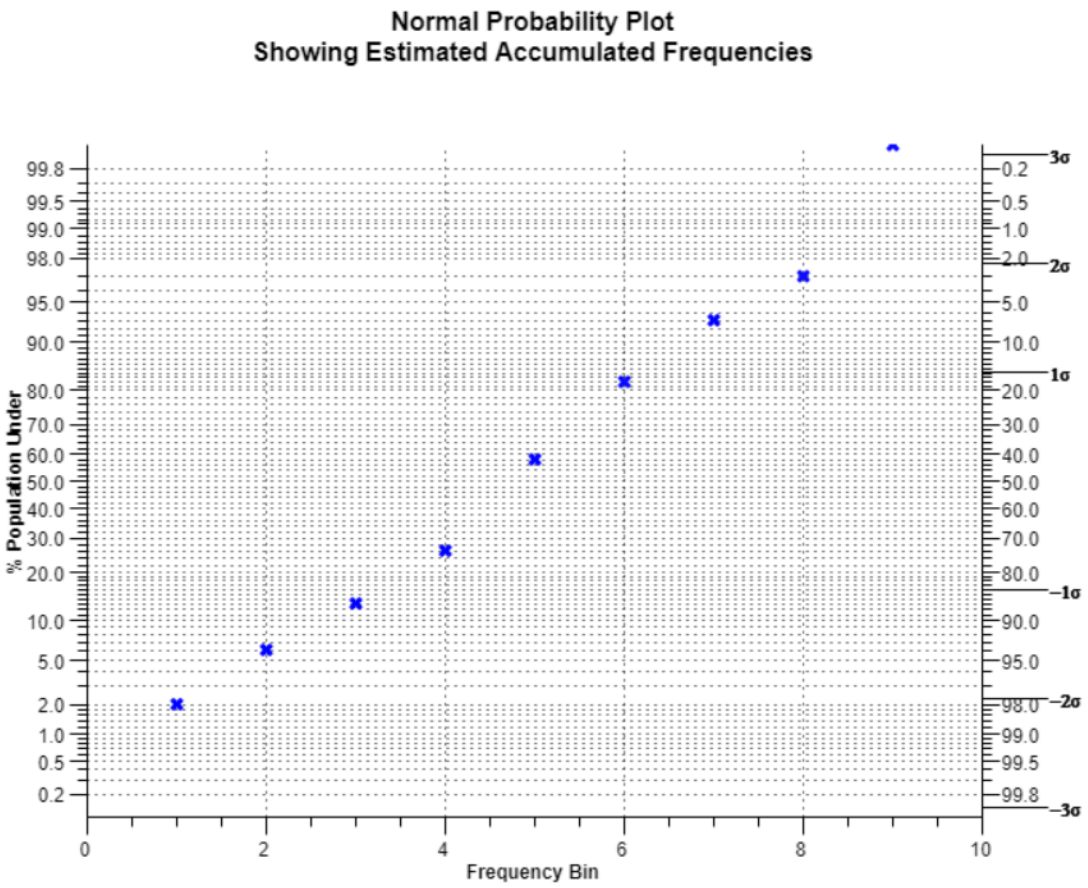
FrequencyHistogramChart

A *Frequency Histogram* checks that the variation in a process variable follows the predicted distribution function (normal, Poisson, chi-squared, etc). The class includes all of the objects needed to draw a complete frequency histogram chart. These objects include objects for data, a coordinate system, titles, axes, axes labels, grids and a bar plot.

ParetoChart

The *Pareto Diagram* is special type of bar graph combined with a line plot, plotted in a chart that uses two different y-axis scales. The bar graph scale is a frequency scale that measures the number of times a specific problem has occurred. The line plot scale is a cumulative percentage scale. The class includes all of the objects needed to draw a complete Pareto chart. These objects include objects for data, coordinate systems, titles, axes, axes labels, grids, numeric labels, and a line plot and bar plot.

Cumulative Normal Probability Chart



Probability Plots

The **ProbabilityChart** class is a highly specialized chart template used to plot cumulative frequency data using a coordinate system that has a cumulative probability y-scale. The class includes all of the objects needed to draw a complete Probability chart. These objects include objects for data, coordinate systems, titles, axes, axes labels, grids, numeric labels, and scatter plot. New classes were developed for the **QCChart2D** charting software capable of rendering of probability chart coordinate systems (**ProbabilityScale**, **ProbabilityAutoScale**, **ProbabilityCoordinates**) and probability axes (**ProbabilityAxis**, **ProbabilitySigmaAxis**).

ProbabilityScale

The **ProbabilityScale** class implements a cumulative normal probability coordinate system for a single coordinate, x or y. Two such scales provide the scaling routines for x and y in an **PhysicalCoordindates** derived class, **CartesianCoordinates**, for example. This allows for different x and y scale types (linear, cumulative normal probability, time) to be installed independently for x- and y-coordinates.

ProbabilityAutoScale

The **ProbabilityAutoScale** class is used with cumulative normal probability coordinates and auto-scales the plotting area of graphs and to set the minimum and maximum values of the axes displayed in the graphs.

ProbabilityCoordinates

The **ProbabilityCoordinates** class extends the **PhysicalCoordinates** class to support a y-axis probability scale in an xy coordinate plane.

ProbabilityAxis The **ProbabilityAxis** class implements a probability axis where the major tick marks are placed at intervals appropriate to a cumulative probability scale.

ProbabilitySigmaAxis

The **ProbabilitySigmaAxis** class implements a linear axis where the tick marks are placed at linear intervals on the sigma levels of the associated probability scale.

NotesLabel The **NotesLabel** class displays the Notes items in the SPC table.

NotesToolTip The **NotesToolTip** displays the Notes tooltip for the notes items in the SPC table.

SPCDataToolTip The **SPCDataTooTip** displays the data tooltip for SPC Charts..

SPC Calculations

SPCArrayStatistics SPC involves many statistical calculations. The **SPCArrayStatistics** class includes routines for the calculation of sums, means, medians, ranges, minimum values, maximum values, variances, and standard deviations. It also includes routines for array sorting and calculating frequency bins for frequency histograms. It also includes functions that compute cumulative probability values for normal, Poisson, and chi-squared distributions.

SPCControlParameters

The **SPCControlParameters** class contains the factors and formulas for calculating SPC control chart limits for *Variable* and *Attribute Control Charts*. It includes calculations for the most

common SPC charts: X-Bar R, Median and Range, X-Bar Sigma, X-R, u-chart, p-chart, np-chart, and c-chart.

Tabular Display

SPCGeneralizedTableDisplay

The **SPCGeneralizedTableDisplay** manages a list of **ChartText** objects (**NumericLabel**, **StringLabel** and **TimeLabel** objects), that encapsulate each unique table entry in the SPC chart table. This class also manages the spacing between the rows and columns of the table, and the alternating stripe used as a background for the table.

SPC Control Alarms

SPCControlLimitAlarmArgs

This class passes event information to a **SPCControlLimitAlarmEventDelegate** alarm processing delegate.

SPCControlLimitAlarmEventDelegate

A delegate type for hooking up control limit alarm notifications

QCChart2D Constants and Classes

class ChartColor

The ChartColors class contains static constants for all of the standard RGB colors.

public static readonly ALICEBLUE: number = (0xfff0f8ff);
public static readonly ANTIQUEWHITE: number = (0xfffaebd7);
public static readonly AQUA: number = (0xff00ffff) ;
public static readonly AQUAMARINE: number = (0xff7fffd4) ;
.

.
.
public static readonly WHITESMOKE: number = (0xffF5F5F5);
public static readonly YELLOW: number = (0xffFFFF00);
public static readonly YELLOWGREEN: number = (0xff9ACD32);

If also contains static function for creating colors from RGB components.

Public Static Functions

static numberToRGBString(n: number): string	Convert a number into a ARG string.
static numberToRGBAString(n: number): string	Convert a number into a ARGB string.
static RGBStringToNumber(s: string): number	Convert a string into a RGB number.
static convertStringToNumber(s: string): number	Convert the string into a number.
static RGBToColor(r: number, g: number, b: number): number	Convert RGB components into an RGB number.
static ARGBToColor(a: number, r: number, g: number, b: number): number	Convert ARGB components into an ARGB number.
static RGBAToColor(r: number, g: number, b: number, a: number): number	Convert RGBA components into and an ARGB number
static fromAC(a: number, c: number): number	Add an Alpha component to the specified color.
static stringToColor(c: string): number	Convert a string to a color number.
static getAlpha(c: number): number	Return the Alpha component of the specified color number.
static getRed(c: number): number	Return the red component of the specified color number.
static getGreen(c: number): number	Return the green component of the specified color number
static getBlue(c: number): number	Return the blue components of the specified: number.
static getA(c: number): number	Return the Alpha component of the specified color number.
static getR(c: number): number	Return the red component of the specified color number.
static getG(c: number): number	Return the green component of the

	specified color number
static getB(c: number): number	Return the blue components of the specified number.

Example

```
charttable.setBackgroundColor1( QCSPCChartTS.ChartColor.LIGHTGRAY);
```

class ChartConstants

The ChartConstants class contains static constants use throughout the software.

Public Static Fields

Line Plot Drawing Constants	
NO_STEP=0	Interpolate directly from one point to the next, drawing a line from (x1, y1) to (x2, y2)
STEP_START =1	Connect two points by drawing two lines (x1, y1) to (x1, y2) to (x2, y2)
STEP_END =2	Connect two points by drawing two lines (x1, y1) to (x2, y1) to (x2, y2)
STEP_NO_RISE_LINE =3	Connect two points by first drawing a line from (x1, y2) to (x2, y2)
Chart Background Type Constants	
PLOT_BACKGROUND = 1	Background for the plotting area of the chart
GRAPH_BACKGROUND = 0	Background for the graph area of the chart
Chart Background Fill Constants	
CHARTATTRIBUTE_BGMODE = 5	Chart Attribute background mode
ALTBARMODE = 4	Use user-defined texture for background texture
Chart Background Fill Constants	
USERTEXTUREMODE = 3	Use user-defined texture for background texture
USERGRADIENTMODE = 2	Use user-defined gradient for background gradient
SIMPLEGRADIENTMODE = 1	Use simple gradient for background gradient
SIMPLECOLORMODE = 0	Use a single color for background

Justification Constants

JUSTIFY_MIN =0	Justify in the direction of decreasing coordinate values
JUSTIFY_CENTER =1	Center justify
JUSTIFY_MAX =2	Justify in the direction of increasing coordinate values
INSIDE_INDICATOR = 5	Position text inside indicator
INSIDE_BAR = 5	Position bar data value inside bar
OUTSIDE_BAR =6	Position bar data value outside bar
OUTSIDE_INDICATOR =6	Position text value outside indicator
CENTERED_BAR =7	Position bar data value centered, inside bar

Position Constants

POS_LEFT =0	Position to the left of the object
POS_RIGHT =2	Position to the right of the object
POS_TOP =3	Position above the object
POS_BOTTOM =5	Position below the object

Direction Constants

VERT_DIR = 1	Oriented in the vertical direction
HORIZ_DIR = 0	Oriented in the horizontal direction

Axis Constants

X_AXIS = 0	Specifies the x-axis, or an x-coordinate value
Y_AXIS = 1	Specifies the y-axis, or an y-coordinate value

Scale Type Constants

LINEAR_SCALE=0	Specifies a linear scale
LOG_SCALE= 1	Specifies a log scale
TIME_SCALE=2	Specifies a time scale
ELAPSEDTIME_SCALE = 4	Specifies an elapsed time scale
EVENT_SCALE = 5	Specifies an event scale

Axis and Tick Mark Direction Constants

AXIS_MIN = 0	Draw in the direction of decreasing coordinate values
AXIS_CENTER = 1	Center justify
AXIS_MAX = 2	Draw in the direction of increasing coordinate values

Line style constants	
static LS_SOLID	Solid line (default)
static LS_DASH_8_4	repeating pattern: 8 dots on, 4 dots off
static LS_DASH_4_4	repeating pattern: 4 dots on, 4 dots off
static LS_DASH_4_2	repeating pattern: 4 dots on, 2 dots off
static LS_DASH_2_2	repeating pattern: 2 dots on, 2 dots off
static LS_DOT_1_1	repeating pattern: 1 dots on, 1 dots off
static LS_DOT_1_2	repeating pattern: 1 dots on, 2 dots off
static LS_DOT_1_4	repeating pattern: 1 dots on, 4 dots off
static LS_DOT_1_8	repeating pattern: 1 dots on, 8 dots off
static LS_DASH_DOT	repeating pattern: 8 dots on, 4 dots off, 1 dot on, 4 dots off
Auto Axes Type Constants	
NO_AUTOSCALE= 3	Auto-scale round mode set is none, for internal use only, no rounding may occur, may not work under all conditions
AUTOAXES_FAR= 2	Auto-scale round mode set is far, usually rounding to the next major tick mark
AUTOAXES_NEAR = 1	Auto-scale round mode set is near, usually rounding to the next minor tick mark
AUTOAXES_EXACT = 0	Auto-scale using the exact minimum and maximum values, no rounding
Auto Axes Stack Constants	
AUTOAXES_UNSTACKED =0	Assume data is not stacked in auto-scale calculations
AUTOAXES_STACKED =1	Assume data is stacked in auto-scale calculations
Axis Labels Constants	
LABEL_ALL=7	Label the axis minimum point, origin, and maximum point
LABEL_MIN=1	Label the axis minimum point
LABEL_INTERCEPT =2	Label the axis origin point
LABEL_ORIGIN =2	Label the axis origin point
LABEL_MAX =4	Label the axis maximum point
Position Type Constants	
DEV_POS = 0	Position defined using device, or window device coordinates
USER_POS = 0	Position defined using device, or window device coordinates

PHYS_POS = 1	Position defined using physical coordinates
POLAR_POS = 2	Position defined using polar coordinates
NORM_GRAPH_POS = 3	Position defined using graph normalized coordinates
NORM_PLOT_POS = 4	Position defined using plot normalized coordinates

Border Constants

LEFT_BORDER = 0	Specifies the left border
TOP_BORDER = 1	Specifies the top border
RIGHT_BORDER = 2	Specifies the right border
BOTTOM_BORDER = 3	Specifies the bottom border

Legend Header and Footer Constants

LEGEND_HEADER = 0	Specifies the legend header, the topmost line of the legend
LEGEND_SUBHEAD = 1	Specifies the legend subhead, the line directly under the legend header
LEGEND_FOOTER = 2	Specifies the legend footer, the last most line of the legend

Chart Header and Footer Constants

CHART_HEADER = 0	Specifies the chart header, justified with respect to the top of the chart
CHART_SUBHEAD = 1	Specifies the chart subhead, the line directly under the chart header
CHART_FOOTER = 2	Specifies the chart footer, justified with respect to the bottom of the chart
CENTER_GRAPH = 0	Center title with respect to the graph area
CENTER_PLOT = 1	Center title with respect to the plot area

Chart Marker Type Constants

MARKER_NULL = 0	No marker
MARKER_VLINE= 1	Vertical line marker
MARKER_HLINE= 2	Horizontal line marker
MARKER_CROSS= 3	Cross hair marker
MARKER_BOX= 4	Box marker
MARKER_HVLINE=5	Combination Horizontal and Vertical line marker

Scatter Plot Chart Symbol Constants

NOSYMBOL= 0	No scatter plot symbol
SQUARE= 1	Square scatter plot symbol

UPTRIANGLE= 2	Up triangle scatter plot symbol
DOWNTRIANGLE= 3	Down scatter plot symbol
DIAMOND = 4	Diamond scatter plot symbol
CROSS = 5	Cross scatter plot symbol
PLUS= 6	Plus scatter plot symbol
STAR= 7	Star scatter plot symbol
LINE= 8	Line scatter plot symbol
HBAR= 9	Horizontal bar scatter plot symbol
VBAR = 10	Vertical bar scatter plot symbol
CIRCLE = 11	Circle scatter plot symbol
CUSTOMSYMBOL = 12	Custom scatter plot symbol

Mouse Button Constants

RIGHT_BUTTON = 2	Right button
LEFT_BUTTON = 1	Left button
MIDDLE_BUTTON = 3	Middle button

Axis Label Numeric Format Constants

DEFAULTFORMAT =0	Default format
DECIMALFORMAT =1	Decimal (1234.5678) numeric format
SCIENTIFICFORMAT =2	Scientific (1.2345678e4) numeric format
BUSINESSFORMAT =3	Business (1.23K) numeric format
ENGINEERINGFORMAT =4	The engineering format use decimal format for numbers less than +-10e6,and scientific format for all others
PERCENTFORMAT = 5	The number is multiplied by 100 and displayed with a trailing % symbol
EXPONENTFORMAT = 6	The exponent is similar to the scientific format, except that theexponent is displayed as a superscript
CURRENCYBUSINESSFORMAT = 7	Similar to the business format, except that a $ is appended to the front.
CURRENCYFORMAT = 8	Similar to the decimal format except a $ is appended to the front.
TIMEDATEFORMAT = 9	A time/date numeric format
PERCENT_NOPERCENTSIGN_FORMAT = 10	The number is multiplied by 100 and displayed without a trailing % symbol
SIGMA_FORMAT = 12	The number is multiplied by 100 and displayed with a trailing % symbol

Time/Date Axis Time Label Constants

TIMEDATEFORMAT_NONE =0	No time format

TIMEDATEFORMAT_24HMS = 5	24 hour (hour:minutes:seconds) time format - Example 23:59:59
TIMEDATEFORMAT_24HM = 6	hour (hour:minutes) time format - Example 23:59
TIMEDATEFORMAT_MS = 7	(minutes:seconds) time format - Example 59:59
TIMEDATEFORMAT_12HMS = 8	12 hour (hour:minutes:seconds) time format - Example 11:59:59
TIMEDATEFORMAT_12HM = 9	12 hour (hour:minutes) time format - Example 1:59
TIMEDATEFORMAT_24HMSD = 10 24 hour (hour:minutes:seconds.milliseconds) time format - Example 23:59:59.9	
TIMEDATEFORMAT_24HMSDD = 11	24 hour (hour:minutes:seconds.milliseconds) time format - Example 23:59:59.99
TIMEDATEFORMAT_12HMSD = 12	12 hour (hour:minutes:seconds.milliseconds) time format - Example 11:59:59.999
TIMEDATEFORMAT_12HMSDD = 13	12 hour (hour:minutes:seconds.milliseconds) time format - Example 11:59:59.9
TIMEDATEFORMAT_MSD = 14	(minutes:seconds) time format - Example 59:59.9
TIMEDATEFORMAT_MSDD = 15	(minutes:seconds.milliseconds) time format - Example 59:59.99
TIMEDATEFORMAT_MSDDD = 16	(minutes:seconds.milliseconds) time format - Example 59:59.999

Time/Date Axis Date Label Constants

TIMEDATEFORMAT_STANDARD =20	Date format (Jan 2, 1998)
TIMEDATEFORMAT_MDY=21	Date format (1/2/98, 1/2/02)
TIMEDATEFORMAT_DMY=22	Date format (2/1/98, 2/1/02)
TIMEDATEFORMAT_MY =23	Date format (1/98)
TIMEDATEFORMAT_Q =24	Date format (Q1)
TIMEDATEFORMAT_MMMM =25	Date format (January)
TIMEDATEFORMAT_DDDD =26	Date format (Tuesday)
TIMEDATEFORMAT_MMM =27	Date format (Jan)
TIMEDATEFORMAT_DDD =28	Date format (Tue)
TIMEDATEFORMAT_M =29	Date format (J)
TIMEDATEFORMAT_D =30	Date format (T)
TIMEDATEFORMAT_Y=31	Date format (98)
TIMEDATEFORMAT_MDY2000=32	Date format (1/2/1998, 1/2/2002)

TIMEDATEFORMAT_DMY2000=33	Date format (2/1/1998, 2/1/2002)
TIMEDATEFORMAT_MY2000 =34	Date format (1/1998)
TIMEDATEFORMAT_MDY_HMS = 35	Date Time format (1/23/1998 14:12:32)
TIMEDATEFORMAT_DMY_HMS = 36	Date Time format (23/1/1998 14:12:32)
TIMEDATEFORMAT_MDY_HM = 37	Date Time format (1/23/1998 14:12)
TIMEDATEFORMAT_DMY_HM = 38	Date Time format (23/1/1998 14:12)
TIMEDATEFORMAT_MDYHMS = 39	Date Time format (1/23/1998 14:12:32)
TIMEDATEFORMAT_DMYHMS = 40	Date Time format (23/1/1998 14:12:32)
TIMEDATEFORMAT_MDYHM = 41	Date Time format (1/23/1998 14:12)
TIMEDATEFORMAT_DMYHM = 42	Date Time format (23/1/1998 14:12)
TIMEDATEFORMAT_DM = 43	Date Time format (23/1)
TIMEDATEFORMAT_MD = 44	Date Time format (1/23 14:12)
TIMEDATEFORMAT_Y2000=50	Date format (1998)

Label Cross-over constants

CROSSOVER_NONE = 0	No crossover labels for time axis
CROSSOVER_DAY = 1	Check for day crossover labels for time axis
CROSSOVER_WEEK = 2	Check for week crossover labels for time axis
CROSSOVER_MONTH = 3	Check for month crossover labels for time axis
CROSSOVER_YEAR = 4	Check for year crossover labels for time axis
CROSSOVER_HOUR = 5	Check for hour crossover labels for time axis
CROSSOVER_MINUTE = 6	Check for minute crossover labels for time axis
CROSSOVER_SECOND = 7	Check for second crossover labels for time axis
CROSSOVER_MILLISECOND = 8	Check for millissecond crossover labels for time axis

Time label Date Crossover Mode

NO_DATECROSSOVER = 0	Do not display date crossover labels for time axis
REPLACE_DATECROSSOVER = 1	Replace the standard tick mark label with a crossover tick mark label
UNDER_DATECROSSOVER = 2	Place the crossover label under the standard tick mark label
INFRONT_DATECROSSOVER = 3	Place the crossover label in front of the standard tick mark label
BEHIND_DATECROSSOVER = 4	Place the crossover label behind the standard tick mark label

Axis label overlap constants	
OVERLAP_LABEL_DRAW = 0	Draw all tick mark labels, even if they overlap
OVERLAP_LABEL_DELETE = 1	If a tick mark label overlaps the previous tick mark label, do not draw the label
OVERLAP_LABEL_STAGGER = 2	If a tick mark label overlaps the previous tick mark label, do not draw the label
Week Type Constants for Time/Date Axis	
WEEK_7D =0	7 day week mode (Sunday through Saturday)
WEEK_5D =1	5 day week mode (Monday through Friday)
Move data point Constants	
MOVE_X=0	Restrict the direction of move operations to the x-coordinate direction
MOVE_Y=1	Restrict the direction of move operations to the y-coordinate direction
MOVE_XY = 2	Allow unrestricted move operations
Nearest Point Constants	
FNP_X=0	Calculate the nearest point based on the x-coordinate physical coordinate system
FNP_Y=1	Calculate the nearest point based on the y-coordinate physical coordinate system
FNP_DIST= 2	Calculate the nearest point based on the shortest distance using the physical coordinate system
FNP_NORMX = 3	Calculate the nearest point based on the x-coordinate plot normalized coordinate system
FNP_NORMY = 4	Calculate the nearest point based on the y-coordinate plot normalized coordinate system
FNP_NORMDIST =5	Calculate the nearest point based on the shortest distance using the physical coordinate system. This should be used if the x and y physical coordinate system do not have an approximately1:1 aspect ratio.
Graphic Object Process Flags	
OBJECT_DISABLE = 0	Do not process the graph object

OBJECT_ENABLE =1	Process and draw the graph object
OBJECT_ENABLE_NODRAW= 2	Process the graph object so that positioning information is accurate, but do not draw.
Text and object resize modes	
NO_RESIZE_OBJECTS = 0	Do not resize graph objects when the chart window is resized
AUTO_RESIZE_OBJECTS = 1	Resize the objects in the graph when the chart window is resized
MANUAL_RESIZE_OBJECTS = 2	The resizeMultiplier property for each objects is used to resize the object.This property should be explicitly set by the programmer if this mode is set.
Table structure constants	
COLUMN_MAJOR = 1	Data in a file is organized so that adjacent x-values are rows in the file,each group of y-values start a new column
ROW_MAJOR = 0	Data in a file is organized so that adjacent x-values are columns in the file,each group of
y-values start a new row	
// Fast clipping constants	
NO_FASTCLIP = 2	Normal, complete, clipping of entire dataset against x and y plot area coordinate system.
FASTCLIP_X = 0	Data is assumed monotonic (always increasing) in x. All data before the firstx-value of the dataset actually in the plot area is discarded, not drawn and clipped as in the NO_FASTCLIP mode.All data after the first x-value that is greater than the ending x-coordinate of the plot area is also discarded. This can speed things up because if a dataset has 1,000,000 data points, and only 100 are in the plotting area, only 100 points are actually plotted using the line drawing routines. This is most useful in zooming applications.
FASTCLIP_Y = 1	Data is assumed monotonic (always increasing) in y. All data before the first

Tooltip constants

DATA_TOOLTIP_CUSTOM = 0	Specifies a user-defined tooltip format
DATA_TOOLTIP_X = 1	Displays the x-value of a datapoint in the tooltip box
DATA_TOOLTIP_Y = 2	Displays the y-value of a datapoint in the tooltip box
DATA_TOOLTIP_XY_ONELINE = 3	Displays the x- and y-values of a datapoint on a single line, separated by a comma, in the tooltip box
DATA_TOOLTIP_XY_TWOLINE = 4	Displays the x- and y-values of a datapoint on two separate lines in the tooltip box
DATA_TOOLTIP_GROUP_MULTILINE = 5	Displays the x- and y-values of a group dataset on separate lines in the tooltip box
DATA_TOOLTIP_OHLC = 6	Displays the x- and y-values of a group dataset in an OHLC format on separate lines in the tooltip box
DATA_TOOLTIP_BW = 7	Displays the x- and y-values of a group dataset in an BW format on separate lines in the tooltip box
DATA_TOOLTIP_EVENT_CUSTOM = 8	Displays tooltip for a custgom event tooltip

/Date/Time constants

YEAR = 1	Field number indicating themonth.
MONTH = 2	Field number indicating theweek number within the current year.
WEEK_OF_YEAR = 3	Field number indicating theweek number within the current month.
WEEK_OF_MONTH = 4	Field number indicating theday of the month.
DAY_OF_MONTH = 5	Field number indicating the day of the month.
DATE = 5	Field number indicating the day of the month.
DAY_OF_YEAR = 6	Field number indicating the daynumber within the current year. The first day of the year has value 1.
DAY_OF_WEEK = 7	MONDAY, TUESDAY, WEDNESDAY,THURSDAY, FRIDAY, and SATURDAY.
HOUR = 10	Field number indicating thehour of the morning or afternoon. HOUR is used for the 12-hourclock.E.g., at 10:04:15.250 PM the HOUR is 10.
HOUR_OF_DAY = 11	Field number indicating thehour of the

	day. HOUR_OF_DAY is used for the 24-hour clock. E.g., at 10:04:15.250 PM the HOUR_OF_DAY is 22.
MINUTE = 12	Field number indicating theminute within the hour.E.g., at 10:04:15.250 PM the MINUTE is 4.
SECOND = 13	Field number indicating thesecond within the minute.E.g., at 10:04:15.250 PM the SECOND is 15.
MILLISECOND = 14	Field number indicating the millisecond within the second.E.g., at 10:04:15.250 PM the MILLISECOND is 250.
TICKS = 15	Field number indicating TICKS.
MTICKS = 16	Field number indicating millisecondTICKS (i.e. TICKS / 10000L).
SUNDAY = 0	Value of the DAY_OF_WEEK field indicating Sunday.
MONDAY = 1	Value of the DAY_OF_WEEK field indicating Monday.
TUESDAY = 2	lue of the DAY_OF_WEEK field indicating Tuesday.
WEDNESDAY = 3	Value of the DAY_OF_WEEK field indicating Wednesday.
THURSDAY = 4	Value of the DAY_OF_WEEK field indicating Thursday.
FRIDAY = 5	Value of the DAY_OF_WEEK field indicatingFriday.
SATURDAY = 6	Value of the DAY_OF_WEEK field indicating Saturday.
JANUARY = 1	Value of the month field indicating the first month of the year.
FEBRUARY = 2	Value of the month field indicating the second month of the year.
MARCH = 3	Value of the month field indicating the third month of the year.
APRIL = 4	Value of the month field indicating the fourth month of the year.
MAY = 5	Value of the month field indicating the fifth month of the year.
JUNE = 6	Value of the month field indicating the sixth month of the year.
JULY = 7	Value of the month field indicating the seventh month of the year.
AUGUST = 8	Value of the month field indicating the

	eighth month of the year.
SEPTEMBER = 9	Value of the month field indicating the ninth month of the year.
OCTOBER = 10	Value of the month field indicating the tenth month of the year.
NOVEMBER = 11	Value of the month field indicating the eleventh month of the year.
DECEMBER = 12	Value of the month field indicating the twelfth month of the year.

Gradient constants

GRADIENT_NONE = 0	No Gradient
GRADIENT_MAPTO_OBJECT = 1	Map the gradient breakpoints to the associated object.
GRADIENT_MAPTO_PLOT_PHYSICAL_COORDINATES = 2	Map the gradient breakpoints to the physical coordinate system of the plot
GRADIENT_MAPTO_PLOT_NORMALIZED_COORDINATES = 4	Map the gradient breakpoints to the plot normalized coordinate system of the plot.
GRADIENT_MAPTO_GRAPH_NORMALIZED_COORDINATES = 5	Map the gradient breakpoints to the plot normalized coordinate system of the plot.

Event label constants

EVENT_NO_LABEL = 0	Specifies not to label tick marks of EventTimeAxisLabels
EVENT_NUMERIC_LABEL= 1	Specifies numeric labels for tick marks of EventTimeAxisLabels
EVENT_TIME_LABEL = 2	Specifies time labels for tick marks of EventTimeAxisLabels
EVENT_ELAPSEDTIME_LABEL = 3	Specifies elapsed time labels for tick marks of EventTimeAxisLabels
EVENT_STRING_LABEL=4	Specifies string labels for tick marks of EventTimeAxisLabels
EVENT_NUMERIC_DISPLAY_LABEL = 5	Specifies numeric x display labels for tick marks of EventTimeAxisLabels
EVENT_TIME_SCALE = 1	Specifies the event axis should be scaled using time values
EVENT_NUMERIC_SCALE = 2	Specifies the event axis should be scaled using numeric values
EVENT_ELAPSEDTIME_SCALE = 3	Specifies the event axis should be scaled using elapsed time values

Event rounding constants

EVENT_ROUNDNEAREST = 0	Specifies the the event date binary search routines should be rounded to the nearest

	date
EVENT_ROUNDDOWN = 1	Specifies the the event date binary search routines should be rounded down to the nearest date
EVENT_ROUNDUP = 2	Specifies the the event date binary search routines should be rounded up to the nearest date

Zoom mode costants	
ZOOM_MODE_XY = 0	Traditional box, XY, zoom mode
ZOOM_MODE_X = 1	Zoom X mode
ZOOM_MODE_Y = 2	Zoom Y mode

Scrollbar positioning on update constants	
SCROLLBAR_POSITION_UNCHANGED = 0	Specifies the scrollbar position should remain unchanged on update
SCROLLBAR_POSITION_MIN = 1	Specifies the scrollbar position should go to minimum on update
SCROLLBAR_POSITION_MAX = 2	Specifies the scrollbar position should go to maximum on update

class ChartAttribute

The ChartAttribute class combines together attributes, such as line color, line style, line thickness, and fill color, simplifying passing attribute data to/from chart objects.

It also contains static functions for creating new ChartAttribute objects.

ChartAttribute static constructors	
Nestatic newChartAttribute(rgbcolor: number, rlinewidth: number, nlinestyle: number, rgbfillcolor: number): ChartAttribute	This constructor creates a new ChartAttribute object using the specified attributes.
static newChartAttribute2(rgbcolor: number, rlinewidth: number): ChartAttribute	This constructor creates a new ChartAttribute object using the specified attributes.
static newChartAttribute3(rgbcolor: number, rlinewidth: number, nlinestyle: number): ChartAttribute	This constructor creates a new ChartAttribute object using the specified attributes.
static newChartAttribute4(rgbcolor: number, rlinewidth: number, nlinestyle: number, rgbfillcolor: number): ChartAttribute	This constructor creates a new ChartAttribute object using the specified attributes.
static newChartAttributeAttrib(source: ChartAttribute): ChartAttribute	This constructor creates a new ChartAttribute object using

	the specified attributes.
static newChartAttributeColor(rgbcolor: number): ChartAttribute	This constructor creates a new ChartAttribute object using the specified attributes.
static newChartAttributeColorWidth(rgbcolor: number, rlinewidth: number): ChartAttribute	This constructor creates a new ChartAttribute object using the specified attributes.
static newChartAttributeColorWidthStyle(rgbcolor: number, rlinewidth: number, nlinestyle: number): ChartAttribute	This constructor creates a new ChartAttribute object using the specified attributes.
static newChartAttributeColorWidthStyleFill(rgbcolor: number, rlinewidth: number, nlinestyle: number, rgbfillcolor: number): ChartAttribute	This constructor creates a new ChartAttribute object using the specified attributes.

where

rgbcolor the rgb color used in line drawing

rgbfillcolor the rgb fill color used to fill objects (rectangles, symbols, backgrounds.

rlinewidth the thickness used in line drawing

nlinewidth the dash-dot line style used in line drawing. Use one of the static line style constants (LS_SOLID...LS_DASH_DOT) defined in ChartChartConstants.

nlinestyle The ChartConstants class contains static constants for defining different line styles (dash-dot line styles) for the ChartAttribute class.

Line style constants (found in ChartConstants)	
static LS_SOLID	Solid line (default)
static LS_DASH_8_4	repeating pattern: 8 dots on, 4 dots off
static LS_DASH_4_4	repeating pattern: 4 dots on, 4 dots off
static LS_DASH_4_2	repeating pattern: 4 dots on, 2 dots off
static LS_DASH_2_2	repeating pattern: 2 dots on, 2 dots off
static LS_DOT_1_1	repeating pattern: 1 dots on, 1 dots off
static LS_DOT_1_2	repeating pattern: 1 dots on, 2 dots off
static LS_DOT_1_4	repeating pattern: 1 dots on, 4 dots off

static LS_DOT_1_8	repeating pattern: 1 dots on, 8 dots off
static LS_DASH_DOT	repeating pattern: 8 dots on, 4 dots off, 1 do t on, 4 dots off

source a source ChartAttribute object to copy from

Example

```
   let primarychart: QCSPCChartTS.SPCChartObjects | null =
xbarsigmachart.getPrimaryChart();

    if (primarychart)
    {
    let lowspecattrib: QCSPCChartTS.ChartAttribute =
QCSPCChartTS.ChartAttribute.newChartAttribute3(QCSPCChartTS.ChartColor.GREEN, 3,
QCSPCChartTS.ChartConstants.LS_DASH_8_4);
    let upperspecattrib: QCSPCChartTS.ChartAttribute =
QCSPCChartTS.ChartAttribute.newChartAttribute3(QCSPCChartTS.ChartColor.ORANGE, 3,
QCSPCChartTS.ChartConstants.LS_DASH_8_4);

    primarychart.addSpecLimit(QCSPCChartTS.SPCChartObjects.SPC_LOWER_SPEC_LIMIT,
18.3, "L SPEC", lowspecattrib);
    primarychart.addSpecLimit(QCSPCChartTS.SPCChartObjects.SPC_UPPER_SPEC_LIMIT,
39.1, "H SPEC", upperspecattrib);

    }
```

Chapter 4 - Notes on JavaScript / TypeScript Programming

We have include dozens of examples of programming QCSPCChart using JavaScript, and TypeScript. While TypeScript is described as a superset of JavaScript, where all JavaScript code is compatible with TypeScript, that is only the case under specific circumstances. If you embrace the TypeScript style of program, using classes (similar to Java, C++ and C#), type checking, and better null checking, you will probably want to use the "strict" compiler option in your projects tsconfig.json file, such as the one from our TypeScript VariableControlCharts example, seen below.

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

}
```

The "strict" option will enforce the following rules:

Rule	Type	Default	Description
noImplicitAny	boolean	false	Raise error on expressions and declarations with an implied any type.
noImplicitReturns	boolean	false	Report an error when not all code paths in function return a value.
noImplicitThis	boolean	false	Raise error on this expressions with an implied any type.
strictBindCallApply	boolean	false	Enable stricter checking of the bind, call, and apply methods on functions.
strictFunctionTypes	boolean	false	Disable bivariant parameter checking for function types.

strictPropertyInitialization	boolean	false	Ensure non-undefined class properties are initialized in the constructor. This option requires strictNullChecks be enabled in order to take effect.
strictNullChecks	boolean	false	In strict null checking mode, the null and undefined values are not in the domain of every type and are only assignable to themselves and any (the one exception being that undefined is also assignable to void).

These rule checks pretty much rule out much of the lax, freewheeling, programming style which goes on in JavaScript code.

Strict Null Checks

Most of these rules are familiar to OOP programmers. One which is different is the strictNullChecks. In strict mode, an object cannot be assigned to the value null. For example, this is not allowed as a property declaration for a class:

```
let data: QCSPCChartTS.DoubleArray = null;
```

So, a simple comparison of an object to null is not a valid method of determining whether or not an object has been initiated. But there is a simple workaround which forces the programmer to consider whether or not an object needs to have the null state as an option. If null must be an option for an object, you should explicitly define it as such in the object declaration, using a union.

```
let data: QCSPCChartTS.DoubleArray | null = null;
```

In this case, the object *data* can either be an instance of DoubleArray, or null. And as a result of this union you can check *data* for whether or not it is null.

```
if (this.data ==null)
  this.data = QCSPCChartTS.DoubleArray.newDoubleArrayN(5);
else
  this.data.reset();
```

or, more commonly

```
if (!this.data)
  this.data = QCSPCChartTS.DoubleArray.newDoubleArrayN(5);
else
  this.data.reset();
```

By permitting a null value for the object *data*, the compiler will require you to do a null check before accessing it.

In our JavaScript example code, we often just grab a reference to the charts internal SPCControlChartData object using code like this:

[JavaScript]

```
xbarrchart.getChartData().setTitle (charttitle);
xbarrchart.getChartData().setChartDescriptior("XBar-R");
```

and use that reference to assign values to its properties. That's because we know that if xbarrchart is valid, then the internal SPCControlChartData object is initialized.

But when working with TypeScript in strict mode, this is not allowed. Because null is one of the possible return values for getChartData(), the compiler will not let you call a member function of the returned object, because getChartData() might return null. So instead, in our TypeScript examples, you will always see that we define a local variable of type SPCControlChartData | null, and then check the value of the local variable to make sure it is not null, before accessing member variables, as seen in the example below.

[TypeScript]

```
let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarrchart.getChartData();
if (chartdata) {
    chartdata.setTitle (charttitle);
    chartdata.setChartDescriptior("XBar-R");
}
```

The compiler actually sees that a null test is being made on the value of chartdata, and that it will not try and access the setTitle and setChartDescriptor methods if chartdata is null. This code will pass the compiler strict test.

Constructor and Function Overloading

Function and operator overloading is a capability many OOP languages have. Using it, you can give many different functions the same name. The compiler is able to differentiate which version of the function (or operator) you want using the types of arguments used in the function call. In short, every function of the same name must have a unique argument signature in order for this to work. It is a particularly useful feature of OOP languages. Unfortunately JavaScript and TypeScript don't support it. There are two main types of overloading which were originally used in the C# and Java versions of QCSPCChart: constructor overloading,and member method overloading. Both of these were translated for JavaScript and TypeScript using the following paradigm.

A class can only have one constructor. So the constructor for every class is just the default constructor, without parameters.

In order to mimic all of the parameterized, overloaded, constructors we used throughout the software, we use static functions within each class. These functions take parameter lists which duplicates the parameters of the original overloaded constructors and will return an initialized object of the proper class. All of these pseudo-overloaded constructors start with the word "new", followed by the class name, followed by some parameter descriptors which will make the name of the function unique when compared to the other static constructors in the same class. For example, we use the DoubleArray array class throughout the software. It is used to store an array of number values, with better runtime bounds checking, and better utilities, than the standard JavaScript Array class. Create a new instance of the DoubleArray class, and size it to length 10, in the following manner:

[JavaScript]
```
var da = new QCSPCChartTS.DoubleArray();
da.initDoubleArrayN(10);
```

[TypeScript]
```
let da: QCSPCChartTS.DoubleArray = new QCSPCChartTS.DoubleArray();
da.initDoubleArrayN (10);
```

But you can also call several static, pseudo-constructors which will return an initialized DoubleArray object. Note that since the static routines are found in the DoubleArray class, you must preface newDoubleArrayN with the class name, DoubleArray.

[JavaScript]
```
var da = QCSPCChartTS.DoubleArray.newDoubleArrayN(10);
```

[TypeScript]
```
let da: QCSPCChartTS.DoubleArray = QCSPCChartTS.DoubleArray.newDoubleArrayN(10);
```

Note: There is no space between new and DoubleArray in the example immediately above. The static function name is newDoubleArrayN, one word.

Another useful static pseudo-constructor in the DoubleArray class is:

```
public static newDoubleArrayArray(x: number[]): DoubleArray
```

This method creates a new DoubleArray which holds the data in the source array (x).

All of the Array classes (DoubleArray, TimeArray, BoolArray, StringArray) have similar methods. Our chart dataset classes (SimpleDataset, GroupDataset, TimeSimpleDataset, TimeGroupDataset, etc.) are built using the Array classes and they too have multiple pseudo-constructors to create an initialize an instance of each class.

This logic continues through to the creation of the SPC Chart objects, such as SPCBatchVariableControlChart. You can create the object using the default, empty

constructor, and then initialize it using the appropriate **initSPC...** method, or you can use one of the static pseudo-constructors to create and initialize the object in one call.

[JavaScript]

```
  var htmlcanvas = document.getElementById(canvasid);
    var spccharttype = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    var subgroupsize = 5;
    var numberpointsinview = 12;
    var charttitle = "XBar-R Example";

// You can initialize it this way
    var xbarrchart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
```

OR

this way.

```
var xbarrchart = new QCSPCChartTS.SPCBatchVariableControlChart();
xbarrchart.initSPCBatchVariableControlChartCanvasChartTypeSubgroupSize(htmlcanvas,
spccharttype, subgroupsize, numberpointsinview);
```

[TypeScript]

```
   let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
    let spccharttype: number  = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    let subgroupsize: number = 5;
    let numberpointsinview: number  = 12;
    let charttitle: string = "XBar-R Example";


let xbarrchart: QCSPCChartTS.SPCBatchVariableControlChart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
```

OR

this way.

```
let xbarrchart: QCSPCChartTS.SPCBatchVariableControlChart = new
QCSPCChartTS.SPCBatchVariableControlChart();

xbarrchart.initSPCBatchVariableControlChartCanvasChartTypeSubgroupSize(htmlcanvas,
spccharttype, subgroupsize, numberpointsinview);
```

Another way to replace some overloaded methods in TypeScript is to use optional parameters. You can read about their use on-line. The drawback of this technique is that the method code then needs to add explicit checks for every optional parameter to determine whether or not it is defined, and to branch to the appropriate code block as a

result. All of this checking for undefined parameters is prone to errors. In general we just use the brute force method and give every previously overloaded method and pseudo-constructor a unique name, in spite of the drawback that many of the new names are annoyingly long.

Chapter 5 - SPC Control Data and Alarm Classes

SPCControlChartData
SPCControlLimitAlarmArgs
SPCControlLimitRecord
SPCCalculatedValueRecrod
SPCSampledValueRecord
SPCGeneralizedTableDisplay

The *Variable* and *Attribute Control Chart* classes share common data and alarm classes. SPC control chart data is stored in the **SPCControlChartData** class. It holds the header information used to customize the chart table, the raw sample data used to prepare the chart, the calculated chart values used in the chart, and the SPC control limits. It contains array lists of **SPCSampledValueRecord**, **SPCControlLimitRecord** and **SPCCalculatedValueRecord** objects. The **SPCGeneralizedTableDisplay** class manages **ChartText** objects used to display data in the table portion of the SPC chart.

Class SPCControlChartData

ChartObj
|
+-- SPCControlChartData

The **SPCControlChartData** class is the core data storage object for all of SPC Control Chart classes. It holds all of the data plotted in the SPC chart. That includes the header information used to customize the chart table,

Header Information

Title: Variable Control Chart (X-Bar R)		Chart No.: 17
Part Name: Transmission Casing Bolt		Part No.: 283501
Operator:J. Fenamore	Machine: #11	Gage: #8645
Date: 7/04/2013	Units: 0.0001 inch	Zero Equals: zero

the raw sample data used in the SPC calculations,

Raw Sample Data

Time	17:07	17:22	17:37	17:52	18:07	18:22	18:37	18:52	19:07	19:22	19:37	19:52
Sample #0	30.8	35.4	31.0	28.6	33.0	32.7	36.0	29.3	36.8	31.7	34.7	35.9
Sample #1	35.6	30.9	28.9	36.4	31.8	32.6	32.5	33.7	37.1	31.3	29.2	35.8
Sample #2	30.8	30.9	28.7	30.4	33.8	33.9	35.9	29.1	32.0	29.8	30.9	34.3
Sample #3	36.9	37.0	30.9	37.5	32.6	35.2	34.3	32.1	36.2	35.5	37.5	36.5
Sample #4	36.6	35.5	35.0	37.2	34.5	35.4	29.3	29.0	37.2	28.8	29.3	33.9

the calculated chart values used in the chart, and the SPC control limits,

Calculated Values

Average	34.1	33.9	30.9	34.0	33.2	34.0	33.6	30.7	35.9	31.4	32.3	35.3
RANGE	6.09	6.16	6.33	8.85	2.65	2.82	6.78	4.67	5.27	6.73	8.37	2.59
SUM	170.7	169.7	154.5	170.0	165.8	169.8	168.0	153.3	179.3	157.2	161.6	176.4
NO. INSP.	5	5	5	5	5	5	5	5	5	5	5	5

and any notes you might want to place in the record.

Status (Alarms and Notes)

Alarm	-	-	-	-	H	-	H	-	-	-	-	-	-	H	H	-	-	-	H	-	-	-	-
Notes	-N-		-N-		-Y-		-N-		-N-		-N-		-N-		-Y-		-N-		-N-		-N-		-N-

There is an instance of **SPCControlChartData** in the **SPCChartBase** class. Since the **SPCChartBase** class is the base class for all of the SPC Control Charts (**SPCEventAttributeControlChart**, **SPCEventVariableControlChart**, **SPCBatchAttributeConrolChart**, **SPCBatchVariableControlChart**, **SPCTimeAttributeConrolChart**, **SPCTimeVariableControlChart**), it is accessible from those classes. The data elements of the **SPCControlChartData** class are accessible to the programmer.

SPCControlChartData Methods

The **SPCControlChartData** object is automatically created when the parent **SPCChartBase** object is created. The programmer does not need to instantiate it.

Note - Since JavaScript/TypeScript does not support properties in the same way that C# does, you set and get the values of the properties using the “set” and “get” in front of the property name, i.e. **getPartNumber** and **setPartNumber.**

Public Static Fields

FRACTION_DEFECTIVE_PARTS_CHART	Chart type constant: Fraction Defective Parts (p-chart) Control Chart
HEADER_STRINGS_LEVEL0	SPC Chart header level constant: display no header strings.
HEADER_STRINGS_LEVEL1	SPC Chart header level constant:

	display minimal header strings, title, partNumber, chartNumber, dateString
HEADER_STRINGS_LEVEL2	SPC Chart header level constant: display most header strings, title, partNumber, chartNumber, partName, operation, operator, machine, dateString
HEADER_STRINGS_LEVEL3	SPC Chart header level constant: display all header strings, title, partNumber, chartNumber, partName, operation, operator, machine, specification limits, gage, unitofMeasure, zeroEqulas and dateString
INDIVIDUAL_RANGE_CHART	Chart type constant: Individual Range (Individual X) SPC Variable Control Chart
MEAN_RANGE_CHART	Chart type constant: Mean and Range (X-Bar R) SPC Variable Control Chart (
MEAN_SIGMA_CHART	Chart type constant: Mean and Sigma (X-Bar Sigma) SPC Variable Control Chart
MEAN_SIGMA_CHART_VSS	Chart type constant: Mean and Sigma (X-Bar Sigma) SPC Variable Control Chart with variable sample size
MEAN_VARIANCE_CHART	Chart type constant: Mean and Variance (X-Bar Variance) SPC Variable Control Chart
MEDIAN_RANGE_CHART	Chart type constant: Median and Range (Median-Range) SPC Variable Control Chart
NUMBER_DEFECTIVE_PARTS_CHART	Chart type constant: Number Defective Parts (np-chart) Control Chart
NUMBER_DEFECTS_CHART	Chart type constant: Number Defects (c-chart) Control Chart
NUMBER_DEFECTS_PERUNIT_CHART	Chart type constant: Number Defects per Unit (u-chart) Control Chart
PERCENT_DEFECTIVE_PARTS_CHART	Chart type constant: Percent Defective Parts (p-chart) Control Chart

SPC_PRIMARY_CONTROL_TARGET	Index of primary chart target control limit in controlLimitData array.
SPC_PRIMARY_LOWER_CONTROL_LIMIT	Index of primary chart lower control limit in controlLimitData array.
SPC_PRIMARY_UPPER_CONTROL_LIMIT	Index of primary chart upper control limit in controlLimitData array.
SPC_SECONDARY_CONTROL_TARGET	Index of secondary chart target control limit in controlLimitData array.
SPC_SECONDARY_LOWER_CONTROL_LIMIT	Index of secondary chart lower control limit in controlLimitData array.
SPC_SECONDARY_UPPER_CONTROL_LIMIT	Index of secondary chart upper control limit in controlLimitData array.

Public Instance Properties

BatchColumnHeadString	Default string used as the batch number column head in the log file. The default value is "Batch #"
NotesColumString	Default string used as the notes column head in the log file. The default value is "Notes"
SampleValueColumnString	Default string used as the sample value column head in the log file. The default value is "Sample #"
TimeStampColumnString	Default string used as the time stamp column head in the log file. The default value is "Time Stamp"
AlarmStateEventEnable	set/get to True to signify that any alarm should invoke the AlarmStateEventHandler.
AlarmTransitionEventEnable	set/get to True to signify that any change in an alarm state should invoke the AlarmTransitionEventHandler.
ChartNumber	set/get data table chart number string.
ChartNumberHeader	set/get the header for the chartNumber field.
CurrentNumberRecords	get the current number of records for the chart.
DateHeader	set/get the header for the dateString field.
DateString	set/get data table date string.
DefectiveDecimalPrecision	set/get the default value to use for the decimal precision used to display defective item counts, -1 = auto.
Gage	set/get data table gage string.
GageHeader	set/get the header for the gage field.
Machine	set/get data table machine string.
MachineHeader	set/get the header for the machine field.
NotesHeader	set/get the data table notes header string.
NotesMessage	set/get data table notes message string.

NotesToolTips	set/get the notes tooltip.
NumberSamplesValueRowHeader	set/get the data table number of samples row header.
NumCalculatedValues	set/get number of calculated values for each record in the chart.
NumRecordsPerChart	set/get the maximum number of records displayable at one time in the chart.
NumSampleCategories	set/get the number of categories in an Attribute Control chart. numSampleCategories == sampleSubgroupSize for Variable Control Charts
Operation	set/get data table operation string.
OperationHeader	set/get the header for the operation field.
OperatorHeader	set/get the header for the theOperator field.
PartName	set/get data table part name string.
PartNameHeader	set/get the header for the partName field.
PartNumber	set/get data table part number string.
PartNumberHeader	set/get the header for the partNumber field.
SampleSubgroupSize	set/get the number of samples in a sample sub group for a Variable Control chart.
SPCChartType	set/get the control chart type: use one of the SPCControlChartData chart type constants: MEAN_RANGE_CHART, MEDIAN_RANGE_CHART, MEAN_SIGMA_CHART, MEAN_SIGMA_CHART_VSS, MEAN_VARIANCE_CHART, INDIVIDUAL_RANGE_CHART, EWMA_CHART, MA_CHART, LEVEY_JENNINGS_CHART, MAMR_CHART, MAMS_CHART TABCUSUM_CHART, CUSTOM_ATTRIBUTE_CONTROL_CHART, PERCENT_DEFECTIVE_PARTS_CHART, FRACTION_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTS_CHART, NUMBER_DEFECTS_PERUNIT_CHART, NUMBER_DEFECTS_PER_MILLION_CHART.
SpecificationLimits	set/get data table specification limits string.
SpecificationLimitsHeader	set/get the header for the specificationLimits field.
TheOperator	set/get data table operator string.
TimeStamp	set/get the time stamp for the most recent sample data added to the class.
TimeValueRowHeader	The data table time value row header.
Title	set/get data table title string.

TitleHeader	set/get the header for the title field.
UnitOfMeasure	set/get data table unit of measure string.
UnitOfMeasureHeader	set/get the header for the unit of measure field.
ZeroEquals	set/get data table zero equals string.
ZeroEqualsHeader	set/get the header for the zeroEqulas field.

Public Instance Functions

addNewSampleRecord	Add a new sample record with notes to a time-based SPC chart that plots variable control limits.
excludeRecordFromControlLimitCalculations	Exclude the specified record from the SOC control limit calculations.
getBatchNumberValue	get the group number value at the specified index.
getCalculatedValue	Get a calculated value at a specific row (index) and column (time).
getCalculatedValueRecord	get the calculated value record at the specified index.
getControlLimit	get the value of a specific SPC chart limit.
getControlLimitRecord	get the control limit record at the specified index.
getControlLimitString	get the text for a specific SPC chart limit.
getControlLimitText	get the control limit text at the specified index.
getControlLimitValue	Get a control limit value (for charts with variable control limits) at a specific row (index) and column (time).
getNotesString	get the notes string at the specified index.
getNumberOfSamplesPerSubgroup	get the number of samples per subgroup value at the specified index.
getPrimaryControlLimits	Using the current sampled data values, high, target and low control limits are calculated for the primary chart using an algorithm appropriate to the SPC chart type.
getSampledValue	Get a sampled value at a specific row (index) and column (time).
getSampleRowHeaderString	Get data table row header for the sampled (or category) item.
getSecondaryControlLimits	Using the current sampled data values, high, target and low control limits are calculated for the primary chart using an algorithm appropriate to the SPC chart type.

getTimeValue	get the time stamp value at the specified index.
getYAxisTitle	get the y-axis title or a specific index, based description of the item in the SPCCalculatedValueRecord or SPCSampledValueRecord record.
resetSPCChartData	Reset the history buffers of all of the SPC data objects.
setControlLimitString	Set the SPC text for a specific SPC chart limit.
setControlLimitStrings	Set the SPC control limit text for an SPC control chart.
setControlLimitValue	Set the SPC value of a specific SPC chart limit.
setControlLimitValues	Set the SPC control limit values for an SPC control chart.
setSampleRowHeaderString	Set data table row header for the sampled (or category) item.
simulateDefectRecord	Simulates a defect measurement for a SPC Attribute Control chart with a specified mean. Used with NUMBER_DEFECTS_CHART and NUMBER_DEFECTS_PERUNIT_CHART charts.
simulateMeasurementRecord	Simulates a sample measurement for a SPC Variable Control chart with a specified mean value.
transitionEventCondition	Returns true if an alarm transition has taken place.

The SPCControlChartData properties are documented in the QCSPCChartJSTSClassesIndex.html documentation file, located in the docs/docs/ subdirectory.

Initializing the SPCControlChartData Class

The control charts constructor initializes the **SPCControlChartData** object. This establishes the SPC chart type, how many samples per subgroup there are, and how many **SPCSampledValueRecord** objects are stored internal to the **SPCControlChartData** to handle the sampled data.

The table strings used to customize the table section of the chart should be set after the chart in instantiated, but before the **rebuildChartUsingCurrentData** call. The example below is from the **VariableControlCharts.BuildXBarRChart** function.

Using JavaScript the programmer can access the ChartData methods and properties using the syntax:

[JavaScript]

```
    xbarrchart.getChartData().setTitle (charttitle);
    xbarrchart.getChartData().setChartDescriptior("XBar-R");
    xbarrchart.getChartData().setPartNumber ( "283501");
```

where you use the SPC charts **getChartData()** method, inline with the method with the ChartData method you want to call. This is allowed because JavaScript doesn't check for possible nulls or undefined objects in method calls. But, if you want to be extra safe, you can use safer syntax such as:

```
    var chartdata = xbarrchart getChartData();
    if (chartdata) {
      chartdata.setTitle (charttitle);
      chartdata.setChartDescriptior("XBar-R");
      chartdata.setPartNumber ( "283501");
     }
```

If you use TypeScript in the “strict” compile mode, you must use the following syntax.

[TypeScript]

```
   let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarrchart.getChartData();
    if (chartdata) {
      chartdata.setTitle (charttitle);
      chartdata.setChartDescriptior("XBar-R");
      chartdata.setPartNumber ( "283501");
     }
```

Below are more complicated examples.

[JavaScript]

```
import * as QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';

export async function BuildXBarRChart(canvasid) {

    var htmlcanvas = document.getElementById(canvasid);
    var spccharttype = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    var subgroupsize = 5;
    var numberpointsinview = 12;
    var charttitle = "XBar-R Example";


    var xbarrchart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);

    xbarrchart.setPreferredSize(800, 600);
```

```
    xbarrchart.setGraphStopPosX(0.825);
    xbarrchart.setGraphStartPosX(0.18);

xbarrchart.setChartAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_SYM
BOL);
    xbarrchart.setEnableScrollBar(true);

    xbarrchart.setEnableCategoryValues(true);
    xbarrchart.setEnableCalculatedValues(true);
    xbarrchart.setEnableAlarmStatusValues(true);
    xbarrchart.setEnableChartToggles(true);

xbarrchart.setTableAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_BAR
);

xbarrchart.setHeaderStringsLevel( QCSPCChartTS.SPCControlChartData.HEADER_STRINGS_
LEVEL1);

    xbarrchart.getChartData().setTitle (charttitle);
    xbarrchart.getChartData().setChartDescriptior("XBar-R");
    xbarrchart.getChartData().setPartNumber ( "283501");
    xbarrchart.getChartData().setChartNumber("17");
    xbarrchart.getChartData().setPartName( "Transmission Casing Bolt");
    xbarrchart.getChartData().setOperation ( "Threading");
    xbarrchart.getChartData().setOperator("J. Fenamore");
    xbarrchart.getChartData().setMachine("#11");
    var  today = new Date();
    xbarrchart.getChartData().setDateString(today.toLocaleString());
    xbarrchart.getChartData().setNotesMessage( "Control limits prepared May 10");
    xbarrchart.getChartData().setNotesHeader("NOTES"); // row header


    xbarrchart.setEnableDisplayOptionToggles(true);

.
.
.

    // Rebuild the chart using the current data and settings
    xbarrchart.rebuildChartUsingCurrentData();

}
```

[TypeScript]

```
import * as QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';


public async  BuildXBarRChart(canvasid: string) {

    let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
    let spccharttype: number  = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    let subgroupsize: number = 5;
    let numberpointsinview: number  = 12;
    let charttitle: string = "XBar-R Example";


    let xbarrchart: QCSPCChartTS.SPCChartBase =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);

    xbarrchart.setPreferredSize(800, 600);

    xbarrchart.setGraphStopPosX(0.825);
    xbarrchart.setGraphStartPosX(0.18);

xbarrchart.setChartAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_SYM
BOL);
    xbarrchart.setEnableScrollBar(true);
```

```
    xbarrchart.setEnableCategoryValues(true);
    xbarrchart.setEnableCalculatedValues(true);
    xbarrchart.setEnableAlarmStatusValues(false);
    xbarrchart.setEnableChartToggles(true);

xbarrchart.setTableAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_BAR
);

xbarrchart.setHeaderStringsLevel( QCSPCChartTS.SPCControlChartData.HEADER_STRINGS_
LEVEL1);

    xbarrchart.setEnableDisplayOptionToggles(true);

    let chartdata: QCSPCChartTS.SPCControlChartData | null = xbarrchart
getChartData();
    if (chartdata) {
      chartdata.setTitle (charttitle);
      chartdata.setChartDescriptior("XBar-R");
      chartdata.setPartNumber ( "283501");
      chartdata.setChartNumber("17");
      chartdata.setPartName( "Transmission Casing Bolt");
      chartdata.setOperation ( "Threading");
      chartdata.setOperator("J. Fenamore");
      chartdata.setMachine("#11");
      let  today: Date = new Date();
      chartdata.setDateString(today.toLocaleString());
      chartdata.setNotesMessage( "Control limits prepared May 10");
      chartdata.setNotesHeader("NOTES"); // row header
    }

.
.
.

    // Rebuild the chart using the current data and settings
    xbarrchart.rebuildChartUsingCurrentData();

}
```

Update the sampled data with your measured values using one of the **addNewSampleRecord**... methods. Since function overloads are not permitted using either JavaScript or TypeScript, there are many different versions of **addNewSampleRecord** to accommodate different parameter lists.

Method addNewSampleRecord

This method adds a new sample record to the SPC chart. While *both variable control charts* and *attribute control charts* share the same **ChartData addNewSampleRecord** methods, the meaning of the data in the *samples* array varies, depending on the chart type. See the sections below: *Adding New Sample Records for Variable Control Charts*, and *Adding New Sample Records for Attribute Control Charts*.

[TypeScript]

```
    addNewSampleRecordDateSamplesControlLimitsNotes(timestamp: Date, samples: DoubleArray,
controllimits: DoubleArray, notes: string): number;
```

```
    addNewSampleRecordDateSamplesControlLimits(timestamp: Date, samples: DoubleArray,
controllimits: DoubleArray): number;

    addNewSampleRecordDateSamplesNotes(timestamp: Date, samples: DoubleArray, notes:
string): number;

    addNewSampleRecordDateSamples(timestamp: Date, samples: DoubleArray): number;

    addNewSampleRecordBatchNumberSamplesControlLimitsNotes(batchnumber: number, samples:
DoubleArray, controllimits: DoubleArray, notes: string): number;

    addNewSampleRecordBatchNumberSamplesControlLimits(batchnumber: number, samples:
DoubleArray, controllimits: DoubleArray): number;

    addNewSampleRecordBatchNumberSamplesNotes(batchnumber: number, samples: DoubleArray,
notes: string): number;

    addNewSampleRecordBatchNumberSamples(batchnumber: number, samples: DoubleArray):
number;

    addNewSampleRecordSamplesNotes(samples: DoubleArray, notes: string): number;

    addNewSampleRecordSamples(samples: DoubleArray): number;

    addNewSampleRecordBatchNumberDateSamplesControlLimitsNotes(batchnumber: number,
timestamp: Date, samples: DoubleArray, controllimits: DoubleArray, notes: string): number;

    addNewSampleRecordBatchNumberDateSamplesControlLimits(batchnumber: number, timestamp:
Date, samples: DoubleArray, controllimits: DoubleArray): number;

    addNewSampleRecordBatchNumberDateSamplesNotes(batchnumber: number, timestamp: Date,
samples: DoubleArray, notes: string): number;

    addNewSampleRecordBatchNumberDateSamples(batchnumber: number, timestamp: Date,
samples: DoubleArray): number;
```

[JavaScript]

```
The TypeScript prototypes above should serve as your guide when calling the
addNewSampleRecord... methods from JavaScript. Since JavaScript has no type checking, make
sure you call place the parameters in the correct position, depending on the particular
method you are calling.
```

Parameters

batchnumber

A batch number associated with the sample interval.

timestamp

Time stamp for the current sample record. Some version use a number while others use a Date object.

samples

Array of new sample values.

controllimits

Array of control limits, one for each control limits (low, target, and high)

notes

A string specifying any notes associated with this sample subgroup

Adding New Sample Records for Variable Control Charts (Fixed Subgroup Sample Size).

Applies to variable control charts of type: MEAN_RANGE_CHART, MEDIAN_RANGE_CHART, INDIVIDUAL_RANGE_CHART, MEAN_SIGMA_CHART, INDIVIDUAL_RANGE_CHART, EWMA_CHART, LEVEY_JENNINGS_CHART, MA_CHART, MAMR_CHART, MAMS_CHART, TABCUSUM_CHART.

In variable control charts, each data value in the *samples* array represents a specific sample in the sample subgroup. In X-Bar R, X-Bar Sigma, and Median-Range charts, where the sample subgroup size is some fraction of the total production level, there is one value in the *samples* array for each measurement sample in the sample subgroup interval. If the production level is sixty items per hour, and the sample size is five items per hour, then the graph would be updated once an hour with five items in the *samples* array.

[JavaScript]

```
import * as QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
.
.
.


var chartdata = spcchart.getChartData();
if (!chartdata) return;

var samples = QCSPCChartTS.DoubleArray.newDoubleArrayN(5);

var timestamp = new Date(); // use a unique time stamp for every update
batchnum++; // defined elsewhere but incremented with every update

// Place sample values in array
samples.setElement(0, 0.121);      // First of five samples
samples.setElement(1, 0.212);      // Second of five samples
samples.setElement(2, 0.322);      // Third of five samples
samples.setElement(3, 0.021);      // Fourth of five samples
samples.setElement(4, 0.133);      // Fifth of five samples

// Add the new sample subgroup to the chart
chartdata.addNewSampleRecordBatchNumberDateSamples(batchnum,timestamp, samples);
```

[TypeScript]

```
import * as QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
.
.
.


let chartdata: QCSPCChartTS.SPCControlChartData | null = spcchart.getChartData();
if (!chartdata) return;

let samples: QCSPCChartTS.DoubleArray=
QCSPCChartTS.DoubleArray.newDoubleArrayN(5);

let timestamp: Date = new Date(); // use a unique time stamp for every update
```

```
batchnum++; // defined elsewhere but incremented with every update
// Place sample values in array
samples.setElement(0, 0.121);      // First of five samples
samples.setElement(1, 0.212);      // Second of five samples
samples.setElement(2, 0.322);      // Third of five samples
samples.setElement(3, 0.021);      // Fourth of five samples
samples.setElement(4, 0.133);      // Fifth of five samples

// Add the new sample subgroup to the chart
chartdata.addNewSampleRecordBatchNumberDateSamples(batchnum,timestamp, samples);
```

In an Individual-Range chart, which by definition samples 100% of the production level, the *samples* array would only have one value for each update. If the production level is sixty items per hour, with 100% sampling, the graph would be updated once a minute, with a single value in the *samples* array.

Adding New Sample Records to a X-Bar Sigma Chart (Variable Subgroup Sample Size)

Applies to variable control charts of type: MEAN_SIGMA_CHART_VSS

The X-Bar Sigma chart also comes in a version where variable sample sizes are permitted, As in the standard variable control charts, there is one value in the *samples* array for each measurement sample in the sample subgroup interval. The difference is that the length of the *samples* array can change from update to update. It is critically important that the size of the samples array exactly matches the number of samples in the current subgroup

[JavaScript]

```
// getCurrentSampleSubgroupSize is a fictional method that gets the
// current number of samples in the  sample subgroup. The value of N
// can vary from sample interval to sample interval. You must have a
// valid sample value for each element.

import * as QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
.
.
.
   var htmlcanvas = document.getElementById(canvasid);
        var spccharttype = QCSPCChartTS.SPCControlChartData.MEAN_SIGMA_CHART_VSS;
        var subgroupsize = 15;
        var numberpointsinview = 12;
        var charttitle = "XBar-Sigma Example";

        var xbarsigmachart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);



.
.
.

var  chartdata = xbarsigmachart.getChartData();
if (!chartdata) return;

var N = this.getCurrentSampleSubgroupSize()
```

```
var samples = QCSPCChartTS.DoubleArray.newDoubleArrayN(N);

var timestamp = new Date();// use a unique time stamp for every update
batchnum++; // defined elsewhere but incremented with every update

// Place sample values in array
// You must have a valid sample value for each element of the array size 0..N-1
// Place sample values in array
// You must have a valid sample value for each element of the array size 0..N-1
samples.setElement(0, 32.121);     // First of  samples
samples.setElement(1, 31.212);     // Second of  samples
.
.
.
samples.setElement(N-1, 30.133); // Last of the samples in the sample subgroup

// Add the new sample subgroup to the chart
chartdata.addNewSampleRecordBatchNumberDateSamples(batchnum,timestamp, samples);
```

[TypeScript]

```
// getCurrentSampleSubgroupSize is a fictional method that gets the
// current number of samples in the  sample subgroup. The value of N
// can vary from sample interval to sample interval. You must have a
// valid sample value for each element.

import * as QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
.
.
.
   let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
        let spccharttype: number  =
QCSPCChartTS.SPCControlChartData.MEAN_SIGMA_CHART_VSS;
        let subgroupsize: number = 15;
        let numberpointsinview: number  = 12;
        let charttitle: string = "XBar-Sigma Example";

        let xbarsigmachart: QCSPCChartTS.SPCBatchVariableControlChart=
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
.
.


let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarsigmachart.getChartData();
if (!chartdata) return;

let N: number = this.getCurrentSampleSubgroupSize()

let samples: QCSPCChartTS.DoubleArray=
QCSPCChartTS.DoubleArray.newDoubleArrayN(N);

let timestamp: Date = new Date();// use a unique time stamp for every update
batchnum++; // defined elsewhere but incremented with every update



// Place sample values in array
// You must have a valid sample value for each element of the array size 0..N-1
// Place sample values in array
// You must have a valid sample value for each element of the array size 0..N-1
samples.setElement(0, 32.121);     // First of  samples
```

```
samples.setElement(1, 31.212);     // Second of  samples
.
.
.
samples.setElement(N-1, 30.133); // Last of the samples in the sample subgroup

// Add the new sample subgroup to the chart
chartdata.addNewSampleRecordBatchNumberDateSamples(batchnum,timestamp, samples);
```

Adding New Sample Records for Attribute Control

Updating p- and np-charts (Fixed Sample Subgroup Size)

p-chart = FRACTION_DEFECTIVE_PARTS_CHART
or
PERCENT_DEFECTIVE_PARTS_CHART

np-chart = NUMBER_DEFECTIVE_PARTS_CHART

In attribute control charts, the meaning of the data in the *samples* array varies, depending on whether the attribute control chart measures the number of defective parts (p-, and np-charts), or the total number of defects (u- and c-charts). The major anomaly is that while the p- and np-charts plot the fraction or number of defective parts, the table portion of the chart can display defect counts for any number of defect categories (i.e. paint scratches, dents, burrs, etc.). It is critical to understand that total number of defects, i.e. the sum of the items in the defect categories for a give sample subgroup, do NOT have to add up to the number of defective parts for the sample subgroup. Every defective part not only can have one or more defects, it can have multiple defects of the same defect category. The total number of defects for a sample subgroup will always be equal to or greater than the number of defective parts. When using p- and np-charts that display defect category counts as part of the table, where N is the *numcategories* parameter in the **initSPCBatchAttributeControlChart** initialization call, the first N (0.. N-1) elements of the *samples* array holds the defect count for each category. The (N+1)th (or element N in the array) element of the *samples* array holds the total defective parts count. For example, if you initialized the chart with a *numcategories* parameter to five, signifying that you had five defect categories, you would use a *samples* array sized to six, as in the code below:

[JavaScript]

```
import * as QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
.
.
.
    var htmlcanvas = document.getElementById(canvasid);
        var spccharttype =
QCSPCChartTS.SPCControlChartData.FRACTION_DEFECTIVE_PARTS_CHART;
        var subgroupsize = 100;
        var numberpointsinview = 12;
        var charttitle = " p-Chart (Fraction)";
```

```
        var attribchart =
QCSPCChartTS.SPCBatchAttributeControlChart.newSPCBatchAttributeControlChartChartTy
peSubgroupSize(htmlcanvas, spccharttype, 1, subgroupsize, numberpointsinview);
.
.
.

var  chartdata = attribchart.getChartData();
if (!chartdata) return;

var  samples = QCSPCChartTS.DoubleArray.newDoubleArrayN(6);

var timestamp = new Date();// use a unique time stamp for every update
batchnum++; // defined elsewhere but incremented with every update
// Place sample values in array
samples.setElement(0, 3);   // Number of defects for defect category #1
samples.setElement(1, 0);       // Number of defects for defect category #2
samples.setElement(2, 4);       // Number of defects for defect category #3
samples.setElement(3, 2);       // Number of defects for defect category #4
samples.setElement(4, 3);       // Number of defects for defect category #5

samples.setElement(5, 4);      // TOTAL number of defective parts in the sample

// Add the new sample subgroup to the chart
chartdata.addNewSampleRecordBatchNumberDateSamples(batchnum,timestamp, samples);
```

[TypeScript]

```
import * as QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
.
.
.
    let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
        let spccharttype: number =
QCSPCChartTS.SPCControlChartData.FRACTION_DEFECTIVE_PARTS_CHART;
        let subgroupsize: number = 100;
        let numberpointsinview: number = 12;
        let charttitle: string = " p-Chart (Fraction)";


        let attribchart: QCSPCChartTS.SPCChartBase =
QCSPCChartTS.SPCBatchAttributeControlChart.newSPCBatchAttributeControlChartChartTy
peSubgroupSize(htmlcanvas, spccharttype, 1, subgroupsize, numberpointsinview);
.
.
.

let chartdata: QCSPCChartTS.SPCControlChartData | null =
attribchart.getChartData();
if (!chartdata) return;

let samples: QCSPCChartTS.DoubleArray=
QCSPCChartTS.DoubleArray.newDoubleArrayN(6);

let timestamp: Date = new Date();// use a unique time stamp for every update
batchnum++; // defined elsewhere but incremented with every update
// Place sample values in array
samples.setElement(0, 3);   // Number of defects for defect category #1
samples.setElement(1, 0);       // Number of defects for defect category #2
samples.setElement(2, 4);       // Number of defects for defect category #3
samples.setElement(3, 2);       // Number of defects for defect category #4
samples.setElement(4, 3);       // Number of defects for defect category #5

samples.setElement(5, 4);      // TOTAL number of defective parts in the sample

// Add the new sample subgroup to the chart
```

```
chartdata.addNewSampleRecordBatchNumberDateSamples(batchnum,timestamp, samples);
```

Our example programs obscure this a bit, because we use a special method to simulate defect data for n- and np-charts. The code below is extracted from our AttributeControlCharts example, function SimulateData.

[JavaScript]

```
var  chartdata = attribchart.getChartData();
if (!chartdata) return;

var samples = chartdata.simulateMeasurementRecordMeanRange(mean, sigma);
// Update chart data using i as the batch number
batchCounter = currentcount + i;
var note = "This is a note";
// Add a new sample record to the chart data
chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter, timestamp,
samples, note);
```

[TypeScript]

```
let chartdata: QCSPCChartTS.SPCControlChartData | null =
attribchart.getChartData();
if (!chartdata) return;

let samples: QCSPCChartTS.DoubleArray =
chartdata.simulateMeasurementRecordMeanRange(mean, sigma);
// Update chart data using i as the batch number
batchCounter = currentcount + i;
let note: string = "This is a note";

// Add a new sample record to the chart data
chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter, timestamp,
samples, note);
```

This particular overload for **chartdata.simulateDefectRecord** knows that since it is a NUMBER_DEFECTIVE_PARTS_CHART chart (np-chart), and since the **ChartData** object was setup with five categories in the **initSPCTimeAttributeControlChart** call, that is should return a **DoubleArray** with (5 + 1 = 6) elements. The first five elements representing simulated defect counts for the five defect categories, and the sixth element the simulated defective parts count. The defect category count data of the *samples* array is only used in the table part of the display; the defect category counts play NO role in the actual SPC chart. The only value plotted in the SPC chart is the last element in the *samples* array, the defective parts count for the sample subgroup.

Updating p-charts (Variable Sample Subgroup Size)

p-chart = FRACTION_DEFECTIVE_PARTS_CHART_VSS
or
PERCENT_DEFECTIVE_PARTS_CHART_VSS

First, you must read the previous section (**Updating p-charts (Fixed Sample Subgroup Size**) and understand it**.** Because in the case of the p-chart variable sample subgroup case, filling out that array is EXACTLY the same as the fixed sample subgroup case. The number of defects in each defect category go into the first N elements (element 0..N-1) of the samples array. The total number of defective parts go into last (element N) of the samples array. Specify the size of the sample subgroup associated with a given update using the **chartdata.SampleSubgroupSize_VSS** property.

[JavaScript]

```
import * as QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
.
.
.
   var htmlcanvas = document.getElementById(canvasid);
        var spccharttype =
QCSPCChartTS.SPCControlChartData.FRACTION_DEFECTIVE_PARTS_CHART_VSS;
        var subgroupsize = 100;
        var numberpointsinview = 12;
        var charttitle = " p-Chart (Fraction)";


        var attribchart =
QCSPCChartTS.SPCBatchAttributeControlChart.newSPCBatchAttributeControlChartChartTy
peSubgroupSize(htmlcanvas, spccharttype, 1, subgroupsize, numberpointsinview);
.
.
.

var chartdata = attribchart.getChartData();
var  (!chartdata) return;

var  samples = QCSPCChartTS.DoubleArray.newDoubleArrayN(6);

var  timestamp = new Date(); // use a unique time stamp for each update

batchnum++; / defined elsewhere but incremented with every update

// Place sample values in array
samples.setElement(0, 3);  // Number of defects for defect category #1
samples.setElement(1, 0);      // Number of defects for defect category #2
samples.setElement(2, 4);      // Number of defects for defect category #3
samples.setElement(3, 2);      // Number of defects for defect category #4
samples.setElement(4, 3);      // Number of defects for defect category #5

samples.setElement(5, 4);     // TOTAL number of defective parts in the sample


// In the case of Attribute Control Charts, the number of samples is set
// initially using  the InitSPCTimeAttributeControlChart method, done
// in InitializeChart above. Unlike Variable Controls Charts, the number
// of samples is not related to the Length of the samples update array.
// Instead you must EXPLICITLY set the sample subgroup size, before each
// addNewSampleRecord, using the charts ChartData.SampleSubgroupSize_VSS
// property, as below.


var randnum = QCSPCChartTS.ChartSupport.getRandomDouble();
var numsamples = subgroupsize - (50 * randnum);
chartdata.setSampleSubgroupSize_VSS(numsamples);


// Add the new sample subgroup to the chart
chartdata.addNewSampleRecordBatchNumberDateSamples(batchnum,timestamp, samples);
```

[TypeScript]

```
import * as QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
.
.
.
    let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
        let spccharttype: number =
QCSPCChartTS.SPCControlChartData.FRACTION_DEFECTIVE_PARTS_CHART_VSS;
        let subgroupsize: number = 100;
        let numberpointsinview: number = 12;
        let charttitle: string = " p-Chart (Fraction)";


        let attribchart: QCSPCChartTS.SPCChartBase =
QCSPCChartTS.SPCBatchAttributeControlChart.newSPCBatchAttributeControlChartChartTy
peSubgroupSize(htmlcanvas, spccharttype, 1, subgroupsize, numberpointsinview);
.
.
.

let chartdata: QCSPCChartTS.SPCControlChartData | null =
attribchart.getChartData();
if (!chartdata) return;

let samples: QCSPCChartTS.DoubleArray=
QCSPCChartTS.DoubleArray.newDoubleArrayN(6);

let timestamp: Date = new Date(); // use a unique time stamp for every update
batchnum++; / defined elsewhere but incremented with every update

// Place sample values in array
samples.setElement(0, 3);   // Number of defects for defect category #1
samples.setElement(1, 0);       // Number of defects for defect category #2
samples.setElement(2, 4);       // Number of defects for defect category #3
samples.setElement(3, 2);       // Number of defects for defect category #4
samples.setElement(4, 3);       // Number of defects for defect category #5

samples.setElement(5, 4);      // TOTAL number of defective parts in the sample

// In the case of Attribute Control Charts, the number of samples is set
// initially using  the InitSPCTimeAttributeControlChart method, done
// in InitializeChart above. Unlike Variable Controls Charts, the number
// of samples is not related to the Length of the samples update array.
// Instead you must EXPLICITLY set the sample subgroup size, before each
// addNewSampleRecord, using the charts ChartData.SampleSubgroupSize_VSS
// property, as below.

let randnum: number = QCSPCChartTS.ChartSupport.getRandomDouble();
let numsamples: number =   subgroupsize - (50 * randnum);
chartdata.setSampleSubgroupSize_VSS(numsamples);



// Add the new sample subgroup to the chart
chartdata.addNewSampleRecordBatchNumberDateSamples(batchnum,timestamp, samples);
```

Updating c- and u-charts (Fixed Sample Subgroup Size)

c-chart = NUMBER_DEFECTS_CHART

u-chart = NUMBER_DEFECTS_PERUNIT_CHART

In c- and u-charts the number of defective parts is of no consequence. The only thing tracked is the number of defects. Therefore, there is no extra array element tacked onto the end of the *samples* array. Each element of the *samples* array represents the total number of defects for a given defect category. If the *numcategories* parameter in the **initSPCBatchAttributeControlChart** is initialized to five, the total number of elements in the *samples* array should be five. For example:

[JavaScript]

```
import * as QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
.
.
.


var chartdata = attribchart.getChartData();
if (!chartdata) return;

var  samples = QCSPCChartTS.DoubleArray.newDoubleArrayN(5);

var  timestamp = new Date();// use a unique time stamp for every update
batchnum++; // defined elsewhere but incremented with every update

// Place sample values in array
samples.setElement(0, 3);   ' Number of defects for defect category #1
samples.setElement(1, 0);       ' Number of defects for defect category #2
samples.setElement(2, 4);       ' Number of defects for defect category #3
samples.setElement(3, 2);       ' Number of defects for defect category #4
samples.setElement(4, 3);       ' Number of defects for defect category #5

// Add the new sample subgroup to the chart
chartdata.addNewSampleRecordBatchNumberDateSamples(batchnum,timestamp, samples);
```

[TypeScript]

```
import * as QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
.
.
.


let chartdata: QCSPCChartTS.SPCControlChartData | null =
attribchart.getChartData();
if (!chartdata) return;

let samples: QCSPCChartTS.DoubleArray=
QCSPCChartTS.DoubleArray.newDoubleArrayN(5);

let timestamp: Date = new Date(); // use a unique time stamp for every update
batchnum++; / defined elsewhere but incremented with every update
// Place sample values in array
samples.setElement(0, 3);   ' Number of defects for defect category #1
samples.setElement(1, 0);       ' Number of defects for defect category #2
samples.setElement(2, 4);       ' Number of defects for defect category #3
samples.setElement(3, 2);       ' Number of defects for defect category #4
samples.setElement(4, 3);       ' Number of defects for defect category #5

// Add the new sample subgroup to the chart
chartdata.addNewSampleRecordBatchNumberDateSamples(batchnum,timestamp, samples);
```

Updating u-charts (Variable Sample Subgroup Size)

u-chart = NUMBER_DEFECTS_PERUNIT_CHART_VSS

First, you must read the previous section (**Updating u-charts Fixed Sample Subgroup Size**) and understand it**.** Because in the case of the u-chart variable sample subgroup case, filling out that array is EXACTLY the same as the fixed sample subgroup case. The number of defects in each defect category go into the first N elements (element 0..N-1) of the samples array. Specify the size of the sample subgroup associated with a given update using the **chartdata.SampleSubgroupSize_VSS** property.

[JavaScript]

```
import * as QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
.
.
.


var chartdata = attribchart.getChartData();
if (!chartdata) return;

var  samples = QCSPCChartTS.DoubleArray.newDoubleArrayN(5);

var  timestamp = new Date();// use a unique time stamp for every update
batchnum++; // defined elsewhere but incremented with every update

// Place sample values in array
samples.setElement(0, 3);  // Number of defects for defect category #1
samples.setElement(1, 0);      // Number of defects for defect category #2
samples.setElement(2, 4);      // Number of defects for defect category #3
samples.setElement(3, 2);      // Number of defects for defect category #4
samples.setElement(4, 3);      // Number of defects for defect category #5

// In the case of Attribute Control Charts, the number of samples is set
// initially using  the InitSPCTimeAttributeControlChart method, done
// in InitializeChart above. Unlike Variable Controls Charts, the number
// of samples is not related to the Length of the samples update array.
// Instead you must EXPLICITLY set the sample subgroup size, before each
// addNewSampleRecord, using the charts ChartData.SampleSubgroupSize_VSS
// property, as below.


var randnum = QCSPCChartTS.ChartSupport.getRandomDouble();
var numsamples = subgroupsize - (50 * randnum);
chartdata.setSampleSubgroupSize_VSS(numsamples);


// Add the new sample subgroup to the chart
chartdata.addNewSampleRecordBatchNumberDateSamples(batchnum,timestamp, samples);
```

[TypeScript]

```
import * as QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
.
.
.


let chartdata: QCSPCChartTS.SPCControlChartData | null =
attribchart.getChartData();
```

```
if (!chartdata) return;

let samples: QCSPCChartTS.DoubleArray=
QCSPCChartTS.DoubleArray.newDoubleArrayN(5);

let timestamp: Date = new Date(); // use a unique time stamp for every update
batchnum++; / defined elsewhere but incremented with every update
// Place sample values in array
samples.setElement(0, 3);  // Number of defects for defect category #1
samples.setElement(1, 0);      //Number of defects for defect category #2
samples.setElement(2, 4);      // Number of defects for defect category #3
samples.setElement(3, 2);      // Number of defects for defect category #4
samples.setElement(4, 3);      // Number of defects for defect category #5

// In the case of Attribute Control Charts, the number of samples is set
// initially using  the InitSPCTimeAttributeControlChart method, done
// in InitializeChart above. Unlike Variable Controls Charts, the number
// of samples is not related to the Length of the samples update array.
// Instead you must EXPLICITLY set the sample subgroup size, before each
// addNewSampleRecord, using the charts ChartData.SampleSubgroupSize_VSS
// property, as below.

let randnum: number = QCSPCChartTS.ChartSupport.getRandomDouble();
let numsamples: number =   subgroupsize - (50 * randnum);
chartdata.setSampleSubgroupSize_VSS(numsamples);


// Add the new sample subgroup to the chart
chartdata.addNewSampleRecordBatchNumberDateSamples(batchnum,timestamp, samples);
```

While the table portion of the display can display defect data broken down into categories, only the sum of the defects for a given sample subgroup is used in creating the actual SPC chart.

[JavaScript]

```
// Simulate sample record
var samples = chartdata.simulateDefectRecord(19.85/5);
// Add a sample record
chartdata.addNewSampleRecordDateSamples( timestamp, samples);
```

[TypeScript]

```
// Simulate sample record
let samples: QCSPCChartTS.DoubleArray = chartdata.simulateDefectRecord(19.85/5);
// Add a sample record
chartdata.addNewSampleRecordDateSamples( timestamp, samples);
```

In addition to these, there are versions that pass in an additional **DoubleArray** that pass in the current value of variable control limits, if used.

If the **addNewSampleRecord** overload does not have an explicit **Date** time stamp parameter, as in the case several of the addNewSampleRecord methods, the current time as stored in the system clock is used as the time stamp.

The sampled values initialize the chart after the SPC chart class is instantiated, but before the **rebuildChartUsingCurrentData** call. The example below is from the **VariableControlCharts** example. The **addNewSampleRecord** routine is called in the **SimulateData** method.

[JavaScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';

export async function BuildXBarRChart(canvasid) {

    var htmlcanvas = document.getElementById(canvasid);
    var spccharttype = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    var subgroupsize = 5;
    var numberpointsinview = 12;
    var charttitle = "XBar-R Example";


    var xbarrchart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);

    xbarrchart.setPreferredSize(800, 600);

    xbarrchart.setGraphStopPosX(0.825);
    xbarrchart.setGraphStartPosX(0.18);
.
.
.

    var numssampleintervals = 100;
    var chartmean = 30;
    var chartsigma = 5;

    SimulateData(xbarrchart, numssampleintervals, chartmean, chartsigma);


    // Calculate the SPC control limits for both graphs of the current SPC chart
(X-Bar R)
    xbarrchart.autoCalculateControlLimits();

    // Out some more data, different mean and sigma, to simulate out of control
    numssampleintervals = 50;
    chartmean = 32;
    chartsigma = 8;

    SimulateData(xbarrchart, numssampleintervals, chartmean, chartsigma);

    // Scale the y-axis of the X-Bar chart to display all data and control limits
    xbarrchart.autoScalePrimaryChartYRange();
    // Scale the y-axis of the Range chart to display all data and control limits
    xbarrchart.autoScaleSecondaryChartYRange();
    // Rebuild the chart using the current data and settings
    xbarrchart.rebuildChartUsingCurrentData();

}

function SimulateData(spcchart, count, mean, sigma) {
    // batch number for a given sample subgroup
    var batchCounter = 0;
    var i;
    var timestamp = new Date();
    if (!spcchart) return;
    if (!spcchart.getChartData()) return;
    var currentcount = spcchart.getChartData().getCurrentNumberRecords();
    for (i = 0; i < count; i++) {
        // Simulate a sample subgroup record
        var samples =
spcchart.getChartData().simulateMeasurementRecordMeanRange(mean, sigma);
        // Update chart data using i as the batch number
        batchCounter = currentcount + i;
        var note = "";
        if ((i % 5) == 0) note = "This is a note";
        // Add a new sample record to the chart data
```

```
spcchart.getChartData().addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter
, timestamp, samples, note);
        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }

}
```

[TypeScript]

```
import * as QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';


public async  BuildXBarRChart(canvasid: string) {

    let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
    let spccharttype: number  = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    let subgroupsize: number = 5;
    let numberpointsinview: number  = 12;
    let charttitle: string = "XBar-R Example";


    let xbarrchart: QCSPCChartTS.SPCChartBase =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);

    xbarrchart.setPreferredSize(800, 600);

    xbarrchart.setGraphStopPosX(0.825);
    xbarrchart.setGraphStartPosX(0.18);
.
.
.


    let numssampleintervals: number = 100;
    let chartmean: number = 30;
    let chartsigma: number = 5;

    this.SimulateData(xbarrchart, numssampleintervals, chartmean, chartsigma);


    // Calculate the SPC control limits for both graphs of the current SPC chart
(X-Bar R)
    xbarrchart.autoCalculateControlLimits();

    numssampleintervals = 50;
    chartmean = 32;
    chartsigma = 8;

    this.SimulateData(xbarrchart, numssampleintervals, chartmean, chartsigma);

    // Scale the y-axis of the X-Bar chart to display all data and control limits
    xbarrchart.autoScalePrimaryChartYRange();
    // Scale the y-axis of the Range chart to display all data and control limits
    xbarrchart.autoScaleSecondaryChartYRange();
    // Rebuild the chart using the current data and settings
    xbarrchart.rebuildChartUsingCurrentData();

}


SimulateData(spcchart: QCSPCChartTS.SPCChartBase, count: number, mean: number,
sigma: number)  {
    // batch number for a given sample subgroup
```

```
    let batchCounter: number = 0;
        let i: number = 0;
        let timestamp: Date = new Date();
        let chartdata: QCSPCChartTS.SPCControlChartData | null =
spcchart.getChartData();
        if (!chartdata) return;
        let currentcount = chartdata.getCurrentNumberRecords();

    for (i = 0; i < count; i++) {
        // Simulate a sample subgroup record
        let samples: QCSPCChartTS.DoubleArray =
chartdata.simulateMeasurementRecordMeanRange(mean, sigma);
        // Update chart data using i as the batch number
        batchCounter = currentcount + i;
        let note: string = "";
        if ((i % 5) == 0) note = "This is a note";
        // Add a new sample record to the chart data
        chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, note);
        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }

}
```

Control Limit Alarms

Class SPCControlLimitRecord

ChartObj
|
+-- SPCControlLimitRecord

The SPCControlLimitRecord stores control limit alarm information for the SPCControlChartData class. The SPCControlLimitRecord class specifies the type of the alarm, the alarm limit value, alarm text messages and alarm hysteresis value. The SPCControlChartData classes store the SPCControlLimitRecord objects in the SPCControlChartData.ControlLimitValues array list

SPCControlLimitRecord constructors

This constructor creates a new instance of a **SPCControlLimitRecord** object, using the specified spc data object, calculated value object, alarm type, alarm limit value and alarm message.

```
public static newSPCControlLimitRecordProcessVarAlarmType(processvar:
SPCControlChartData, parametertype: number): SPCControlLimitRecord

public static newSPCControlLimitRecordAlarmTypeAlarmValue(parametertype: number,
alarmlimitvalue: number): SPCControlLimitRecord

public static newSPCControlLimitRecordProcessVarAlarmTypeAlarmValue(processvar:
SPCControlChartData, parametertype: number, alarmlimitvalue: number):
SPCControlLimitRecord
```

```
public static
newSPCControlLimitRecordProcessVarAlarmTypeAlarmValueAlarmMessages(processvar:
SPCControlChartData, parametertype: number, alarmlimitvalue: number,
        normalmessage: string, alarmmessage: string): SPCControlLimitRecord

public static
newSPCControlLimitRecordProcessVarRulesetRuleNumAlarmTypeAlarmValueMessages(proces
svar: SPCControlChartData, controlruleset: number, controlrulenum: number,
parametertype: number, alarmlimitvalue: number, normalmessage: string,
alarmmessage: string): SPCControlLimitRecord

public static
newSPCControlLimitRecordProcessVarCalculatedValueRulesetRuleNumAlarmTypeAlarmValue
Messages(processvar: SPCControlChartData, clr: SPCCalculatedValueRecord,
controlruleset: number, controlrulenum: number, parametertype: number,
alarmlimitvalue: number, normalmessage: string, alarmmessage: string):
SPCControlLimitRecord

public static
newSPCControlLimitRecordProcessVarCalculatedValueAlarmTypeAlarmValueMessages(proce
ssvar: SPCControlChartData, clr: SPCCalculatedValueRecord, parametertype: number,
alarmlimitvalue: number, normalmessage: string, alarmmessage: string):
SPCControlLimitRecord

public static
newSPCControlLimitRecordProcessVarProcessCapabilityAlarmTypeAlarmValueMessages(pro
cessvar: SPCControlChartData, clr: SPCProcessCapabilityRecord,
parametertype: number, alarmlimitvalue: number, normalmessage: string,
alarmmessage: string): SPCControlLimitRecord

public static
newSPCControlLimitRecordProcessVarAlarmTypeAlarmValueMessagesHysteresis(processva:
SPCControlChartData, parametertype: number, alarmlimitvalue: number,
normalmessage: string, alarmmessage: string, hysteresisvalue: number):
SPCControlLimitRecord

public static
newSPCControlLimitRecordProcessVarAlarmTypeAlarmValueSigma(parametertype: number,
alarmlimitvalue: number, sigma: number): SPCControlLimitRecord
```

Parameters

processvar
: Specifies the process variable that the alarm is attached to.

clr
: Specifies the calculated value record the alarm is attached to.

parametertype
: Specifies the alarm type: SPC_NOTA_LIMIT, SPC_LOWERTHAN_LIMIT, or SPC_GREATERTHAN_LIMIT.

alarmlimitvalue
: Specifies the alarm limit value.

normalmessage
: Specifies display message when no alarm present.

alarmmessage
: Specifies the alarm message.

sigma
: Specifies the sigma value of the alarm.

hysteresisvalue

Specifies the hysteresis value of the alarm.

Note - Since JavaScript/TypeScript does not support properties in the same way that C# does, you set and get the values of the properties using the "set" and "get" in front of the property name, i.e. **getControlLimitText** and s**etControlLimitText.**

The most commonly used SPCControlLimitRecord properties are:

Public Static Fields

SPC_GREATERTHAN_LIMIT	Specifies the alarm is a greater than alarm.
SPC_LOWERTHAN_LIMIT	Specifies the alarm is a lower than alarm.
SPC_NOTA_LIMIT	Specifies the limit is not an alarm, just a value.

Public Instance Constructors

SPCControlLimitRecord	Initializes a new instance of the **SPCControlLimitRecord** class.

Public Instance Properties

AlarmDisplay	set/get the alarm display flag.
AlarmEnable	set/get the alarm enable flag.
AlarmMessage	set/get the current alarm message.
AlarmState	set/get the alarm state, true if the last call to CheckAlarm show that the process variable currently in alarm.
ControlLimitText	set/get the Normal alarm message;
ControlLimitType	set/get the alarm type: SPC_NOTA_LIMIT, SPC_LOWERTHAN_LIMIT, or SPC_GREATERTHAN_LIMIT.
ControlLimitValue	set/get the alarm limit value.
ControlLimitValues	set/get the controlLimitValues array.
HysteresisValue	set/get the alarm hysteresis value.
PrevAlarmState	set/get the previous alarm state.
SPCProcessVar	set/get the spcDataVar object.
SymbolColor	set/get the alarm symbol color.
TextColor	set/get the alarm text color.

Public Instance Functions

checkAlarm	Check the current value against the parameter value.
getAlarm	Returns the current alarm state based on the passed in value.
getControlLimitHistoryValue	Get a value for the controlLimitsValues historical buffer.

setControlLimitValue	Set current value of the control limit and adds that value to the controlLimitValues historical array.

The SPCControlLimitRecord properties are documented in the QCSPCChartJSTSClassesIndex.html documentation file, located in the docs/docs/ subdirectory.

[JavaScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';

function alarmEventChanged( source,  e)
{

   var alarm = e.getEventAlarm();
   if (alarm)
   {
     var alarmlimitvalue = alarm.getControlLimitValue();
     var alarmlimitvaluestring =  alarmlimitvalue.toString();
     var spcData  = alarm.getSPCProcessVar();
   }
}
```

[TypeScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';

export class VariableControlCharts implements QCSPCChartTS.SPCAlarmEventListener {


    public constructor() {

    }

 alarmEventChanged(source: QCSPCChartTS.SPCControlChartData, e:
QCSPCChartTS.SPCControlLimitAlarmArgs): void
 {

    let alarm: QCSPCChartTS.SPCControlLimitRecord | null = e.getEventAlarm();
    if (alarm != null)
    {
      let alarmlimitvalue: number = alarm.getControlLimitValue();
      let alarmlimitvaluestring: string =  alarmlimitvalue.toString();
       let spcData: QCSPCChartTS.SPCControlChartData | null  =
alarm.getSPCProcessVar();
    }
}
```

Control Limit Alarm Event Handling

Class SPCControlLimitAlarmArgs

ChartObj

|
+-- **SPCControlLimitAlarmArgs**

The **SPCControlChartData** class can throw an alarm event based on either the current alarm state, or an alarm transition from one alarm state to another. The **SPCControlLimitAlarmArgs** passes alarm data to the event handler. If you want the alarm event triggered only on the initial transition from the no-alarm state to the alarm state, set the **SPCControlChartData.AlarmTransitionEventEnable** to true and the **SPCControlChartData.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 **SPCControlLimitRecord** object is in alarm, set the **SPCControlChartData.AlarmTransitionEventEnable** to false and the **SPCControlChartData.AlarmStateEventEnable** to true. The alarm events will be generated at the same rate as the **SPCControlChartData.addNewSampleRecord()** method is called.

SPCControlLimitAlarmArgs constructors

You don't really need the constructors since **SPCControlLimitAlarmArgs** objects are created inside the **SPCControlChartData** class when an alarm event needs to be generated.

Note - Since JavaScript/TypeScript does not support properties in the same way that C# does, you set and get the values of the properties using the "set" and "get" in front of the property name, i.e. **getAlarmChannel** and s**etAlarmChannel.**

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

Selected Public Instance Properties

Public Instance Properties

AlarmChannel	set/get the alarm channel associated with the alarm.
EventAlarm	set/get the **SPCControlLimitRecord** object.
SPCSource	set/get the SPCCalculatedValueRecord object associated with the alarm.

A complete listing of **SPCControlLimitAlarmArgs** properties are documented in the QCSPCChartJSTSClassesIndex.html documentation file, located in the /docs/docs subdirectory.

Setup and enable an alarm transition event handler using the following three steps.

- [TypeScript] Add the **implements SPCAlarmEventListener** interface to the main class.
- Add an implementation of the **SPCAlarmEventListener** method **alarmEventChanged** to the chart class
- Turn on alarm transition alarm events using the ChartData **setAlarmTransitionEventEnable**(true) method.

[JavaScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';

function alarmEventChanged( source,  e)
{

   var alarm = e.getEventAlarm();
   if (alarm)
   {
     var alarmlimitvalue = alarm.getControlLimitValue();
     var alarmlimitvaluestring =  alarmlimitvalue.toString();
     var spcData  = alarm.getSPCProcessVar();
   }
}

export async function BuildXBarRChart(canvasid) {

    var htmlcanvas = document.getElementById(canvasid);
    var spccharttype = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    var subgroupsize = 5;
    var numberpointsinview = 12;
    var charttitle = "XBar-R Example";


    var xbarrchart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);

    xbarrchart.setPreferredSize(800, 600);

    xbarrchart.setGraphStopPosX(0.825);
    xbarrchart.setGraphStartPosX(0.18);

xbarrchart.setChartAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_SYM
BOL);
    xbarrchart.setEnableScrollBar(true);

    xbarrchart.setEnableCategoryValues(true);
    xbarrchart.setEnableCalculatedValues(true);
    xbarrchart.setEnableAlarmStatusValues(true);
    xbarrchart.setEnableChartToggles(true);

xbarrchart.setTableAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_BAR
);

xbarrchart.setHeaderStringsLevel( QCSPCChartTS.SPCControlChartData.HEADER_STRINGS_
LEVEL1);

    var chartdata = xbarrchart.getChartData();
    if (chartdata)
    {
        chartdata.setTitle (charttitle);
        chartdata.setChartDescriptior("XBar-R");
        chartdata.setPartNumber ( "283501");
        chartdata.setChartNumber("17");
        chartdata.setPartName( "Transmission Casing Bolt");
        chartdata.setOperation ( "Threading");
        chartdata.setOperator("J. Fenamore");
```

```
        chartdata.setMachine("#11");
         var  today = new Date();
         chartdata.setDateString(today.toLocaleString());
         chartdata.setNotesMessage( "Control limits prepared May 10");
         chartdata.setNotesHeader("NOTES"); // row header
         chartdata.addAlarmTransitionEventListener(alarmEventChanged);
         chartdata.setAlarmTransitionEventEnable(true);
    }

    xbarrchart.setEnableDisplayOptionToggles(true);

    var numssampleintervals = 100;
    var chartmean = 30;
    var chartsigma = 5;

    SimulateData(xbarrchart, numssampleintervals, chartmean, chartsigma);


    // Calculate the SPC control limits for both graphs of the current SPC chart
(X-Bar R)
    xbarrchart.autoCalculateControlLimits();

    // Out some more data, different mean and sigma, to simulate out of control
    numssampleintervals = 50;
    chartmean = 32;
    chartsigma = 8;

    SimulateData(xbarrchart, numssampleintervals, chartmean, chartsigma);

    // Scale the y-axis of the X-Bar chart to display all data and control limits
    xbarrchart.autoScalePrimaryChartYRange();
    // Scale the y-axis of the Range chart to display all data and control limits
    xbarrchart.autoScaleSecondaryChartYRange();
    // Rebuild the chart using the current data and settings
    xbarrchart.rebuildChartUsingCurrentData();

}
```

[TypeScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';

export class VariableControlCharts implements QCSPCChartTS.SPCAlarmEventListener {


    public constructor() {

    }

 alarmEventChanged(source: QCSPCChartTS.SPCControlChartData, e:
QCSPCChartTS.SPCControlLimitAlarmArgs): void
 {

    let alarm: QCSPCChartTS.SPCControlLimitRecord | null = e.getEventAlarm();
    if (alarm != null)
    {
      let alarmlimitvalue: number = alarm.getControlLimitValue();
      let alarmlimitvaluestring: string =  alarmlimitvalue.toString();
      let spcData: QCSPCChartTS.SPCControlChartData | null  =
alarm.getSPCProcessVar();
    }
}

public async  BuildXBarRChart(canvasid: string) {

    let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
    let spccharttype: number  = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    let subgroupsize: number = 5;
    let numberpointsinview: number  = 12;
    let charttitle: string = "XBar-R Example";
```

```
    let xbarrchart: QCSPCChartTS.SPCChartBase =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
Subgroupsize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);

    xbarrchart.setPreferredSize(800, 600);

    xbarrchart.setGraphStopPosX(0.825);
    xbarrchart.setGraphStartPosX(0.18);

xbarrchart.setChartAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_SYM
BOL);
    xbarrchart.setEnableScrollBar(true);

    xbarrchart.setEnableCategoryValues(true);
    xbarrchart.setEnableCalculatedValues(true);
    xbarrchart.setEnableAlarmStatusValues(false);
    xbarrchart.setEnableChartToggles(true);

xbarrchart.setTableAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_BAR
);

xbarrchart.setHeaderStringsLevel( QCSPCChartTS.SPCControlChartData.HEADER_STRINGS_
LEVEL1);

    xbarrchart.setEnableDisplayOptionToggles(true);

    let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarrchart.getChartData();
    if (chartdata) {
      chartdata.setTitle (charttitle);
      chartdata.setChartDescriptior("XBar-R");
      chartdata.setPartNumber ( "283501");
      chartdata.setChartNumber("17");
      chartdata.setPartName( "Transmission Casing Bolt");
      chartdata.setOperation ( "Threading");
      chartdata.setOperator("J. Fenamore");
      chartdata.setMachine("#11");
      let  today: Date = new Date();
      chartdata.setDateString(today.toLocaleString());
      chartdata.setNotesMessage( "Control limits prepared May 10");
      chartdata.setNotesHeader("NOTES"); // row header
      chartdata.addAlarmTransitionEventListener(this);
      chartdata.setAlarmTransitionEventEnable(true);
    }
```

The event handler method is the method **alarmEventChanged.**

[JavaScript]

```
function alarmEventChanged( source,  e)
{

   var alarm = e.getEventAlarm();
   if (alarm)
   {
     var alarmlimitvalue = alarm.getControlLimitValue();
     var alarmlimitvaluestring =  alarmlimitvalue.toString();
     var spcData  = alarm.getSPCProcessVar();
   }
}
```

[TypeScript]

```
 alarmEventChanged(source: QCSPCChartTS.SPCControlChartData, e:
QCSPCChartTS.SPCControlLimitAlarmArgs): void
 {

    let alarm: QCSPCChartTS.SPCControlLimitRecord | null = e.getEventAlarm();
    if (alarm != null)
    {
      let alarmlimitvalue: number = alarm.getControlLimitValue();
      let alarmlimitvaluestring: string =  alarmlimitvalue.toString();
      let spcData: QCSPCChartTS.SPCControlChartData | null  =
alarm.getSPCProcessVar();
    }
}
```

There is difference in the way the event handler is defined in JavaScript compared to TypeScript. In JavaScript, the event handler is a function local to the module. It is specified as the event handler using the code:

```
chartdata.addAlarmTransitionEventListener(alarmEventChanged);
chartdata.setAlarmTransitionEventEnable(true);
```

In JavaScript, the function signature (i.e. parameter list) of the event handler must look exactly like code seen for alarmEventChanged, because you will not get any error indication if it is wrong. Setup and enable an alarm state event handler in an identical manner:

```
chartdata.addAlarmStateEventListener(alarmEventChanged);
chartdata.setAlarmStateEventEnable(true);
```

In the case of TypeScript, the enclosing class, VariableControlCharts designates that it implements the SPCAlarmEventListener using the line

```
export class VariableControlCharts implements QCSPCChartTS.SPCAlarmEventListener {
```

To resolve the implementation, the alarmEventChanged method is added to the class.

```
 alarmEventChanged(source: QCSPCChartTS.SPCControlChartData, e:
QCSPCChartTS.SPCControlLimitAlarmArgs): void
```

In order to assign this method as the event handler for the alarm use the ChartData addAlarmTransitionEventListener (or addAlarmStateEventListener) method, passing in the current class (this), which contains the event handler alarmEventChanged.

```
chartdata.addAlarmTransitionEventListener(this);
chartdata.setAlarmTransitionEventEnable(true);
```

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

```
chartdata.addAlarmStateEventListener(this);
chartdata.setAlarmStateEventEnable(true);
```

SPCSampledValueRecord

This class encapsulates a sample data value. It includes a description for the item, the current value of the sampled value, and a history of previous values.

While SPCSampledValueRecord has several static constructors, there is no circumstance where you would use them.

An array list of **SPCSampledValueRecord** objects, one for each sample category, is automatically created when the parent **SPCChartBase** object is created. The programmer does not need to instantiate it.

Note - Since JavaScript/TypeScript does not support properties in the same way that C# does, you set and get the values of the properties using the "set" and "get" in front of the property name, i.e. **getSampledValue** and **setSampledValue.**

Public Instance Constructors

SPCSampledValueRecord	Initializes a new instance of the SPCSampledValueRecord class.

Public Instance Properties

SampledValue	set/get the current value for this record.
SampledValues	set/get the historical array of the sampled value record.
ValueDescription	set/get the description of sampled value record.

Public Instance Functions

getCalculatedValueStatistic	Calculate a statistic for the historical data associated with the sample item.
setSampledValue	Set the current value of the record, and adds the value to the historical array of the sampled value record.

A complete listing of **SPCSampledValueRecord** properties are documented in the QCSPCChartJSTSClassesIndex.html documentation file, located in the /docs/docs subdirectory.

SPCCalculatedValueRecord

This is the record class for a calculated SPC statistic. It holds the calculated value type (mean, median, sum, variance, standard deviation, etc.), value, description and historical data.

While SPCCalculatedValueRecord has several static constructors, there is no circumstance where you would use them.

Note - Since JavaScript/TypeScript does not support properties in the same way that C# does, you set and get the values of the properties using the "set" and "get" in front of the property name, i.e. **getCalculatedValue** and s**etCalculatedValue.**

Public Static Methods

calculateHistoryStatistic	Calculate the calculated value value based on the data in the source array and the specified calculation type.

Public Instance Properties

CalculatedValue	set/get the current calculation value for this record.
CalculatedValues	set/get the reference to the calculatedValue array.
CalculationType	set/get the calculation type for this calculation value record. Use one of the SPCCalculatedValueRecord calculation type constants.
MostRecentSampledValues	set/get an array holding the values of the most recent sampled, or measured values used in calculating the records calculateValue value.
ValidValueFlags	set/get the reference to the validValueFlags array.
ValueDescription	set/get the description of calculation value record.

Public Instance Functions

getCalculatedValueStatistic	Returns the calculated value value based on the data in the calculated historical data array, calculatedValues. Excludes values that are marked invalid in the validValueFlags array.
isValueValid	Checks to the validValueFlags to see if a value in the calculated historical data array, calculatedValues, is valid.
setCalculatedValue	Calculate the calculated value value based on the data in the source array. Sets the calculatedValue property of the class to the result.

A complete listing of SPCCalculatedValueRecord properties are documented in the QCSPCChartJSTSClassesIndex.html documentation file, located in the /docs/docs subdirectory.

SPCProcessCapabilityRecord

This is the record class for calculating and storing SPC process capability statistics. It supports calculating the Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppl, Ppu, and Ppk statistics.

While SPCProcessCapabilityRecord has several static constructors, there is no circumstance where you would use them.

Note - Since JavaScript/TypeScript does not support properties in the same way that C# does, you set and get the values of the properties using the "set" and "get" in front of the property name, i.e. **getCurrentValue** and s**etCurrentValue.**

Public Static Fields

SPC_CP_CALC	Constant value CP calculation.
SPC_CPK_CALC	Constant value CPK calculation.
SPC_CPL_CALC	Constant value CPL calculation.
SPC_CPM_CALC	Constant value CPM calculation.
SPC_CPU_CALC	Constant value CPU calculation.
SPC_CUSTOM_PC_CALC	Constant value for a custom SPC calculation (unused).
SPC_PP_CALC	Constant value for a sum SPC calculation.
SPC_PPK_CALC	Constant value PPK calculation.
SPC_PPL_CALC	Constant value PPL calculation.
SPC_PPU_CALC	Constant value PPU calculation.

Public Instance Properties

CalculationType	set/get the calculation type for this calculation value record. Use one of the SPCProcessCapabilityRecord calculation type constants.
CurrentValue	set/get the current calculation value for this record.
CurrentValues	set/get the reference to the currentValue array.
LSLValue	set/get the LSL value for this record.
USLValue	get the USL value for this record.
ValidValueFlags	set/get the reference to the validValueFlags array.

ValueDescription	set/get the description of calculation value record.

Public Instance Functions

calculateProcessCapabilityValue	Calculate the process capability value..
calculateProcessCapabilityValues	Calculate the process capability value..
isValueValid	Checks to the validValueFlags to see if a value in the calculated historical data array, currentValues, is valid.
reset	Reset the history buffer of the SPCProcessCapabilityRecord class.
setProcessCapabilityValue	Calculate the process capability value. Sets the currentValue property of the class to the result.

A complete listing of **SPCProcessCapabilityRecord** properties are documented in the QCSPCChartJSTSClassesIndex.html documentation file, located in the /docs/docs subdirectory.

SPCGeneralizedTableDisplay

This class manages a list of **ChartText** objects (**NumericLabel**, **StringLabel** and **TimeLabel** objects), that encapsulate each unique table entry in the SPC chart table. This class also manages the spacing between the rows and columns of the table, and the alternating stripe used as a background for the table.

While SPCGeneralizedTableDisplay has several static constructors, there is no circumstance where you would use them.

The main chart class has an instance of SPCGeneralizedTableDisplay. This is one of the properties that in the case of TypeScript, you must check the value against null before trying to access member constants or methods.

[TypeScript]

```
let charttable: QCSPCChartTS.SPCGeneralizedTableDisplay | null =
xbarsigmachart.getChartTable()
    if (charttable)
    {
        charttable.setTableBackgroundMode(QCSPCChartTS.SPCGeneralizedTableDisplay.
TABLE_NO_COLOR_BACKGROUND );
    }
```

[JavaScript]

In Javarscript you can use the same technique,

```
var charttable = xbarsigmachart.getChartTable()
    if (charttable)
```

```
    {
        charttable.setTableBackgroundMode(QCSPCChartTS.SPCGeneralizedTableDisplay.
TABLE_NO_COLOR_BACKGROUND );
    }
```

or just call getChartTable() inline.

```
xbarsigmachart.getChartTable().setTableBackgroundMode(QCSPCChartTS.SPCGeneralizedT
ableDisplay. TABLE_NO_COLOR_BACKGROUND );
```

Note - Since JavaScript/TypeScript does not support properties in the same way that C# does, you set and get the values of the properties using the "set" and "get" in front of the property name, i.e. **getBackgroundColor1** and **setBackgroundColor1.**

Public Static Fields

NUMERIC_ROW_SPACING	Constant specifies that the next row to the table should user numeric row spacing.
TABLE_NO_COLOR_BACKGROUND	Constant specifies that the table does not use a background color.
TABLE_SINGLE_COLOR_BACKGROUND	Constant specifies that the table uses a single color for the background (backgroundColor1).
TABLE_SINGLE_COLOR_BACKGROUND_GRID	Constant specifies that the table uses horizontal stripes of color for the background (backgroundColor1 and backgroundColor2).
TABLE_STRIPED_COLOR_BACKGROUND	Constant specifies that the table uses horizontal stripes of color for the background (backgroundColor1 and backgroundColor2).
TEXT_ROW_SPACING	Constant specifies that the next row to the table should user text row spacing.

Public Instance Properties

DefaultTableFont	set/get the default font used in the table display.
BackgroundColor1	set/get the first of two colors used in the alternating background colors used to delineate the table rows.

BackgroundColor2	set/get the second of two colors used in the alternating background colors used to delineate the table rows.
CalculatedItemTemplate	set/get the CalculatedItemTemplate object used as a template for displaying calculated numeric values in the table.
CalculatedLabelFont	set/get the font used in the display of calculated numeric values in the table.
CurrentColumnPosition	set/get the current column position.
CurrentRowPosition	set/get the current column position.
NotesItemTemplate	set/get the StringItemTemplate object used as a template for displaying string values in the table.
NotesLabelFont	set/get the font used in the display of string values in the table.
NumericColumnSpacing	set/get the numeric column spacing.
NumericRowSpacing	set/get the numeric row spacing.
SampleItemTemplate	set/get the SampleItemTemplate object used as a template for displaying numeric values in the table.
SampleLabelFont	set/get the font used in the display of sample numeric values in the table.
StartColumnPosition	set/get the starting x-position, in normalized coordinates, of the left-most column of the table.
StartRowPosition	set/get the starting y-position, in normalized coordinates, of the first row of the table.
StringItemTemplate	set/get the StringItemTemplate object used as a template for displaying string values in the table.
StringLabelFont	set/get the font used in the display of string values in the table.
TableBackgroundMode	set/get the first of two colors used in the alternating background colors used to delineate the table rows.
TextColumnSpacing	set/get the text column spacing.
TextRowOffset	set/get the offset between the start of the row and the top of the text, in normalized coordinates.
TextRowSpacing	set/get the text row spacing.
TimeColumnSpacing	set/get the time column spacing.
TimeItemTemplate	set/get the TimeLabel object used as a template for displaying time values in the table.

TimeLabelFont	set/get the font used in the display of time values in the table.
TimeRowSpacing	set/get the time row spacing.

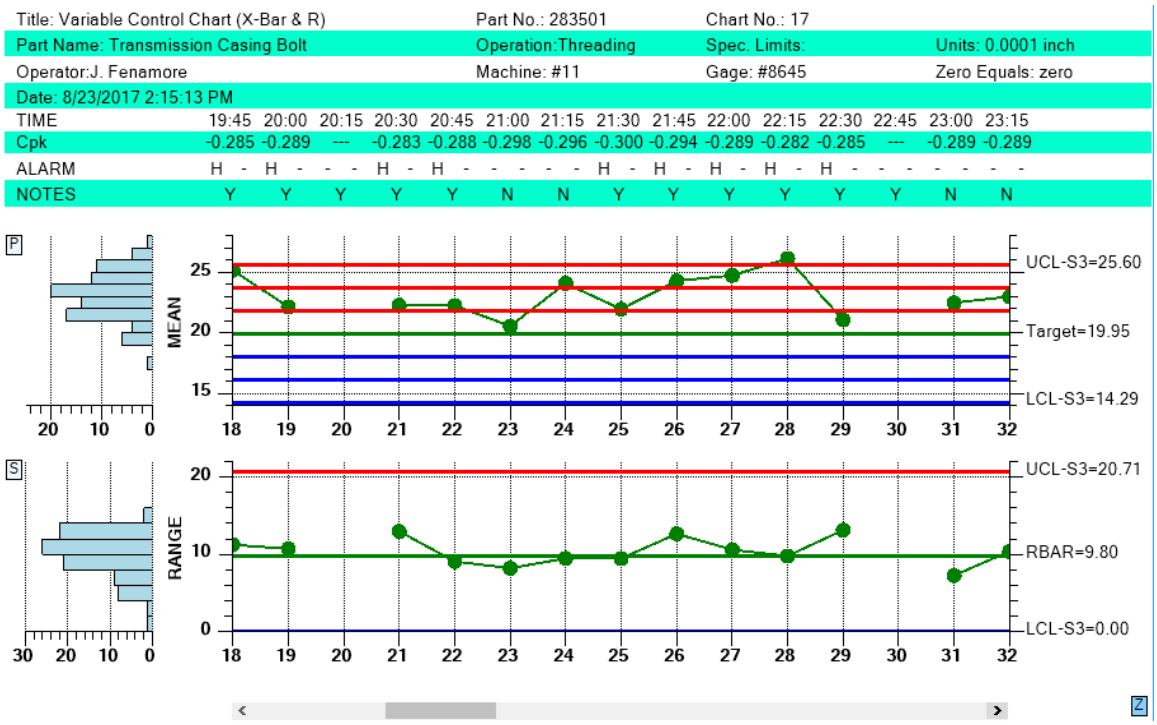
Public Instance Functions

addCalculatedItem	Add a calculated numeric item to the table, using the specified column spacing increment.
addHorizontalBar	Add a horizontal bar as a row background for the table.
addNotesItem	Add a string item to the table, using the specified column spacing increment.
addNumericItem	Add a numeric item to the table, using the specified column spacing increment.
addStringItem	Add a string item to the table, using the specified column spacing increment.
addTimeItem	Add a time item to the table, using the specified column spacing increment.
getChartLabel	Get a specific ChartLabel object in the chartLabelArray array list.
incrementRow	Add another row to the table, using the specified row spacing increment.
initDefaults	Initialize default values for the class.

A complete listing of **SPCGeneralizedTableDisplay** properties are documented in the QCSPCChartJSTSClassesIndex.html documentation file, located in the /docs/docs subdirectory.

Marking a sample internal as bad

A sample interval can be marked as bad. This will disable plotting of the sample interval in the Primary and Secondary charts. It will also prevent the sample interval values from being evaluated in control limit testing, and in auto- calculations for control limits and y-axis range.



A hole (sample intervals 20 and 30 in the picture above) will appear in the main plot of the Primary and Secondary charts when a sample interval is marked bad.

excludeRecordFromControlLimitCalculations

Exclude the specified record from the SOC control limit calculations.

```
public excludeRecordFromControlLimitCalculations( item: number, exclude: boolean)
```

item The index of the item to exclude.

exclude Set true and the item is excluded. Set false and it is included.

For example:

[JavaScript]

```
var chartdata = xbarsigmachart.getChartData();
if (chartdata)
  chartdata.excludeRecordFromControlLimitCalculations(7, true);
```

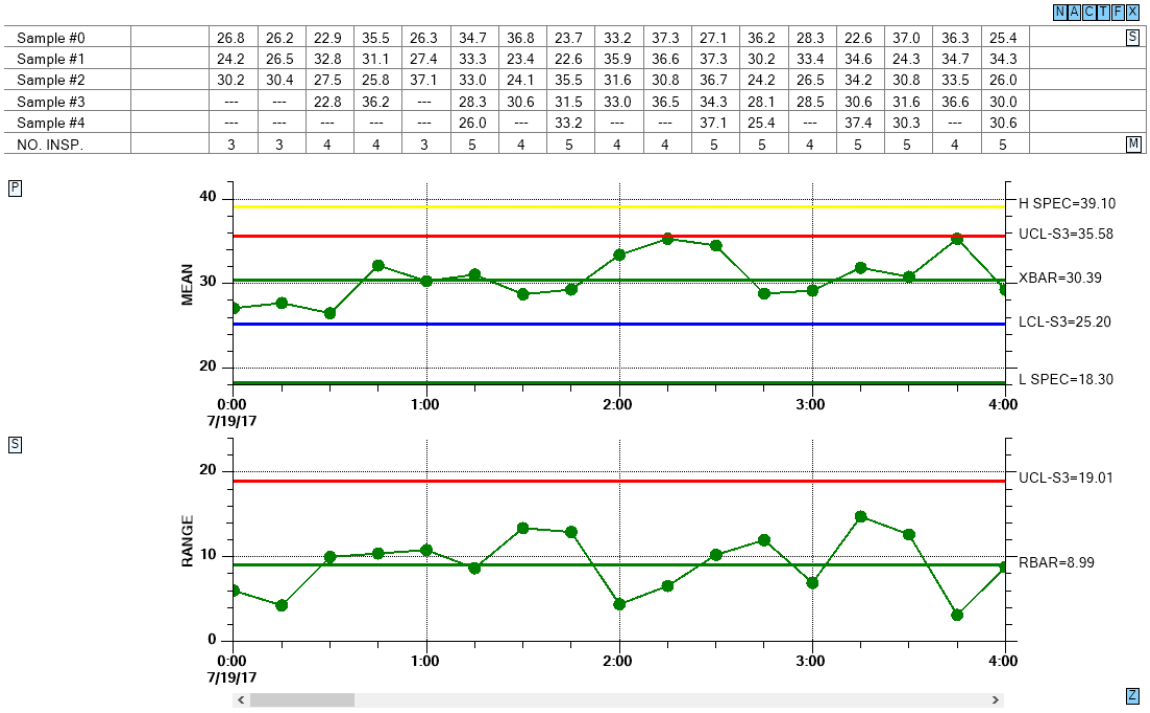
[TypeScript]

```
let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarsigmachart.getChartData();
if (chartdata)
  chartdata.excludeRecordFromControlLimitCalculations(7, true);
```

Sample Interval Update with an invalid number of samples

When you have specified one of the fixed sample size charts, a more lenient evaluation method for sample interval updates has been implemented. You can now enter less than or greater than the specified number of samples for an interval update without the software complaining. It will process the samples as if the proper number of values was entered, without introducing zero values into the sample interval. So if the specified number of samples per sample interval is 5, and 3 are entered, it will still work. It will calculate the mean and range of the sample interval using only 3 values. However, it will not adjust the control limits to take into account the incorrect number of samples, unlike what is done in the dedicated variable sample size (_VSS) charts. Also, you cannot enter in a single value for any of the control charts which require more than one value. Because then the software cannot calculate the range or sigma value, a required calculation. The programmer and user are cautioned that any alarms (or lack thereof) triggered when using this option may not be valid.

In the example below, the number of samples per sample interval varies from three to five samples. Note that the table displays as many rows of sample data as the fixed number of samples per sample interval in the chart definition, five in this case. Missing data values are designated using "...".



Sample #0		26.8	26.2	22.9	35.5	26.3	34.7	36.8	23.7	33.2	37.3	27.1	36.2	28.3	22.6	37.0	36.3	25.4
Sample #1		24.2	26.5	32.8	31.1	27.4	33.3	23.4	22.6	35.9	36.6	37.3	30.2	33.4	34.6	24.3	34.7	34.3
Sample #2		30.2	30.4	27.5	25.8	37.1	33.0	24.1	35.5	31.6	30.8	36.7	24.2	26.5	34.2	30.8	33.5	26.0
Sample #3		---	---	22.8	36.2	---	28.3	30.6	31.5	33.0	36.5	34.3	28.1	28.5	30.6	31.6	36.6	30.0
Sample #4		---	---	---	---	---	26.0	---	33.2	---	---	37.1	25.4	---	37.4	30.3	---	30.6
NO. INSP.		3	3	4	4	3	5	4	5	4	4	5	5	4	5	5	4	5

Note that the sample data in the table has many columns where the number of samples is less than the specified 5 samples per sample interval.

If you update the sample interval with more than the specified number of samples, the software will calculate the mean, sigma, and range of the sample interval (and any other calculations in needs to) using all of the values values. It will NOT store the additional values though. The table will not show these additional values. The sample interval data is only displayed up to the original, fixed, number of samples per sample interval.

If you acquire your sample data, and find that you do not have valid, or sufficient samples for the current chart (> 1 for all but the Individual Range charts), you should skip the sample interval update for that sample interval.

Alarm Forcing

High and low control lines can be set to explicit values, or you can have the software auto-calculate the values based on historical data. As new sample intervals are added to the chart, the new values are compared to existing control limits and the alarm conditions noted. But, you can override the alarm limit display, so that a normal condition shows as an alarm, or an alarm condition shows as normal. Obviously you can distort the meaning of the chart if you hide alarms or show alarms that the sample data does not support. But that issue is left to the programmer and end user.

setForcedControlLimitValues

Force the current sample interval of the chart to the specified forcing value.

```
public setForcedControlLimitValues( chartnum: number,  forcingvalue: number)
```

chartnum	chart number. Use SPCChartObjects.PRIMARY_CHART or SPCChartObjects.SECONDARY_CHART
forcingvalue	forcing value. Use one of forcing value constants: SPCChartForceAlarm.FORCE_LOW, SPCChartForceAlarm.FORCE_HIGH, SPCChartForceAlarm.FORCE_NORMAL

For example:

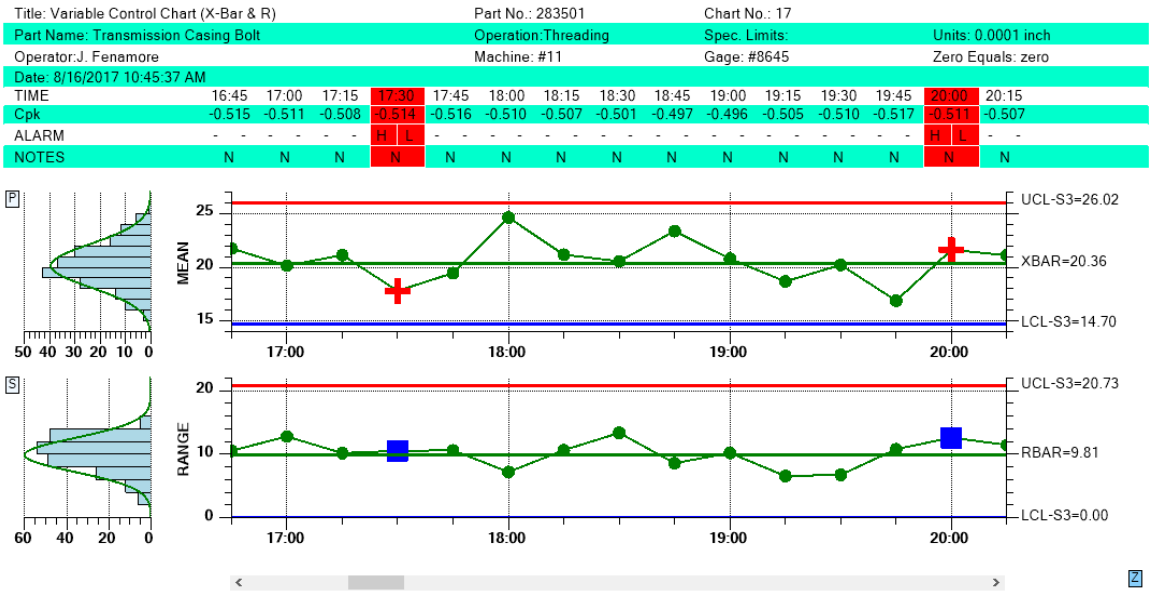
[JavaScript]

```
var  chartdata = xbarsigmachart.getChartData();
if (chartdata)
{
chartdata.setForcedControlLimitValuesChartNumForcingValue(QCSPCChartTS.SPCChartObj
ects.PRIMARY_CHART, QCSPCChartTS.SPCChartForceAlarm.FORCE_HIGH);
chartdata.setForcedControlLimitValuesChartNumForcingValue(QCSPCChartTS.SPCChartObj
ects.SECONDARY_CHART, QCSPCChartTS.SPCChartForceAlarm.FORCE_LOW);
}:
```

[TypeScript]

```
let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarsigmachart.getChartData();
```

```
if (chartdata)
{
chartdata.setForcedControlLimitValuesChartNumForcingValue(QCSPCChartTS.SPCChartObj
ects.PRIMARY_CHART, QCSPCChartTS.SPCChartForceAlarm.FORCE_HIGH);
chartdata.setForcedControlLimitValuesChartNumForcingValue(QCSPCChartTS.SPCChartObj
ects.SECONDARY_CHART, QCSPCChartTS.SPCChartForceAlarm.FORCE_LOW);
}
```



Sample intervals can be forced to into an alarm state, regardless of the calculated value. All of the alarms in the example above are forced and not representative of the sample data.

Chapter 6 - SPC Variable Control Charts

SPCEventVariableControlChart
SPCTimeVariableControlChart
SPCBatchVariableControlChart

Variable Control Charts are used with sampled quality data that can be assigned a specific numeric value, other than just 0 or 1. This includes, but is not limited to, the measurement of a critical dimension (height, length, width, radius, etc.), the weight a specific component, or the measurement of an important voltage. The variable control charts supported by this software include X-Bar R (Mean and Range), X-Bar Sigma, Median and Range, X-R (Individual Range), MA (Move Average), MAMR (Moving Average / Moving Range), MAMS (Moving Average / Moving Sigma), EWMA (Exponentially Weighted Moving Average) and CUSum charts.

X-Bar R Chart – Also known as the Mean (or Average) and Range Chart

The X-Bar R chart monitors the trend of a critical process variable over time using a statistical sampling method that results in a subgroup of values at each sample interval. The X-Bar part of the chart plots the mean of each sample subgroup and the Range part of the chart monitors the difference between the minimum and maximum value in the subgroup. X-Bar R charts are created using the **SPCEventVariableControlChart**, **SPCTimeVariableControlChart** and **SPCBatchVariableControlChart** classes.

X-Bar Sigma – Also known as the X-Bar S Chart

Very similar to the X-Bar R chart, the X-Bar Sigma chart replaces the Range plot with a Sigma plot based on the standard deviation of the measured values within each subgroup. This is a more accurate way of establishing control limits if the sample size of the subgroup is moderately large (> 10). Though computationally more complicated, the use of a computer makes this a non-issue. The X-Bar Sigma chart comes in fixed sample subgroup size, and variable sample subgroup size, versions. X-Bar Sigma charts are created using the **SPCEventVariableControlChart, SPCTimeVariableControlChart** and **SPCBatchVariableControlChart** classes.

Median Range – Also known as the Median and Range Chart

Very similar to the X-Bar R Chart, Median Range chart replaces the Mean plot with a Median plot representing the median of the measured values within each subgroup. In order to use a Median Range chart the process needs to be well behaved, where the variation in measured variables are (1) known to be distributed normally, (2) are not very

often disturbed by assignable causes, and (3) are easily adjusted. . Median Range charts are created using the **SPCEventVariableControlChart, SPCTimeVariableControlChart** and **SPCBatchVariableControlChart** classes.

Individual Range Chart – Also known as the X-R or I-R Chart

The Individual Range Chart is used when the sample size for a subgroup is 1. This happens frequently when the inspection and collection of data for quality control purposes is automated and 100% of the units manufactured are analyzed. It also happens when the production rate is low and it is inconvenient to have sample sizes other than 1. The X part of the control chart plots the actual sampled value (not a mean or median) for each unit and the R part of the control chart plots a moving range that is calculated using the current value of sampled value minus the previous value. . Individual Range charts are created using the **SPCEventVariableControlChart, SPCTimeVariableControlChart** and **SPCBatchVariableControlChart** classes.

Levey-Jennings Chart

The Levey-Jennings chart is used almost exclusively in laboratory settings. It uses a chart very similar to the Individual Range chart above, the major difference being that it only uses the Primary individual data point graph of the chart and does not include the Secondary range graph. Also, the Levey-Jennings chart uses the Westgard rules which utilizes tests involving 1-, 2- and 3- sigma control limits. The control limit calculations depart from all of the other SPC Chart types in that the target value (mean) and control limit (sigma) calculations use the overall mean and standard deviation values from the entire, charted, sample population. See the links https://en.wikipedia.org/wiki/Laboratory_quality_control and https://www.westgard.com/lesson12.htm for more information about the underlying principles of the Levey-Jennings chart.

EWMA Chart – Exponentially Weighted Moving Average

The EWMA chart is an alternative to the preceding Shewhart type control charts (X-Bar R and I-R charts in particular) and is most useful for detecting small shifts in the process mean. It uses a weighted moving average of previous values to “smooth” the incoming data, minimizing the affect of random noise on the process. It weights the current and most recent values more heavily than older values, allowing the control line to react faster than a simple MA (Moving Average) plot to changes in the process. Like the Shewhart charts, if the EWMA value exceeds the calculated control limits, the process is considered out of control. While it is usually used where the process uses 100% inspection and the sample subgroup size is 1 (same is the X-R chart), it can also be used when sample subgroup sizes are greater than one. EWMA charts are created using the **SPCEventVariableControlChart, SPCTimeVariableControlChart** and **SPCBatchVariableControlChart** classes.

MA Chart – Moving Average

The MA chartis another alternative to the preceding Shewhart type control charts (X-Bar Rand I-R charts in particular) and is most useful for detecting small shifts in the process mean. It uses a moving average, where the previous (N-1) sample values of the process variable are averaged together along with the current value to produce the current chart value. This helps to "smooth" the incoming data, minimizing the affect of random noise on the process. Unlike the EWMA chart the MA chart weights the current and previous (N-1) values equally in the average. While the MA chart can often detect small changes in the process mean faster than the Shewhart chart types, it is generally less effective than either the EWMA chart, or the CuSum chart MA charts are created using the **SPCEventVariableControlChart, SPCTimeVariableControlChart**and **SPCBatchVariableControlChart** classes.

MAMR Chart – Moving Average/Moving Range

The MAMR chart combines our Moving Average chart with a Moving Range chart. The Moving Average chart is primary (topmost) chart, and the Moving Range chart is the secondary (bottom) chart. It uses a single sample/subgroup, same as our standard [Individual-Range], [Moving Average], [EWMA], and [Moving Average] charts. When calculating the Moving Range, it windows the same data values used in the Moving Average calculation. Note that the limits are variable (wider) at the beginning, taking into account the fewer samples in the start up of any type of SPC chart which uses a sliding window in the calculation of moving average and moving range statistics.

MAMS Chart – Moving Average / Moving Sigma

The MAMS chart combines our Moving Average chart with a Moving Sigma chart. The Moving Average chart is primary (topmost) chart, and the Moving Sigma chart is the secondary (bottom) chart. It uses a single sample/subgroup, same as our standard [Individual-Range], [Moving Average], [EWMA], and [Moving Average] charts. When calculating the Moving Sigma, it windows the same data values used in the Moving Average calculation. Note that the limits are variable (wider) at the beginning, taking into account the fewer samples in the start up of any type of SPC chart which uses a sliding window in the calculation of moving average and moving sigma statistics.

CuSum Chart – Tabular, one-sided, upper and lower cumulative sum

The CuSum chart is a specialized control chart, which like the EWMA and MA charts, is considered to be more efficient that the Shewhart charts at detecting small shifts in the process mean, particularly if the mean shift is less than 2 sigma. There are several types of CuSum charts, but the easiest to use and the most accurate is considered the tabular CuSum chart and that is the one implemented in this software. The tabular cusum works

by accumulating deviations that are above the process mean in one statistic (C+) and accumulating deviations below the process mean in a second statistic (C-). If either statistic (C+ or C-) falls outside of the calculated limits, the process is considered out of control.

Event-Based, Time-Based and Batch-Based SPC Charts

The **QCSPCChart** software further categorizes *Variable Control* as either event-, time- or batch- based. Time-based SPC charts are used when data is collected using a subgroup interval corresponding to a specific time interval. Batch-based (and Event-based) SPC charts are used when the data subgroup interval is a sequential batch number that does not correspond to a uniform time interval. The major difference in these two types of SPC charts is the display of the x-axis. Variable control charts that sample using a uniform time interval will generally use a time-based x-axis, with time/date axis labels. Variable control charts that sample based on batches will generally use a numeric-based x-axis, with numeric axis labels.

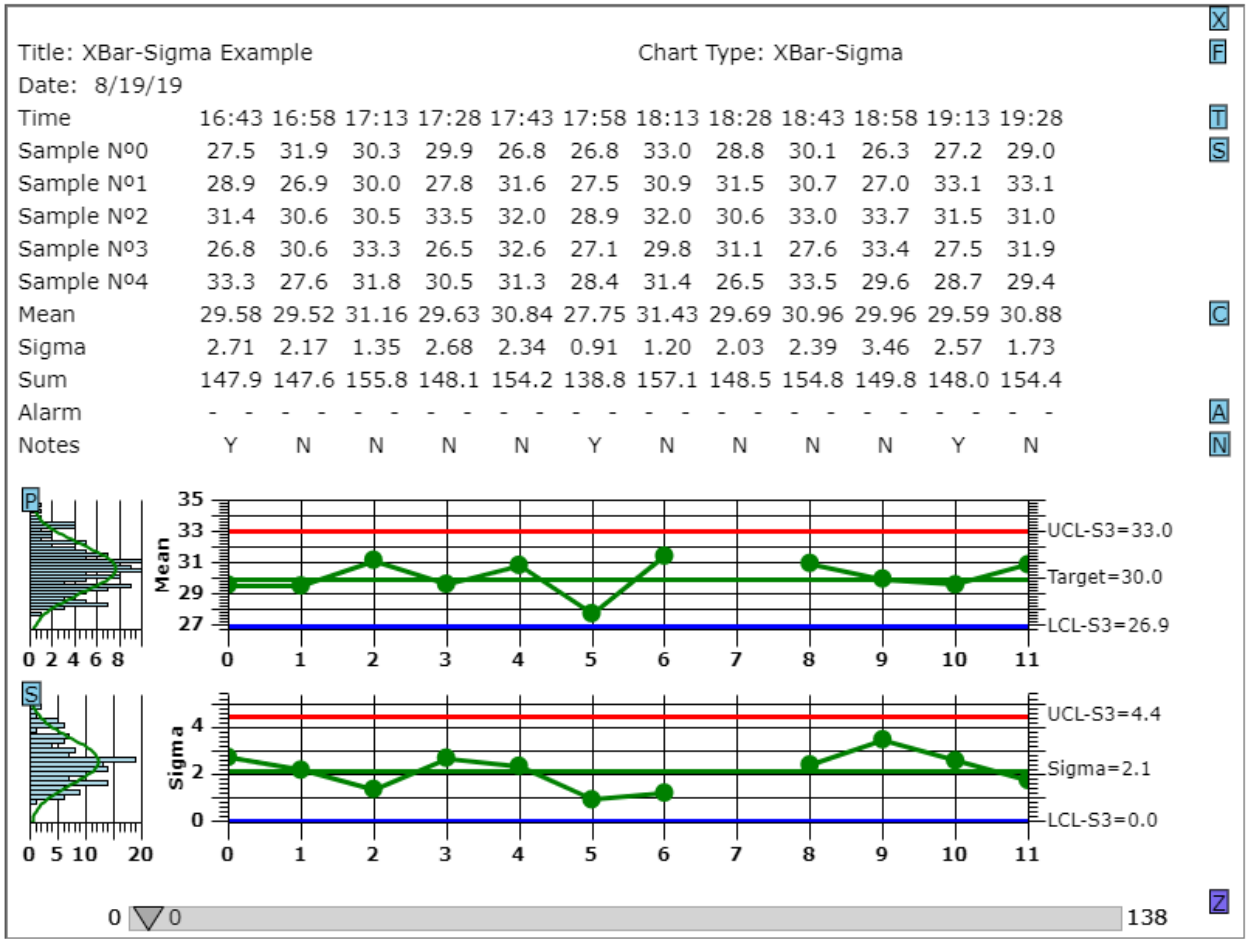
SPECIAL NOTE: The time-based and batch-based SPC charts have been deprecated and replaced by the new event-based charts. In order to maintain backward compatibility, we also keep the old SPCTime... and SPCBatch... control chart classes, but derive them from the new Event-based SPC chart classes. The only difference you might see is an actual benefit. No matter what the time stamp is on a SPCTime... chart, adjacent points will always be equally spaced. So if your sample interval is irregular, or you even skip days or weeks in your sampling, the resulting chart will still display equally spaced adjacent sample records. If you used the SPCTimeVariableControlChart class in other versions of the QCSPCChart software, we recommend that you instead use either the SPCBatchVariableControl (or SPCEventVariableControlChart) class. If you want to see time/date values on the x-axis, set the XAxisStringLabelMode of the chart to AXIS_LABEL_MODE_TIME.

[JavaScript / TypeScript]

```
xbarrchart.setXAxisStringLabelMode(QCSPCChartTS.SPCChartObjects.AXIS_LABEL_MODE_TI
ME)
```

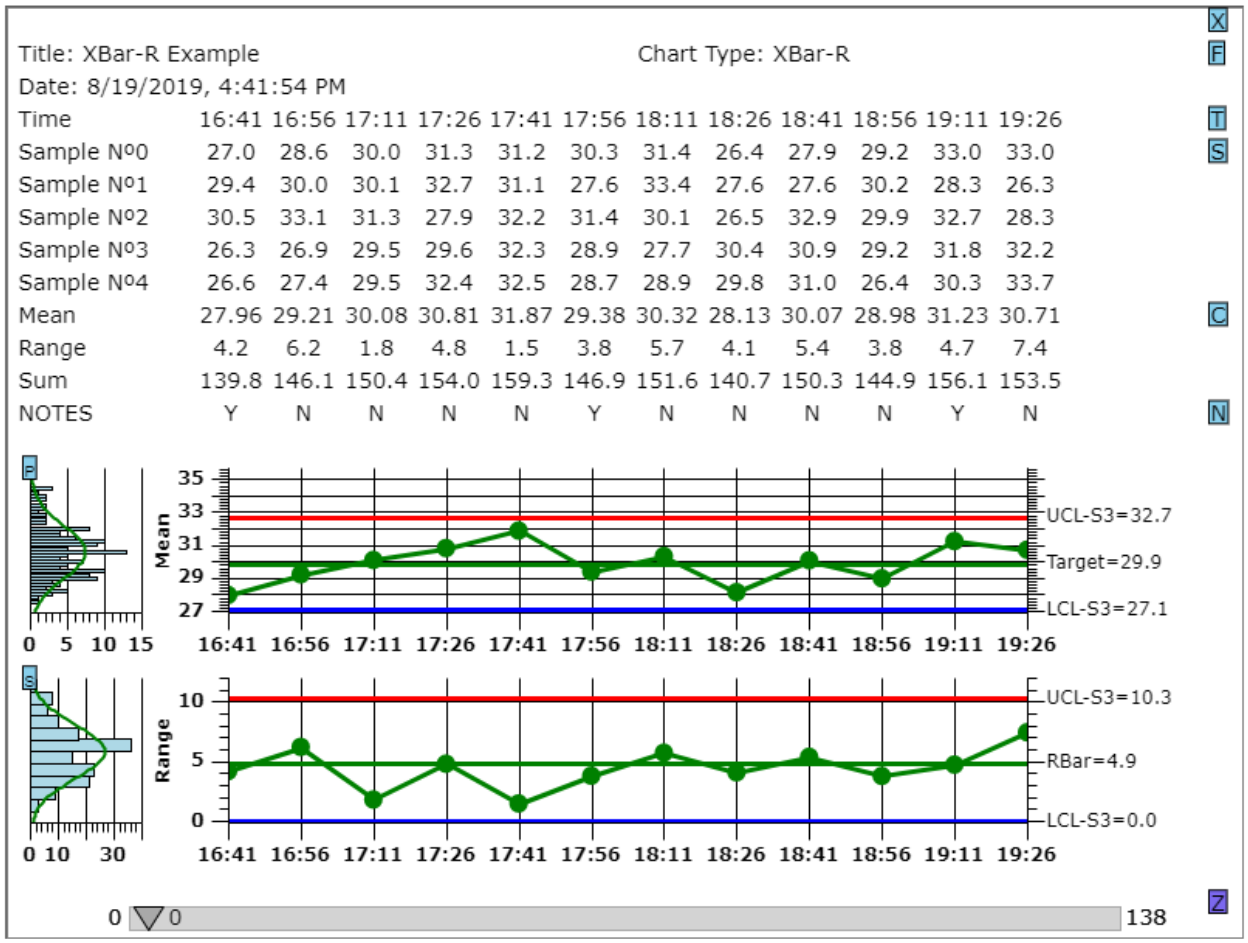
The new class heirarchy looks like:

ChartView
- SPCChartBase
  - SPCEventVariableControlChart
    - SPCTimeVariableControlChart
    - SPCBatchVariableControlChart
  - SPCEventAttributeControlChart
    - SPCTimeAttributeControlChart
    - SPCBatchAttributeControlChart



Batch-Based Variable Control Chart with numeric x-axis

Note the numeric based x-axis for both graphs



Batch-Based Variable Control Chart with time stamp x-axis

Even though the time stamp values may not have consistent time interval, the data points are spaced evenly by batch number.

Creating a Variable Control Chart

We recommend that you use the **newSPCBatchVariableControlChart...** static constructor to create a new instance of a **SPCBatchVariableControlChart**. The example below is extracted from the **VariableControlCharts.BuildXBarRChart** example program.

[JavaScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';

export async function BuildXBarRChart(canvasid) {
```

```
    var htmlcanvas = document.getElementById(canvasid);
    var spccharttype = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    var subgroupsize = 5;
    var numberpointsinview = 12;
    var charttitle = "XBar-R Example";


    var xbarrchart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);

    xbarrchart.setPreferredSize(800, 600);

.
.
.

}
```

[TypeScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';

export class VariableControlCharts implements QCSPCChartTS.SPCAlarmEventListener {


    public constructor() {

    }



public async  BuildXBarRChart(canvasid: string) {

    let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
    let spccharttype: number  = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    let subgroupsize: number = 5;
    let numberpointsinview: number  = 12;
    let charttitle: string = "XBar-R Example";


    let xbarrchart: QCSPCChartTS.SPCChartBase =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);

    xbarrchart.setPreferredSize(800, 600);

 .
.
.


}
.
.

}
```

SPCBatchVariableControlChart Members

Static Instance Constructors

newSPCBatchVariableControlChartChartTypeSubgroupSize	Creates a new instance of the SPCBatchVariableControlChart class and initializes it with the supplied

	parameters.
newSPCBatchCusumControlChart	Creates a new instance of the SPCBatchVariableControlChart initialized as a Cusum chart.

Public Instance Constructors

SPCBatchVariableControlChart	Initializes a new instance of the SPCBatchVariableControlChart class. You will still need to call initSPCBatchVariableControlChart

Public Instance Functions

initSPCBatchVariableControlChartCanvasChartTypeSubgroupSize	Initialize the class for a specific SPC chart type.
initSPCBatchCusumControlChart	Initialize the class a cusum chart type.

The SPCBatchVariableControlChart properties are documented in the QCSPCChartJSTSClassesIndex.html documentation file, located in the docs/docs/ subdirectory.

Note that the X-Bar Sigma chart, with a variable subgroup sample size, is initialized with a *charttype* value of MEAN_SIGMA_CHART_VSS. X-Bar Sigma charts with sub groups that use a variable sample size must be updated properly. See the section "Adding New Sample Records to a X-Bar Sigma Chart (Variable Subgroup Sample Size)" in the "SPC Control Data and Alarm Classes" chapter.

SPCBatchVariableControlChart.initSPCBatchVariableControlChart Method

This initialization method initializes the most important values in the creation of a SPC chart. If you are creating a cusum chart (type TABCUSUM_CHART), your can use the similar **initSPCBatchCusumControlChart** method instead. That version of the init routine has added parameters for the H and K value of the tabular cusum chart.

[TypeScript]

```
public initSPCBatchVariableControlChartCanvasChartTypeSubgroupSize(context: Canvas,
charttype: number, numsamplespersubgroup: number, numdatapointsinview: number)

public static newSPCBatchVariableControlChartChartTypeSubgroupSize(context: Canvas,
charttype: number, numsamplespersubgroup: number, numdatapointsinview: number):
SPCBatchVariableControlChart

public initSPCBatchCusumControlChart(charttype: number, numsamplespersubgroup:
number, numdatapointsinview: number, mean: number, kvalue: number, hvalue: number)

public static newSPCBatchCusumControlChart(context: Canvas, charttype: number,
numsamplespersubgroup: number, numdatapointsinview: number,
        mean: number, kvalue: number, hvalue: number):
SPCBatchVariableControlChart
```

Parameters

canvas
A reference to the the HTML5 canvas object the chart is placed in. Typically the string id of the canvas is passed in, and the reference is looked up using a call document.getElementById(canvasid), as done in each of the example programs.

charttype
The SPC chart type parameter. Use one of the SPCControlChartData SPC chart types: MEAN_RANGE_CHART, MEDIAN_RANGE_CHART, MEAN_SIGMA_CHART, MEAN_SIGMA_CHART_VSS, INDIVIDUAL_RANGE_CHART, EWMA_CHART, LEVEY_JENNINGS_CHART, MA_CHART, MAMR_CHART, MAMS_CHART and TABCUSUM_CHART.

numsamplespersubgroup
Specifies the number of samples that make up a sample subgroup.

numdatapointsinview
Specifies the number of sample subgroups displayed in the graph at one time.

Note: The *timeincrementminutes* was found in the original intSPCTimeVariableControlChart method, but is not present in the initSPCBatchVariableControlChart method

timeincrementminutes
Specifies the normal time increment between adjacent subgroup samples.

For the Cusum charts:

mean
The process mean.

kvalue
The k-value needed for the Cusum chart.

hvalue
The h-value needed for the Cusum chart.

Once the init routine is called, the chart can be further customized using properties inherited from **SPCBaseChart**, described below.

Note - Since JavaScript/TypeScript does not support properties in the same way that C# does, you set and get the values of the properties using the “set” and “get” in front of the property name, i.e. **getBottomLabelMargin** and **setBottomLabelMargin**.

Public Static Properties

DefaultChartFontString	set/get the default font used in the table display.

Public Instance Properties

AutoLogAlarmsAsNotes	set/get to true to automatically log alarm details in the sample interval Notes record.
BottomLabelMargin	set/get an additional margin, in normalized coordinates, if only the primary graphs is displayed, allowing for the x-axis labels
ChartAlarmEmphasisMode	set/get to SPCChartBaseALARM_HIGHLIGHT_SYMBOL to highlight the process variable symbol if an alarm condition exists. Set to Set to SPCChartBase.ALARM_NO_HIGHLIGHT_SYMBOL to turn off alarm highlighting.
ChartData	set/get the object that holds the descriptive text, sampled and calculated values associated with the control chart.
ChartInitialized	Returns true if the control chart has been initialized at least once.
ChartTable	set/get the object that holds the data table information needed to display the data table along with the chart
DefaultControlLimitSigma	set/get that SPC control limits are to be calculated using the 3 sigma level standard.
EnableAlarmStatusValues	If set true enables the alarm status row of the chart table.
EnableCategoryValues	If set true enables the category or sample values rows of the data table
EnableDataToolTip	If set true enables data tooltips
EnableInputStringsDisplay	If set true enables the input string rows of the data table
EnableNotes	If set true enables the notes row of the data table
EnableNotesToolTip	If set true enables data tooltips
EnableScrollBar	If set true the scroll bar is added to the bottom of the chart.
EnableTimeValues	If set true enables the time row of the data table
EnableTotalSamplesValues	If set true enables the total of sampled values row of the data table
GraphBottomPos	set/get the bottom edge, using normalized coordinates, of the plotting area for the secondary chart
GraphStartPosX	set/get the left edge, using normalized coordinates, of

	the plotting area for both primary and secondary charts
GraphStartPosY1	set/get the top edge, using normalized coordinates, of the plotting area for the primary chart
GraphStartPosY2	set/get the top edge, using normalized coordinates, of the plotting area for the secondary chart
GraphStopPosX	set/get the right edge, using normalized coordinates, of the plotting area for both primary and secondary charts
GraphStopPosY1	set/get the bottom edge, using normalized coordinates, of the plotting area for the primary chart
GraphStopPosY2	set/get the bottom edge, using normalized coordinates, of the plotting area for the secondary chart
GraphTopTableOffset	set/get the offset of the top of the primary chart from the bottom of the data table, using normalized coordinates
HeaderStringsLevel	set/get the level of header strings to include in the chart. Use one of the SPCControlChartData header strings constants: HEADER_STRINGS_LEVEL0, HEADER_STRINGS_LEVEL1, HEADER_STRINGS_LEVEL2, or HEADER_STRINGS_LEVEL3
InterGraphMargin	set/get the margin, in normalized coordinates, between the primary and secondary charts
MultipleMouseListener	set/get the MultiMouseListener.
PrimaryChart	set/get the object that holds he the chart objects needed to display the primary chart
ScrollBarBottomPosition	set/get the bottom edge, using normalized coordinates, of the optional scroll bar
ScrollBarPixelHeight	set/get the height of the scrollbar in pixels
SecondaryChart	set/get the object that holds he the chart objects needed to display the secondary chart
SPCChartType	Specifies the control chart type: use one of the SPCControlChartData chart type constants: MEAN_RANGE_CHART, MEDIAN_RANGE_CHART, MEAN_SIGMA_CHART, MEAN_SIGMA_CHART_VSS, INDIVIDUAL_RANGE_CHART, EWMA_CHART, LEVEY_JENNINGS_CHART, MA_CHART, MAMR_CHART, MAMS_CHART, TABCUSUM_CHART, CUSTOM_ATTRIBUTE_CONTROL_CHART, PERCENT_DEFECTIVE_PARTS_CHART, FRACTION_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTS_CHART,

	NUMBER_DEFECTS_PERUNIT_CHART.
TableAlarmEmphasisMode	Set the table alarm highlighting to one of the SPCChartBase table highlight constants: ALARM_HIGHLIGHT_NONE, ALARM_HIGHLIGHT_TEXT, ALARM_HIGHLIGHT_OUTLINE, ALARM_HIGHLIGHT_BAR
TableStartPosY	set/get the top edge, using normalized coordinates, of the SPC chart table
XScaleMode	set/get whether the x-axis is time based, or numeric based.

Public Instance Functions

addAnnotation	Add a simple annotation to a data point in the specified SPC chart.
autoCalculateControlLimits	Using the current sampled data values, high, target and low control limits are calculated for both primary and secondary charts using an algorithm appropriate to the SPC chart type.
autoCalculatePrimaryControlLimits	Using the current sampled data values, high, target and low control limits are calculated for the primary chart using an algorithm appropriate to the SPC chart type.
autoCalculateSecondaryControlLimits	Using the current sampled data values, high, target and low control limits are calculated for the primary chart using an algorithm appropriate to the SPC chart type.
autoScaleChartYRange	Auto-scale the y-range of the SPC chart so that all of the sampled data and chart control limits are within the bounds of the y-axis.
autoScalePrimaryChartYRange	Auto-scale the y-range of the primary SPC chart so that all of the sampled data and chart control limits are within the bounds of the y-axis.
autoScaleSecondaryChartYRange	Auto-scale the y-range of the SPC chart so that all of the sampled data and chart control limits are within the bounds of the y-axis.
rebuildChartUsingCurrentData	Rebuild the graph taking into account the most recent data values.
resetSPCChartData	Reset the history buffers of all of the SPC data objects.
useNoTable	Specifies to create the primary and secondary charts without a table. Just the charts, chart title and optional histograms.

The SPCBatchVariableControlChart properties are documented in the QCSPCChartJSTSClassesIndex.html documentation file, located in the docs/docs/ subdirectory.

Adding New Sample Records for Variable Control Charts.

In variable control charts, each data value in the *samples* array represents a specific sample in the sample subgroup. In X-Bar R, X-Bar Sigma, and Median-Range charts, where the sample subgroup size is some fraction of the total production level, there is one value in the *samples* array for each measurement sample in the sample subgroup interval. If the production level is sixty items per hour, and the sample size is five items per hour, then the graph would be updated once an hour with five items in the *samples* array.

[JavaScript]

```
var samples = QCSPCChartTS.DoubleArray.newDoubleArrayN(5);
//  ChartCalendar initialized with current  time by default
var timestamp = new Date();
// Place sample values in array
samples.setElement(0, 0.121);      // First of five samples
samples.setElement(1, 0.212);      // Second of five samples
samples.setElement(2, 0.322);      // Third of five samples
samples.setElement(3, 0.021);      // Fourth of five samples
samples.setElement(4, 0.133);      // Fifth of five samples

// Add the new sample subgroup to the chart
// xbarrchart is a reference to the underling spc chart object

var chartdata = xbarrchart.getChartData();
if (chartdata)
  chartdata.addNewSampleRecordDateSamples(timestamp, samples);
```

[TypeScript]

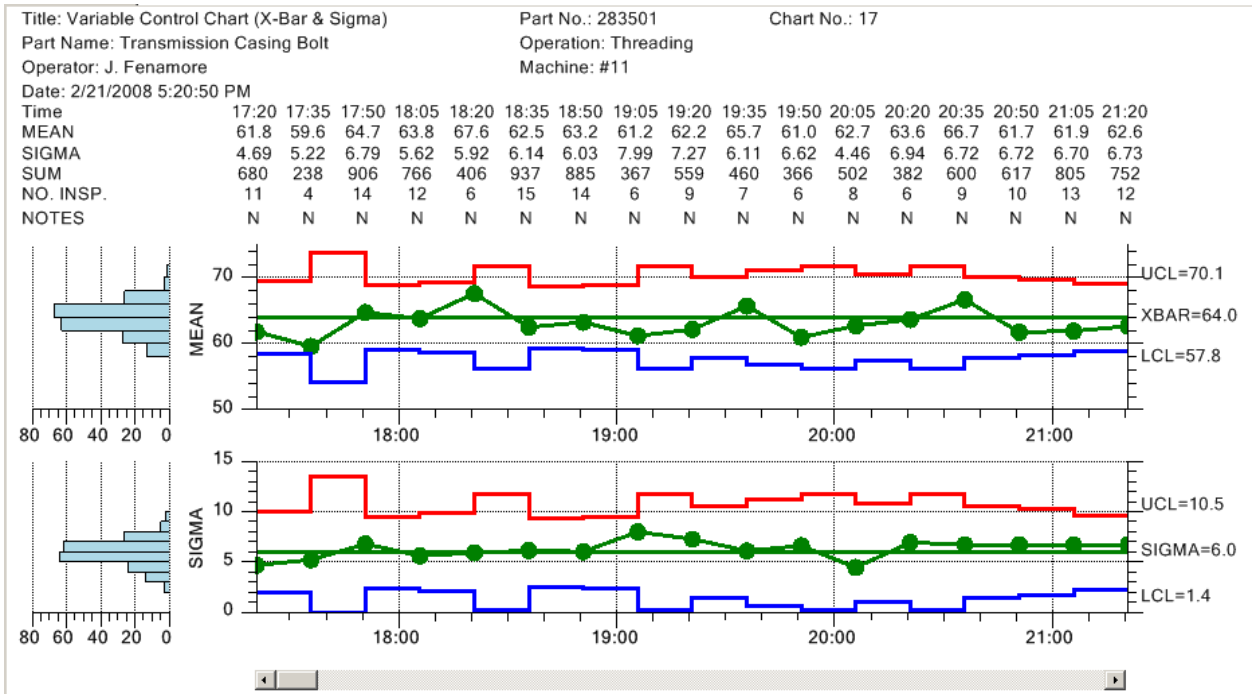
```
let samples : QCSPCChartTS.DoubleArray = new QCSPCChartTS.DoubleArrayN(5);
'  ChartCalendar initialized with current  time by default
let timestamp : Date = new Date();
' Place sample values in array


// Place sample values in array
samples.setElement(0, 0.121);      // First of five samples
samples.setElement(1, 0.212);      // Second of five samples
samples.setElement(2, 0.322);      // Third of five samples
samples.setElement(3, 0.021);      // Fourth of five samples
samples.setElement(4, 0.133);      // Fifth of five samples

// Add the new sample subgroup to the chart
// xbarrchart is a reference to the underling spc chart object
let chartdata: SPCControlChartData | null = xbarrchart.getChartData();
if (chartdata)
  chartdata.addNewSampleRecordDateSamples(timestamp, samples);
```

In an Individual-Range chart, and EWMA and MA charts that uses rational subgroup sizes of 1, the *samples* array would only have one value for each update. If the production level is sixty items per hour, with 100% sampling, the graph would be updated once a minute, with a single value in the *samples* array.

Updating MEAN_SIGMA_CHART_VSS with a variable number of samples per subgroup

The X-Bar Sigma chart also comes in a version where variable sample sizes are permitted, As in the standard variable control charts, there is one value in the *samples* array for each measurement sample in the sample subgroup interval. The difference is that the length of the *samples* array can change from update to update. It is critically import that the size of the samples array exactly matches the number of samples in the current subgroup

X-Bar Sigma Chart with variable sample size



In this case, the control chart high and low limits vary from sample interval to sample interval, depending on the number of samples in the associated sample subgroup. You can read the sample sizes along the NO. INSP row in the data table above the chart. A low number of samples in the sample subgroup make the band between the high and low

limits wider than if a higher number of samples are available. The X-Bar Sigma chart is the only variable control chart that can be used with a variable sample size.

[JavaScript]

```
// getCurrentSampleSubgroupSize is a fictional method that gets the
// current number of samples in the  sample subgroup. The value of N
// can vary from sample interval to sample interval. You must have a
// valid sample value for each element.

N = getCurrentSampleSubgroupSize();

// Size array exactly to a length of N
var samples = DoubleArray.newDoubleArrayN(N);
//  Date initialized with current  time by default
var timestamp = new Date();

// Place sample values in array
// You must have a valid sample value for each element of the array size 0..N-1
samples.setElement(0, 0.121);      // First of five samples
samples.setElement(1, 0.212);      // Second of five samples
.
.
.
samples.setElement(N-1, 0.133);    // Last of the samples in the sample subgroup

// Add the new sample subgroup to the chart
// xbarsigma is a reference to the underling spc chart object
var chartdata = xbarsigma.getChartData();
if (chartdata)
  chartdata.addNewSampleRecordDateSamples(timestamp, samples);
```

[TypeScript]

```
// getCurrentSampleSubgroupSize is a fictional method that gets the
// current number of samples in the  sample subgroup. The value of N
// can vary from sample interval to sample interval. You must have a
// valid sample value for each element.

N = this.getCurrentSampleSubgroupSize();

// Size array exactly to a length of N
let samples: DoubleArray = DoubleArray.newDoubleArrayN(N);
//  Date initialized with current  time by default
let timestamp: Date = new Date();

// Place sample values in array
// You must have a valid sample value for each element of the array size 0..N-1
samples.setElement(0, 0.121);      // First of five samples
samples.setElement(1, 0.212);      // Second of five samples
.
.
.
samples.setElement(N-1, 0.133);    // Last of the samples in the sample subgroup

// Add the new sample subgroup to the chart
// xbarsigma is a reference to the underling spc chart object
let chartdata: SPCControlChartData | null = xbarsigma.getChartData();
if (chartdata)
  chartdata.addNewSampleRecordDateSamples(timestamp, samples);
```

Measured Data and Calculated Value Tables

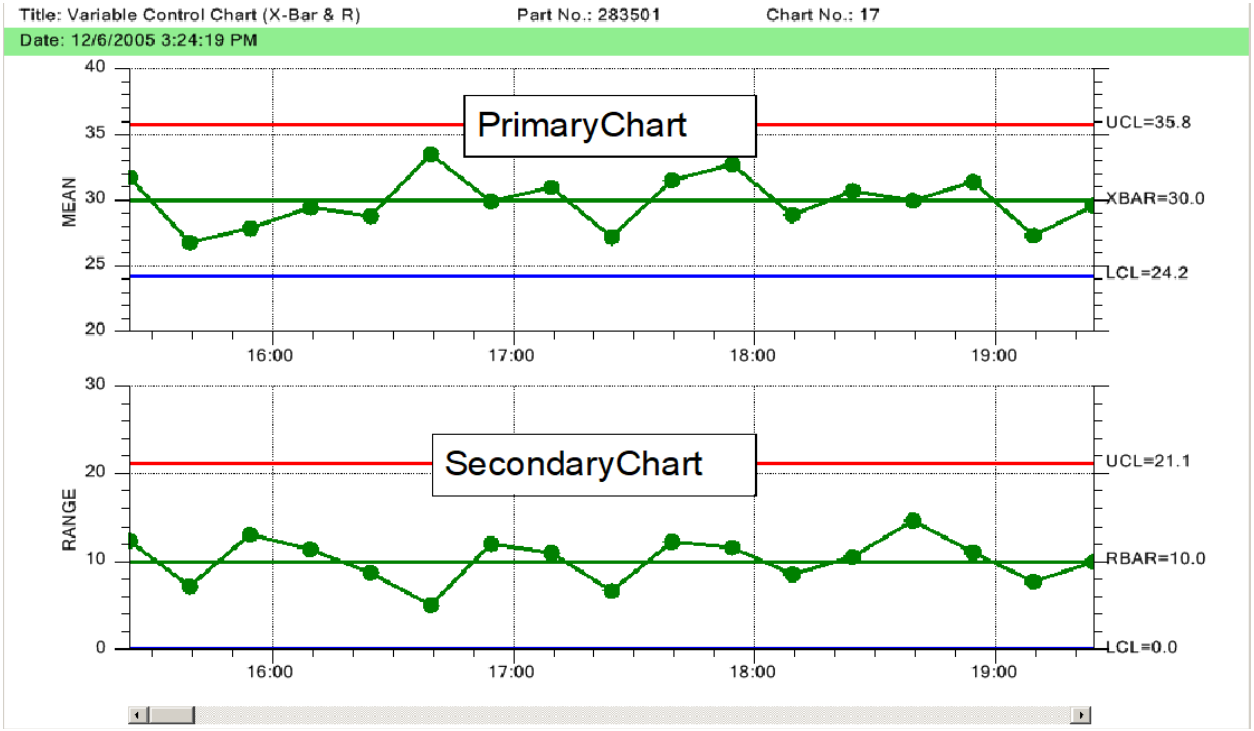
Standard worksheets used to gather and plot SPC data consist of three main parts.

- The first part is the header section, identifying the title of the chart, the monitored process, the machine operator, part number and other important information specific to the chart.
- The second part is the measurement data recording and calculation section, organized as a table where the sample data and calculated values are recorded in a neat, readable fashion.
- The third part plots the calculated SPC values for the sample group variables as a SPC chart.

The chart includes options that enable the programmer to customize and automatically include header information along with a table of the measurement and calculated data, in the SPC chart.

The following properties enable sections of the chart header and table:
:

EnableInputStringsDisplay
EnableCategoryValues
EnableCalculatedValues
EnableTotalSamplesValues
EnableNotes
EnableTotalSamplesValues
EnableTimeValues
EnableProcessCapabilityValues



In the program the code looks like the following code extracted from the VariableControlCharts.BuildXBarRChart example program

[JavaScript]

```
    var htmlcanvas = document.getElementById(canvasid);
    var spccharttype = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    var subgroupsize = 5;
    var numberpointsinview = 12;
    var charttitle = "XBar-R Example";


    var xbarrchart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
    xbarrchart.setPreferredSize(800, 600);

    xbarrchart.setGraphStopPosX(0.825);
    xbarrchart.setGraphStartPosX(0.18);

xbarrchart.setChartAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_SYM
BOL);
    xbarrchart.setEnableScrollBar(true);

    xbarrchart.setEnableCategoryValues(true);
    xbarrchart.setEnableCalculatedValues(true);
    xbarrchart.setEnableAlarmStatusValues(false);
    xbarrchart.setEnableChartToggles(true);

xbarrchart.setTableAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_BAR
);
    xbarrchar

  var  chartdata = xbarrchart.getChartData();
  if (chartdata) {
      chartdata.setTitle (charttitle);
      chartdata.setChartDescriptior("XBar-R");
      chartdata.setPartNumber ( "283501");
      chartdata.setChartNumber("17");
      chartdata.setPartName( "Transmission Casing Bolt");
      chartdata.setOperation ( "Threading");
      chartdata.setOperator("J. Fenamore");
      chartdata.setMachine("#11");
      var  today = new Date();
      chartdata.setDateString(today.toLocaleString());
      chartdata.setNotesMessage( "Control limits prepared May 10");
      chartdata.setNotesHeader("NOTES"); // row header
      chartdata.addAlarmTransitionEventListener(this);
      chartdata.setAlarmTransitionEventEnable(true);

     var  primarychart = xbarrchart.getPrimaryChart();
     var  secondarychart = xbarrchart.getSecondaryChart();
    // Set initial scale of the y-axis of the mean chart

// If you are calling autoScalePrimaryChartYRange this isn't really needed
if (primarychart)
{
  primarychart.setMinY( 0);
  primarychart.setMaxY( 40);
}

// Set initial scale of the y-axis of the range chart
// If you are calling autoScaleSecondaryChartYRange this isn't really needed
if (secondarychart)
{
  secondarychart.setMinY( 0);
  secondarychart.setMaxY(10);
}
```

```
// Display the Sampled value rows of the table
xbarrchart.setEnableInputStringsDisplay(true);
// Display the Sampled value rows of the table
xbarrchart.setEnableCategoryValues(true);
// Display the Calculated value rows of the table
xbarrchart.setEnableCalculatedValues(true);
// Display the total samples per subgroup value row
xbarrchart.setEnableTotalSamplesValues(true);
// Display the Notes row of the table
xbarrchart.setEnableNotes(true);
// Display the time stamp row of the table
xbarrchart.setEnableTimeValues(true);
```

[TypeScript]

```
    let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
    let spccharttype: number  = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    let subgroupsize: number = 5;
    let numberpointsinview: number  = 12;
    let sampleinterval: number = 15;
    let charttitle: string = "XBar-R Example";


    let xbarrchart: QCSPCChartTS.SPCBatchVariableControlChart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);


   xbarrchart.setPreferredSize(800, 600);

    xbarrchart.setGraphStopPosX(0.825);
    xbarrchart.setGraphStartPosX(0.18);

xbarrchart.setChartAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_SYM
BOL);
    xbarrchart.setEnableScrollBar(true);

    xbarrchart.setEnableCategoryValues(true);
    xbarrchart.setEnableCalculatedValues(true);
    xbarrchart.setEnableAlarmStatusValues(false);
    xbarrchart.setEnableChartToggles(true);

xbarrchart.setTableAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_BAR
);

  let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarrchart.getChartData();
  if (chartdata) {
      chartdata.setTitle (charttitle);
      chartdata.setChartDescriptior("XBar-R");
      chartdata.setPartNumber ( "283501");
      chartdata.setChartNumber("17");
      chartdata.setPartName( "Transmission Casing Bolt");
      chartdata.setOperation ( "Threading");
      chartdata.setOperator("J. Fenamore");
      chartdata.setMachine("#11");
      let  today: Date = new Date();
      chartdata.setDateString(today.toLocaleString());
      chartdata.setNotesMessage( "Control limits prepared May 10");
      chartdata.setNotesHeader("NOTES"); // row header
      chartdata.addAlarmTransitionEventListener(this);
      chartdata.setAlarmTransitionEventEnable(true);
 }

     let primarychart: QCSPCChartTS.SPCChartObjects | null =
xbarrchart.getPrimaryChart();
     let secondarychart: QCSPCChartTS.SPCChartObjects | null =
xbarrchart.getSecondaryChart();
```

```
    // Set initial scale of the y-axis of the mean chart

// If you are calling autoScalePrimaryChartYRange this isn't really needed
if (primarychart)
{
  primarychart.setMinY( 0);
  primarychart.setMaxY( 40);
}

// Set initial scale of the y-axis of the range chart
// If you are calling autoScaleSecondaryChartYRange this isn't really needed
if (secondarychart)
{
  secondarychart.setMinY( 0);
  secondarychart.setMaxY(10);
}

// Display the Sampled value rows of the table
xbarrchart.setEnableInputStringsDisplay(true);
// Display the Sampled value rows of the table
xbarrchart.setEnableCategoryValues(true);
// Display the Calculated value rows of the table
xbarrchart.setEnableCalculatedValues(true);
// Display the total samples per subgroup value row
xbarrchart.setEnableTotalSamplesValues(true);
// Display the Notes row of the table
xbarrchart.setEnableNotes(true);
// Display the time stamp row of the table
xbarrchart.setEnableTimeValues(true);
```

Process Capability Ratios and Process Performance Indices

The data table also displays any process capability statistics that you want to see. The software supports the calculation and display of the Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppu, Ppl, and Ppk process capability statistics.

In order to display process capability statistics you must first specify the process specification limits that you want the calculations based on. These are not the high and low SPC control limits calculate by this software; rather they externally calculated limits based on the acceptable tolerances allowed for the process under measure. Set the lower specification limit (LSL) and upper specification limit (USL) using the **chartdata.ProcessCapabilityLSLValue** and **chartdata.ProcessCapabilityUSLValue** properties of the chart. The code below is from the VariableControlCharts.BuildXBarRChart example.

[JavaScript]

```
 var chartdata = xbarrchart.getChartData();
  if (chartdata) {
     chartdata.setProcessCapabilityLSLValue(27);
     chartdata.setProcessCapabilityUSLValue(35);
  }
```

[TypeScript]

```
 let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarrchart.getChartData();
  if (chartdata) {
     chartdata.setProcessCapabilityLSLValue(27);
     chartdata.setProcessCapabilityUSLValue(35);
```

```
  }
```

Use the **chartdata.addProcessCapabilityValue** method to specify exactly which process capability statistics you want to see in the table. Use one of the **SPCProcessCapabilityRecord** constants below to specify the statistics that you want displayed.

SPC_CP_CALC	Constant value Cp calculation
SPC_CPL_CALC	Constant value Cpl calculation.
SPC_CPU_CALC	Constant value Cpu calculation.
SPC_CPK_CALC	Constant value Cpk calculation.
SPC_CPM_CALC	Constant value Cpm calculation.
SPC_PP_CALC	Constant value Pp calculation.
SPC_PPL_CALC	Constant value Ppl calculation.
SPC_PPU_CALC	Constant value Ppu calculation.
SPC_PPK_CALC	Constant value PPK calculation.

The code below is from the VariableControlCharts.BuildXBarSigmaChart example.

[JavaScript]

```
 var chartdata = xbarsigmachart.getChartData();
  if (chartdata) {
chartdata.addProcessCapabilityValue(QCSPCChartTS.SPCProcessCapabilityRecord.SPC_CP
K_CALC);
chartdata.addProcessCapabilityValue(QCSPCChartTS.SPCProcessCapabilityRecord.SPC_CP
M_CALC);
chartdata.addProcessCapabilityValue(QCSPCChartTS.SPCProcessCapabilityRecord.SPC_PP
K_CALC);
  }
```

[TypeScript]

```
 let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarsigmachart.getChartData();
  if (chartdata) {
chartdata.addProcessCapabilityValue(QCSPCChartTS.SPCProcessCapabilityRecord.SPC_CP
K_CALC);
chartdata.addProcessCapabilityValue(QCSPCChartTS.SPCProcessCapabilityRecord.SPC_CP
M_CALC);
chartdata.addProcessCapabilityValue(QCSPCChartTS.SPCProcessCapabilityRecord.SPC_PP
K_CALC);
  }
```

This selection will add three rows to the data table, one row each for the Cpk, Cpm and Ppk process capability statistics. Once these steps are carried out, the calculation and display of the statistics is automatic.

Formulas Used in Calculating the Process Capability Ratios

The formulas used in calculating the process capability statistics vary. We use the formulas found in the textbook. “Introduction to Statistical Quality Control” by Douglas C. Montgomery, John Wiley and Sons, Inc. 2001.

SPC Control Chart Nomenclature

USL = Upper Specification Limit

LSL = Lower Specification Limit

N = number of samples intervals

M = number of samples per subgroup

Tau = Midpoint between USL and LSL = ½ * (LSL + USL)

$\overline{X}$ = XDoubleBar - Mean of sample subgroup means (also called the grand average)

$\overline{R}$ = RBar – Mean of sample subgroup ranges

S = Sigma – sample standard deviation – all samples from all subgroups are used to calculate the standard deviation S.

$\overline{S}$ = SigmaBar – Average of sample subgroup sigma’s. Each sample subgroup has a calculated standard deviation and the SigmaBar value is the mean of those subgroup standard deviations.

d_2 = a constant tabulated in every SPC textbook for various sample sizes.

By convention, the quantity RBar/d_2 is used to estimate the process sigma for the Cp, Cpl and Cpu calculations

MINIMUM – a function that returns the lesser of two arguments

SQRT – a function returning the square root of the argument.

Process Capability Ratios (Cp, Cpl, Cpu, Cpk and Cpm)

$$C_p = \frac{(USL - LSL)}{(6 * \bar{R} / d_2)}$$

$$C_{pl} = \frac{(\bar{\bar{X}} - LSL)}{(3 * \bar{R}/d_2)}$$

$$C_{pu} = \frac{(USL - \bar{\bar{X}})}{(3 * \bar{R}/d_2)}$$

$$C_{pk} = MINIMUM\,(C_{pl}, C_{pu})$$

$$C_{pm} = \frac{C_p}{\sqrt{1 + V^2}}$$

where

$$V = \frac{(\bar{\bar{X}} - \tau)}{S}$$

Process Performance Indices (Pp, Ppl, Ppu, Ppk)

$$P_p = \frac{(USL - LSL)}{(6 * S)}$$

$$P_{pl} = \frac{(\bar{\bar{X}} - LSL)}{(3 * S)}$$

$$P_{pu} = \frac{(USL - \bar{\bar{X}})}{(3 * S)}$$

$$P_{pk} = MINIMUM\,(P_{pl}, P_{pu})$$

The major difference between the Process Capability Ratios (Cp, Cpl, Cpu, Cpk) and the Process Performance Indices (Pp, Ppl, Ppu, Ppk) is the estimate used for the process sigma. The Process Capability Ratios use the estimate ($RBar/d_2$) and the Process Performance Indices uses the sample standard deviation S. If the process is in control, then Cp vs Pp and Cpk vs Ppk should returns approximately the same values, since both

(RBar/d_2) and the sample sigma S will be good estimates of the overall process sigma. If the process is NOT in control, then ANSI (American National Standards Institute) recommends that the Process Performance Indices (Pp, Ppl, Ppu, Ppk) be used.

Table Strings

The input header strings display has four sub-levels that display increasing levels of information. The input header strings display level is set using the charts HeaderStringsLevel property. Strings that can be displayed are: Title, PartNumber, ChartNumber, DateString, PartName, Operation, Machine, SpecificationLimits, Gage, UnitOfMeasure, ZeroEquals and DateString. The four levels and the information displayed is listed below:

HEADER_STRINGS_LEVEL0	Display no header information
HEADER_STRINGS_LEVEL1	Display minimal header information: Title, PartNumber, ChartNumber, DateString
HEADER_STRINGS_LEVEL2	Display most header strings: Title, PartNumber, ChartNumber, PartName, Operation, Operator, Machine, DateString
HEADER_STRINGS_LEVEL3	Display all header strings: Title, PartNumber, ChartNumber, DateString, PartName, Operation, Machine, SpecificationLimits, Gage, UnitOfMeasure, ZeroEquals and DateString

The example program VariableControlCharts.BuildXBarRChart demonstrates the use of the **HeaderStringsLevel** property. The example below displays a minimum set of header strings (HeaderStringsLevel = SPCControlChartData.HEADER_STRINGS_LEVEL1).

[JavaScript]

```
 var  chartdata = xbarrchart.getChartData();
 if (chartdata) {
// Set the strings used in the header section of the table
chartdata.setTitle("Variable Control Chart (X-Bar & R)");
chartdata.setPartNumber("283501");
chartdata.setChartNumber("17");
var  today = new Date();
chartdata.setDateString(today.toLocaleString());
}
xbarrchart.setHeaderStringsLevel(QCSPCChartTS.SPCControlChartData.HEADER_STRINGS_L
EVEL1);
```

[TypeScript]

```
 let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarrchart.getChartData();
 if (chartdata) {
// Set the strings used in the header section of the table
```

```
chartdata.setTitle("Variable Control Chart (X-Bar & R)");
chartdata.setPartNumber("283501");
chartdata.setChartNumber("17");
let  today: Date = new Date();
chartdata.setDateString(today.toLocaleString());
}
xbarrchart.setHeaderStringsLevel(QCSPCChartTS.SPCControlChartData.HEADER_STRINGS_L
EVEL1);
```

The example below displays a maximum set of header strings (HeaderStringsLevel = SPCControlChartData.HEADER_STRINGS_LEVEL3).

[JavaScript]

```
 var  chartdata = xbarrchart.getChartData();
 if (chartdata) {
// Set the strings used in the header section of the table
chartdata.setTitle( "Variable Control Chart (X-Bar & R)");
chartdata.setPartNumber( "283501");
chartdata.setChartNumber("17");
chartdata.setPartName("Transmission Casing Bolt");
chartdata.setOperation("Threading");
chartdata.setSpecificationLimits("");
chartdata.setTheOperator("J. Fenamore");
chartdata.setMachine("#11");
chartdata.setGage("#8645";
chartdata.setUnitOfMeasure("0.0001 inch");
var  today= new Date();
chartdata.setDateString(today.toLocaleString());


xbarrchart.setHeaderStringsLevel
(QCSPCChartTS.SPCControlChartData.HEADER_STRINGS_LEVEL3);
```

[TypeScript]

```
 let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarrchart.getChartData();
 if (chartdata) {
  // Set the strings used in the header section of the table
  chartdata.setTitle( "Variable Control Chart (X-Bar & R)");
  chartdata.setPartNumber( "283501");
  chartdata.setChartNumber("17");
  chartdata.setPartName("Transmission Casing Bolt");
  chartdata.setOperation("Threading");
  chartdata.setSpecificationLimits("");
  chartdata.setTheOperator("J. Fenamore");
  chartdata.setMachine("#11");
  chartdata.setGage("#8645";
  chartdata.setUnitOfMeasure("0.0001 inch");
  let  today: Date = new Date();
  chartdata.setDateString(today.toLocaleString());
}
xbarrchart.setHeaderStringsLevel
(QCSPCChartTS.SPCControlChartData.HEADER_STRINGS_LEVEL3);
```

The identifying string displayed in front of the input header string can be any string that you want, including non-English language strings. For example, if you want the input header string for the Title to represent a project name:

Project Name: Project XKYZ for PerQuet

Set the properties:

[JavaScript]

```
var  chartdata = xbarrchart.getChartData();
if (chartdata) {
  chartdata.setTitle( "Project XKYZ for PerQuet");
  chartdata.setTitleHeader("Project Name:");
}
```

[TypeScript]

```
 let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarrchart.getChartData();
 if (chartdata) {
  chartdata.setTitle( "Project XKYZ for PerQuet");
  chartdata.setTitleHeader("Project Name:");
}
```

Change other header strings using the **ChartData** properties listed below.

- TitleHeader
- PartNumberHeader
- ChartNumberHeader
- PartNameHeader
- OperationHeader
- OperatorHeader
- MachineHeader
- DateHeader
- SpecificationLimitsHeader
- GageHeader
- UnitOfMeasureHeader
- ZeroEqualsHeader
- NotesHeader

Even though the input header string properties have names like Title, PartNumber, ChartNumber, etc., those names are arbitrary. They are really just placeholders for the strings that are placed at the respective position in the table. You can display any combination of strings that you want, rather than the ones we have selected by default, based on commonly used standardized SPC Control Charts.

Table Background Colors

The **ChartTable** property of the chart has some properties that can further customize the chart. The default table background uses the accounting style green-bar striped background. You can change this using the **ChartTable.TableBackgroundMode**

property**.** Set the value to one of the TableBackgroundMode constants in the class **SPCGeneralizedTableDisplay**:

TABLE_NO_COLOR_BACKGROUND	Constant specifies that the table does not use a background color.
TABLE_SINGLE_COLOR_BACKGROUND	Constant specifies that the table uses a single color for the background (backgroundColor1)
TABLE_STRIPED_COLOR_BACKGROUND	Constant specifies that the table uses horizontal stripes of color for the background (backgroundColor1 and backgroundColor2)
TABLE_SINGLE_COLOR_BACKGROUND_GRIDCELL	Constant specifies that the table uses a grid background, with backgroundColor1 the overall background color and backgroundColor2 the color of the grid lines.

Extracted from the VariableControlCharts.BuildIRChart example program

[JavaScript]

```
    var charttable = irchart.getChartTable();

    if (charttable)
    {

charttable.setTableBackgroundMode(   QCSPCChartTS.SPCGeneralizedTableDisplay.TABLE
_STRIPED_COLOR_BACKGROUND);
        charttable.setBackgroundColor1( QCSPCChartTS.ChartColor.BISQUE);

charttable.setBackgroundColor2( QCSPCChartTS.ChartColor.LIGHTGOLDENRODYELLOW);
    }
```

[TypeScript]

```
    let charttable: QCSPCChartTS.SPCGeneralizedTableDisplay | null =
irchart.getChartTable();
    if (charttable)
    {

charttable.setTableBackgroundMode(   QCSPCChartTS.SPCGeneralizedTableDisplay.TABLE
_STRIPED_COLOR_BACKGROUND);
        charttable.setBackgroundColor1( QCSPCChartTS.ChartColor.BISQUE);

charttable.setBackgroundColor2( QCSPCChartTS.ChartColor.LIGHTGOLDENRODYELLOW);
    }
```

Extracted from the VariableControlCharts.BuildMedianRangeChart example program

[JavaScript]

```
    var charttable = medianrangechart.getChartTable();

    if (charttable)
    {

charttable.setTableBackgroundMode(   QCSPCChartTS.SPCGeneralizedTableDisplay.TABLE
_SINGLE_COLOR_BACKGROUND );
        charttable.setBackgroundColor1( QCSPCChartTS.ChartColor.LIGHTGRAY);
    }
```

[TypeScript]

```
    let charttable: QCSPCChartTS.SPCGeneralizedTableDisplay | null =
medianrangechart.getChartTable();
    if (charttable)
    {

charttable.setTableBackgroundMode(   QCSPCChartTS.SPCGeneralizedTableDisplay.TABLE
_SINGLE_COLOR_BACKGROUND );
        charttable.setBackgroundColor1( QCSPCChartTS.ChartColor.LIGHTGRAY);
    }
```

From VariableControlCharts.BuildXBarSigmaChart example program

[JavaScript]

```
    var charttable = xbarsigmachart.getChartTable();

    if (charttable)
    {
        charttable.setTableBackgroundMode(QCSPCChartTS.SPCGeneralizedTableDisplay.
TABLE_NO_COLOR_BACKGROUND );
    }
```

[TypeScript]

```
    let charttable: QCSPCChartTS.SPCGeneralizedTableDisplay | null =
xbarsigmachart.getChartTable()

    if (charttable)
    {
        charttable.setTableBackgroundMode(QCSPCChartTS.SPCGeneralizedTableDisplay.
TABLE_NO_COLOR_BACKGROUND );
    }
```

From VariableControlCharts.BuildLeveyJenninsChart

[JavaScript]

```
    var charttable = leveyjenningschart.getChartTable();

    if (charttable)
    {
```

```
charttable.setTableBackgroundMode(   QCSPCChartTS.SPCGeneralizedTableDisplay.TABLE
_SINGLE_COLOR_BACKGROUND );
        charttable.setBackgroundColor1( QCSPCChartTS.ChartColor.LIGHTGRAY);
    }
```

[TypeScript]

```
    let charttable: QCSPCChartTS.SPCGeneralizedTableDisplay | null =
leveyjenningschart.getChartTable()

    if (charttable)
    {

charttable.setTableBackgroundMode(   QCSPCChartTS.SPCGeneralizedTableDisplay.TABLE
_SINGLE_COLOR_BACKGROUND );
        charttable.setBackgroundColor1( QCSPCChartTS.ChartColor.LIGHTGRAY);
    }
```

From VariableControlCharts.BuildEWMAChart

[JavaScript]

```
    var charttable = ewmachart.getChartTable();

    if (charttable)
    {

charttable.setTableBackgroundMode(   QCSPCChartTS.SPCGeneralizedTableDisplay.TABLE
_SINGLE_COLOR_BACKGROUND_GRIDCELL );
        charttable.setBackgroundColor1( QCSPCChartTS.ChartColor.WHITE);
        charttable.setBackgroundColor2( QCSPCChartTS.ChartColor.GRAY);
    }
```

[TypeScript]

```
    let charttable: QCSPCChartTS.SPCGeneralizedTableDisplay | null =
ewmachart.getChartTable();
    if (charttable)
    {
charttable.setTableBackgroundMode(   QCSPCChartTS.SPCGeneralizedTableDisplay.TABLE
_SINGLE_COLOR_BACKGROUND_GRIDCELL );
        charttable.setBackgroundColor1( QCSPCChartTS.ChartColor.WHITE);
        charttable.setBackgroundColor2( QCSPCChartTS.ChartColor.GRAY);
    }
```

Table and Chart Fonts

There are a large number of fonts that you have control over, both the fonts in the table and the fonts in the chart. The programmer can select a default font (as in the case of a non-US character set), or select individual fonts for different elements of the table and charts. If you change a font size,

Table Fonts

The table fonts are accessed through the charts **ChartTable** property. Below is a list of accessible table fonts:

Property Name	**Description**
TimeLabelFont	The font used in the display of time values in the table.
SampleLabelFont	The font used in the display of sample numeric values in the table.
CalculatedLabelFont	The font used in the display of calculated values in the table.
StringLabelFont	The font used in the display of header string values in the table.
NotesLabelFont	The font used in the display of notes values in the table.

[JavaScript]

```
var charttable = irchart.getChartTable();
if (charttable)
{
  charttable.setSampleLabelFont( QCSPCChartTS.ChartFont.newChartFont("Arial",
QCSPCChartTS.ChartFont.REGULAR, 14));
  charttable.setCalculatedLabelFont( QCSPCChartTS.ChartFont.newChartFont("Arial",
QCSPCChartTS.ChartFont.REGULAR, 14));
}
```

[TypeScript]

```
let charttable: QCSPCChartTS.SPCGeneralizedTableDisplay | null =
irchart.getChartTable();
if (charttable)
{
  charttable.setSampleLabelFont( QCSPCChartTS.ChartFont.newChartFont("Arial",
QCSPCChartTS.ChartFont.REGULAR, 14));
  charttable.setCalculatedLabelFont( QCSPCChartTS.ChartFont.newChartFont("Arial",
QCSPCChartTS.ChartFont.REGULAR, 14));
}
```

The **SPCGeneralizedTableDisplay** class has a static property, **SPCGeneralizedTableDisplay.DefaultTableFont**, that sets the default Font. Use this if you want to establish a default font for all of the text in a table. This static property must be set BEFORE the chart is instantiated. All of the other fonts in the class are initialized to this default font. The DefaultTableFont is initialized internally using the following code:

```
public static defaultTableFont: ChartFont =
ChartFont.newChartFont3(SPCChartStrings.getString(SPCStringEnum.chartFont),
ChartFont.PLAIN, 14);
```

so you can see that it looks to the value of the chartFont in SPCChartStrings lookup table, obtained by calling SPCChartStrings.getString(SPCStringEnum.chartFont). So if you change the font name in the SPCChartStrings class, all fonts in the software will use that font type.

Extracted from the example VariableControlCharts.BuildIRChart

[JavaScript]

```
QCSPCChartTS.SPCGeneralizedTableDisplay.setDefaultTableFont(
    QCSPCChartTS.ChartFont.newChartFont("Arial",  QCSPCChartTS.ChartFont.REGULAR,
14));
// Initialize the SPCBatchVariableControlChart
   var  irchart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
Subgroupsize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
.
.
.
var charttable = irchart.getChartTable();
if (charttable)
{
   charttable.setSampleLabelFont( QCSPCChartTS.ChartFont.newChartFont("Arial",
QCSPCChartTS.ChartFont.REGULAR, 14));
   charttable.setCalculatedLabelFont( QCSPCChartTS.ChartFont.newChartFont("Arial",
QCSPCChartTS.ChartFont.REGULAR, 14));
}
```

[TypeScript]

```
QCSPCChartTS.SPCGeneralizedTableDisplay.setDefaultTableFont(
    QCSPCChartTS.ChartFont.newChartFont("Arial",  QCSPCChartTS.ChartFont.REGULAR,
14));
// Initialize the SPCBatchVariableControlChart
   let irchart: QCSPCChartTS.SPCBatchVariableControlChart=
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
Subgroupsize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
.
.
.
let charttable: QCSPCChartTS.SPCGeneralizedTableDisplay | null =
irchart.getChartTable();
if (charttable)
{
  charttable.setSampleLabelFont( QCSPCChartTS.ChartFont.newChartFont("Arial",
QCSPCChartTS.ChartFont.REGULAR, 14));
  charttable.setCalculatedLabelFont( QCSPCChartTS.ChartFont.newChartFont("Arial",
QCSPCChartTS.ChartFont.REGULAR, 14));
}
```

Chart Fonts

There are chart fonts for different chart objects (axes, titles, toolips, annoations, etc.) in the **SPCChartObjects** class. They establish the default fonts for related chart objects. Since the fonts are many different sizes, and styles, only the font name is made a static variable. It is defined internally using the following code.

```
public static defaultChartFontString: string =
SPCChartStrings.getString(SPCStringEnum.chartFont);
```

You can set the default font name by setting using either the static DefaultChartFontString property of SPCChartBase, or by changing the font name in the SPCChartStrings table, from which the default value is pulled.

```
QCSPCChartTS.SPCChartBase.setDefaultChartFontString("Arial");
```

or

```
QCSPCChartTS.SPCChartStrings.setString(QCSPCChartTS.SPCStringEnum.chartFont,"Arial
")
```

The method using the SPCChartStrings table will affect all fonts used in the software, chart and table alike, while changing the SPCChartBase.DefaultChartFontString property affects only the chart (but not the table) text objects.

Property name	Description	Default size
AxisLabelFont	The font used to label the x- and y- axes.	12
AxisTitleFont	The font used for the axes titles.	12
MainTitleFont	The font used for the chart title.	18
SubheadFont	The font used for the chart subhead.	14
ToolTipFont	The tooltip font.	12
FooterFont	The font used for the chart footer.	12
AnnotationFont	The annotation font.	12
ControlLimitLabelFont	The font used to label the control limits	12
LegendFont	The font used to label the legend items	12

[JavaScript]

```
QCSPCChartTS.SPCTimeVariableControlChart.setDefaultChartFontString("Times");
// Initialize the SPCBatchVariableControlChart
   var  irchart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
.
.
```

[TypeScript]

```
QCSPCChartTS.SPCTimeVariableControlChart.setDefaultChartFontString("Times");
// Initialize the SPCBatchVariableControlChart
   let irchart: QCSPCChartTS.SPCBatchVariableControlChart=
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
```

These properties establish the default fonts for a group of objects within a chart. For example, a chart will normally use the same x- and y-axis label fonts. You can still change the individual fonts for an individual object in a specific chart. For example, if in the Primary Chart you want the x-axis label font to be size 10, and the y-axis label font to be size 14, you can set them individually after the charts Init.. method has been called.

[JavaScript]

```
var ewmachart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
```

```
.
.
var primarychart = ewmachart.getPrimaryChart();
if (primarychart)
{

primarychart.getXAxisLab().setTextFont(QCSPCChartTS.ChartFont.newChartFont3("Times
",   QCSPCChartTS.ChartFont.REGULAR, 10));

primarychart.getYAxisLab().setTextFont(QCSPCChartTS.ChartFont.newChartFont3("Times
", QCSPCChartTS.ChartFont.REGULAR, 14));
}
```

[TypeScript]

```
let ewmachart: QCSPCChartTS.SPCBatchVariableControlChart=
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
.
.
let primarychart: QCSPCChartTS.SPCChartObjects | null =
ewmachart.getPrimaryChart();
if (primarychart)
{

primarychart.getXAxisLab().setTextFont(QCSPCChartTS.ChartFont.newChartFont3("Times
",   QCSPCChartTS.ChartFont.REGULAR, 10));

primarychart.getYAxisLab().setTextFont(QCSPCChartTS.ChartFont.newChartFont3("Times
", QCSPCChartTS.ChartFont.REGULAR, 14));
}
```

Font Size (using setPreferredSize)

The fonts all have an initial, default size, usually between 10 and 14 device units, regardless of the window size. If you place the chart in a small window, the fonts size may be too a large for the window and the text within the window may overlap. If you place the chart in a full screen window, there may be a lot of wasted whitespace in the chart. The resolution of the output device will also effect the apparent size of the text, with small phone type displays in portrait mode showing crowded overlapping text, and large computer screens or tablets showing a lot of whitespace. This is the main reason that many web pages have multiple variants the browser can choose from, depending on the resolution of the browser display device. The display output orientation, whether portrait (usually mobile devices) or landscape, can also effect the crowding of text.

In all versions of QCSPCChart we have included a function, setPreferredSize, which allows a degree of device independence when displaying text in charts. It establishes a relative screen size, against which you size your text. In all of our examples we explicitly set a preferred size of (800x600) device units, representing a typical Canvas window size within an HTML page.

```
var spcchart = QCSPCChartTS.SPCBatchAttributeControlChart.newSPCBatchAttributeCont
rolChartChartTypeSubgroupSize(htmlcanvas, spccharttype, 1, subgroupsize, numberpoi
ntsinview);

var spcchart .setPreferredSize(800, 600);
```

All of the fonts used in the software are initialized to a size which works pretty good on average for a page size of (800x600). If you place the chart in a much larger Canvas (2400x1800) for example, but leave the **PreferredSize** property still sized for (800x600), all fonts will be scaled up by a factor of 3, taking into account the new, larger display window size. The software assumes that you want the relative size of the charts text to look the same, regardless of whether you displaying the chart in a (800x600) or a (2400x1800) output device. So, with a **PreferredSize** of (800x600), but an actual display size of (2400x1800), a size 12 font will temporarily be increased to a size 36 font for the purposes of output scaling. If you were to call **setPreferredSize**(2400, 1800) in your program, the software would output a size 12 font to the much larger display, making it appear very small with respect to the overall chart size. This simple resizing works best if the aspect ratio (W/H) of the display screen matches the aspect ratio of the **PreferredSize** dimensions. If they don't match, for example a change from landscape orientation to portrait mode, the font size scaling factor will use the scale factor (width or height) which changes the least when calculating (preferred size)/(actual size ratios) for width and height. The actual scale factor calculation routine in the software looks like:

```
    public calcResizedWindowFontMultiplier(preferreddim: ChartDimension,
        actualdim: ChartDimension): number {
        let changewidth: number = actualdim.getWidth() / preferreddim.getWidth();
        let changeheight: number = actualdim.getHeight() /
preferreddim.getHeight();
        let result: number = 1.0;
        if (this.resizeMode == ChartConstants.NO_RESIZE_OBJECTS)
            result = 1.0;
        else if (this.resizeMode == ChartConstants.AUTO_RESIZE_OBJECTS)
            result = Math.min(changewidth, changeheight);
        else if (this.resizeMode == ChartConstants.MANUAL_RESIZE_OBJECTS)
            result = this.resizeMultiplier; // return manually set value
        return result;
    }
```

In that code you will see a resizeMode property, which you can set to either ChartConstants.AUTO_RESIZE_OBJECTS (the default mode), or ChartConstants.NO_RESIZE_OBJECTS, using the setResizeMode method of the chart.

```
spcchart.setResizeMode( QCSPCChartTS.ChartConstants.NO_RESIZE_OBJECTS);
```

If the resizeMode is explicitly set to ChartConstants.NO_RESIZE_OBJECTS, the scaling factor for up- or down-scaling font size will always be 1.

SPC Charts without a Table

If you don’t want any of the items we have designated table items, just call the **useNoTable** method. That method removes all of the table items, and displays the primary and/or secondary charts with a simple title and optional histograms.

This initialization method initializes the most important values in the creation of a SPC chart.

[TypeScript]

```
public useNoTable(primary: boolean, secondary: boolean,
        histograms: boolean, title: string)
```

Parameters

primarychart
Set to true to display primary chart.

secondarychart
Set to true to display secondary chart.

histograms
Set to true to display chart histograms

title
Specifies the title for the charts

[JavaScript] / [TypeScript]

```
    xbarrchart.useNoTable(true, true, true,"XBar-R Chart");
```

Chart Position

If the SPC chart does not include frequency histograms on the left (they take up about 20% of the available chart width), you may want to adjust the left and right edges of the chart using the **GraphStartPosX** and **GraphStopPlotX** properties to allow for more room in the display of the data. This also affects the table layout, because the table columns must line up with the chart data points.

[JavaScript] / [TypeScript]

```
xbarrchart.setGraphStartPosX( 0.1); // start here
xbarrchart.setGraphStopPosX( 0.875);  // end here
```

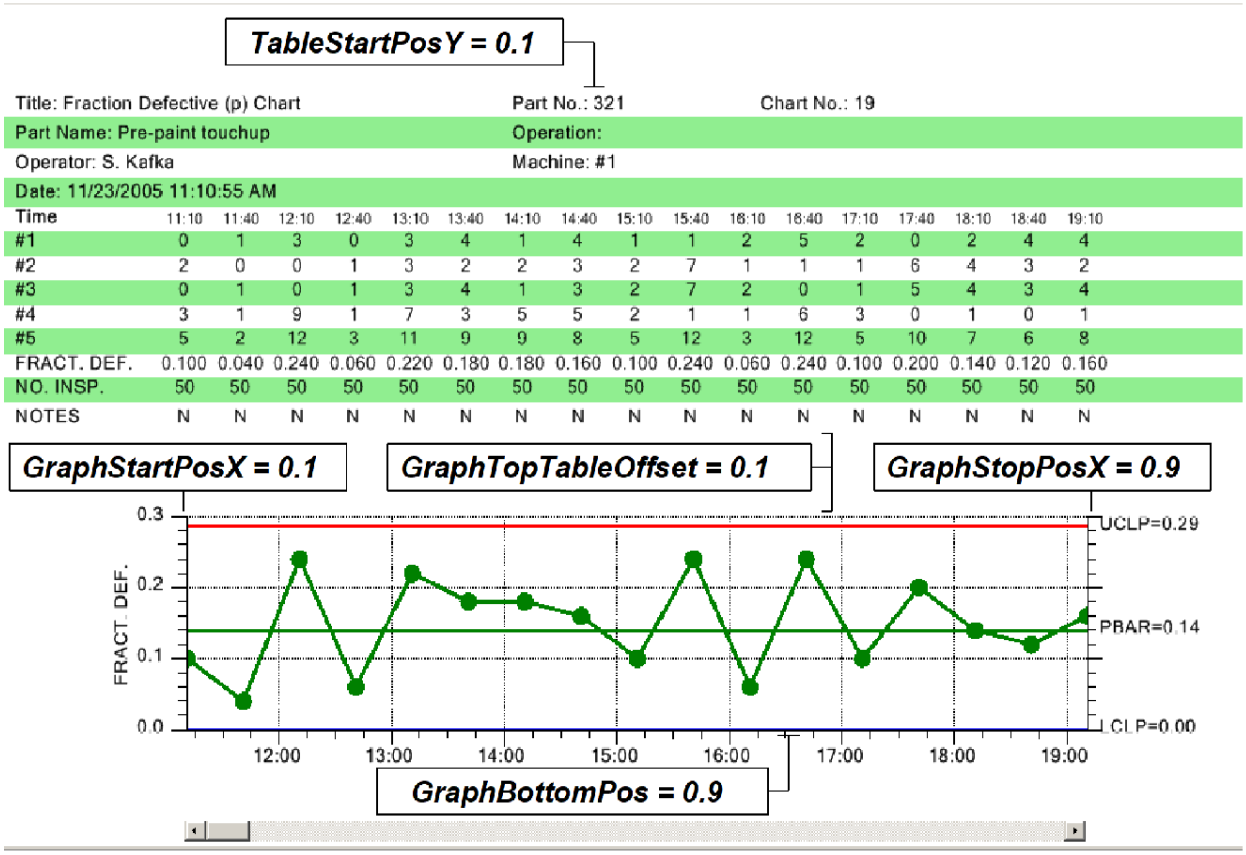
There is not much flexibility positioning the top and bottom of the chart. Depending on the table items enabled, the table starts at the position defined the **TableStartPosY** property, and continues until all of the table items are displayed. It then offsets the top of the primary chart with respect to the bottom of the table by the value of the property **GraphTopTableOffset**. The top of the secondary chart offsets from the bottom of the primary chart by the amount of the property **InterGraphMargin**. The value of the

property **GraphBottomPos** defines the bottom of the graph. The default values for these properties are:

[JavaScript] / [TypeScript]

```
xbarrchart.setTableStartPosY( 0.00);
xbarrchart.setGraphTopTableOffset(0.02);
xbarrchart.setInterGraphMargin(0.075);
xbarrchart.setGraphBottomPos(0.925);
```

The picture below uses different values for these properties in order to emphasize the affect that these properties have on the resulting chart.



SPC Control Limits

There are two ways to set the SPC control limit for a chart. The first explicitly sets the limits to values that you calculate on your own, because of some analysis that a quality

engineer does on previously collected data. The second auto-calculates the limits using the algorithms supplied in this software.

The quick way to set the limit values and limit strings is to use the charts **chartdata.setControlLimitValues** and **chartdata.setControlLimitStrings** methods. This method only works for the default +-3-sigma level control limits, and not any others you may have added using the charts **addAdditionalControlLimit** method discussed in the *Multiple Control Limits* section. The data values in the *controllimitvalues* and *controllimitstrings* arrays used to pass the control limit information must be sorted in the following order:

[SPC_PRIMARY_CONTROL_TARGET,
SPC_PRIMARY_LOWER_CONTROL_LIMIT,
SPC_PRIMARY_UPPER_CONTROL_LIMIT,
SPC_SECONDARY_CONTROL_TARGET,
SPC_SECONDARY_LOWER_CONTROL_LIMIT,
SPC_SECONDARY_UPPER_CONTROL_LIMIT]

[JavaScript]

```
var chartdata = xbarrchart.getChartData();

var controllimitvalues = [42, 30, 53, 10, 0, 22];
if (chartdata)
    chartdata.setControlLimitValues(controllimitvalues);;


var controllimitstrings = ["XBAR","LCL", "UCL","RBAR","LCL","UCL"];

if (chartdata)
  chartdata.setControlLimitStrings(controllimitstrings);
```

[TypeScript]

```
let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarrchart.getChartData();
let controllimitvalues: number[] = [42, 30, 53, 10, 0, 22];
if (chartdata)
    chartdata.setControlLimitValues(controllimitvalues);;


let controllimitstrings: string [] = ["XBAR","LCL", "UCL","RBAR","LCL","UCL"];

if (chartdata)
  chartdata.setControlLimitStrings(controllimitstrings);
```

You can also set the control limit values and control limit text one value at a time using the **chartdata.setControlLimitValue** and **chartdata.setControlLimitStrings** methods.

A more complicated way to set the control limits explicitly is to first grab a reference to the **SPCControlLimitRecord** for a given control limit, and then change the value of that control limit, and the control limit text, if desired. The example below sets the control limit values and text for the three control limits (target value, upper control limit, and

lower control limit) of the primary chart, and the three control limit values for the secondary chart.

[JavaScript]

```
//target control limit primary chart
 var chartdata = xbarrchart.getChartData();

 var primarytarget =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_CONTR
OL_TARGET);
      primarytarget.setControlLimitValue ( 30);
      primarytarget.setControlLimitText("XBAR");

      //lower control limit primary chart
      var primarylowercontrollimit =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_LOWER
_CONTROL_LIMIT);
      primarylowercontrollimit.setControlLimitValue(24);
      primarylowercontrollimit.setControlLimitText("LCL");

      //upper control limit primary chart
      var primaryuppercontrollimit =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_UPPER
_CONTROL_LIMIT);
      primaryuppercontrollimit.setControlLimitValue( 36);
      primaryuppercontrollimit.setControlLimitText( "UCL");

      // Set control limits for secondary chart

      //target control limit secondary chart
      var secondarytarget =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_SECONDARY_CON
TROL_TARGET);
      secondarytarget.setControlLimitValue (10);
      secondarytarget.setControlLimitText("RBAR");

      //lower control limit secondary chart
      var secondarylowercontrollimit=
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_SECONDARY_LOW
ER_CONTROL_LIMIT);
      secondarylowercontrollimit.setControlLimitValue (0);
      secondarylowercontrollimit.setControlLimitText  ("LCL");

      //upper control limit secondary chart
      var secondaryuppercontrollimit =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_SECONDARY_UPP
ER_CONTROL_LIMIT);
      secondaryuppercontrollimit.setControlLimitValue (22);
      secondaryuppercontrollimit.setControlLimitText ("UCL");
```

[TypeScript]

```
//target control limit primary chart
 let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarrchart.getChartData();
 let primarytarget: QCSPCChartTS.SPCControlLimitRecord =

chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_CONTR
OL_TARGET);
      primarytarget.setControlLimitValue ( 30);
      primarytarget.setControlLimitText("XBAR");

      //lower control limit primary chart
```

```
      let primarylowercontrollimit: QCSPCChartTS.SPCControlLimitRecord =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_LOWER
_CONTROL_LIMIT);
      primarylowercontrollimit.setControlLimitValue(24);
      primarylowercontrollimit.setControlLimitText("LCL");

      //upper control limit primary chart
      let primaryuppercontrollimit: QCSPCChartTS.SPCControlLimitRecord =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_UPPER
_CONTROL_LIMIT);
      primaryuppercontrollimit.setControlLimitValue( 36);
      primaryuppercontrollimit.setControlLimitText( "UCL");

      // Set control limits for secondary chart

      //target control limit secondary chart
      let secondarytarget: QCSPCChartTS.SPCControlLimitRecord =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_SECONDARY_CON
TROL_TARGET);
      secondarytarget.setControlLimitValue (10);
      secondarytarget.setControlLimitText("RBAR");

      //lower control limit secondary chart
      let secondarylowercontrollimit: QCSPCChartTS.SPCControlLimitRecord =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_SECONDARY_LOW
ER_CONTROL_LIMIT);
      secondarylowercontrollimit.setControlLimitValue (0);
      secondarylowercontrollimit.setControlLimitText  ("LCL");

      //upper control limit secondary chart
      let secondaryuppercontrollimit: QCSPCChartTS.SPCControlLimitRecord =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_SECONDARY_UPP
ER_CONTROL_LIMIT);
      secondaryuppercontrollimit.setControlLimitValue (22);
      secondaryuppercontrollimit.setControlLimitText ("UCL");
```

We also added a method (add3SigmaControlLimits) which will generate multiple control limits, for +-1, 2, and 3- sigma levels, based on an initial specification of the target value, and the +-3 sigma control limits. This is most useful if you want to generate +-1, 2 and 3-sigma control limits in order to fill in between them with a zone fill color. See the MultiLimitCharts.BuildMultiLimitXBarRChart example. If you call the autoCalculateControlLimits method, the initial +-1,2 and 3-sigma control limit values will be altered to the new, calculated values, but the control limit lines remain, with their new values. Since you do not normally want to be generating alarm messages for excursions into the +-1 and 2-sigma limit areas, the add3SigmaControl limits has the option of disabling alarm notification in the case of +-1 and +-2 alarm conditions.

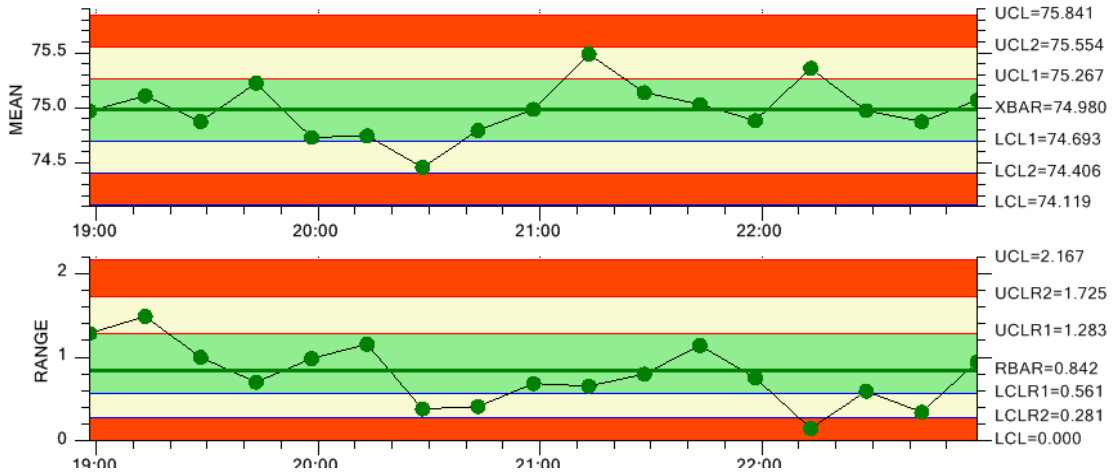
[JavaScript]

```
var target = 75;
var lowlim = 74,
var highlim = 76;
var limitcheck = false;

var primarychart = xbarrchart.getPrimaryChart();
var secondarychart = xbarrchart.getSecondaryChart();

if (primarychart)
{
  primarychart.add3SigmaControlLimits(target, lowlim, highlim, limitcheck);
  primarychart.setControlLimitLineFillMode( true);
}
```

```
if (secondarychart)
{
  target = 1; lowlim = 0; highlim = 2;
  secondarychart.add3SigmaControlLimits(target, lowlim, highlim, limitcheck);
  secondarychart.setControlLimitLineFillMode (true);
}
```

[TypeScript]

```
let target: number = 75;
let lowlim: number = 74,
let highlim: number = 76;
let limitcheck: boolean = false;

let primarychart: QCSPCChartTS.SPCChartObjects | null =
xbarrchart.getPrimaryChart();
let secondarychart: QCSPCChartTS.SPCChartObjects | null =
xbarrchart.getSecondaryChart();

if (primarychart)
{
  primarychart.add3SigmaControlLimits(target, lowlim, highlim, limitcheck);
  primarychart.setControlLimitLineFillMode( true);
}
if (secondarychart)
{
  target = 1; lowlim = 0; highlim = 2;
  secondarychart.add3SigmaControlLimits(target, lowlim, highlim, limitcheck);
  secondarychart.setControlLimitLineFillMode (true);
}
```



Control Limit Fill Option used with +-1, 2 and 3-sigma control limits

The second way to set the control limits is to call the **autoCalculateControlLimits** method. You must have already added a collection of sampled data values to the charts **ChartData** SPC data object before you can call this method, since the method uses the internal **ChartData** object to provide the historical values needed in the calculation.

[JavaScript]

```
// Must have data loaded before any of the Auto.. methods are called
SimulateData(xbarrchart, numssampleintervals, chartmean, chartsigma);
```

```
// Calculate the SPC control limits for both graphs of the current SPC

xbarrchart.autoCalculateControlLimits();
```

[TypeScript]

```
// Must have data loaded before any of the Auto.. methods are called
this.SimulateData(xbarrchart, numssampleintervals, chartmean, chartsigma);

// Calculate the SPC control limits for both graphs of the current SPC

xbarrchart.autoCalculateControlLimits()
```

You can add data to the **ChartData** object, auto-calculate the control limits to establish the SPC control limits, and continue to add new data values. Alternatively, you can set the SPC control limits explicitly, as the result of previous runs, using the previously described **chartdata.setControlLimitValues** method, add new sampled data values to the **ChartData** object, and after a certain number of updates call the **autoCalculateControlLimits** method to establish new control limits.

[JavaScript]

```
updateCount++;
chartdata.addNewSampleRecordDateSamples(timestamp, samples);
if (updateCount > 50) // After 50 sample groups and calculate limits on the fly
{
// Calculate the SPC control limits for the X-Bar part of the current SPC chart
    xbarrchart.autoCalculateControlLimits();
    // Scale the y-axis of the X-Bar chart to display all data and control limits
    xbarrchart.autoScalePrimaryChartYRange();
    // Scale the y-axis of the Range chart to display all data and control limits
    xbarrchart.autoScaleSecondaryChartYRange();
}
```

[TypeScript]

```
updateCount++;
chartdata.addNewSampleRecordDateSamples(timestamp, samples);
if (updateCount > 50) // After 50 sample groups and calculate limits on the fly
{
// Calculate the SPC control limits for the X-Bar part of the current SPC chart
    xbarrchart.autoCalculateControlLimits();
    // Scale the y-axis of the X-Bar chart to display all data and control limits
    xbarrchart.autoScalePrimaryChartYRange();
    // Scale the y-axis of the Range chart to display all data and control limits
    xbarrchart.autoScaleSecondaryChartYRange();
}
```

The **autoCalculateControlLimits** method calculates the control limits for both the primary and secondary charts. If you want to auto-calculate the control limits for just one of the charts, call the **autoCalculatePrimaryControlLimits** or **autoCalculateSecondaryControlLimits** method.

Need to exclude records from the control limit calculation? Call the **chartdata.excludeRecordFromControlLimitCalculations** method, passing in true to exclude the record.

[JavaScript]

```
var i = 0;
for ( i=0; i < 10; i++)
      chartdata.excludeRecordFromControlLimitCalculations(i,true);
```

[TypeScript]

```
let i: number = 0;
for ( i=0; i < 10; i++)
      chartdata.excludeRecordFromControlLimitCalculations(i,true);
```

Formulas Used in Calculating Control Limits for Variable Control Charts

The SPC control limit formulas used by **autoCalculateControlLimits** in the software derive from the following sources:

X-Bar R, X-Bar Sigma, EWMA, MA and CuSum - "Introduction to Statistical Quality Control" by Douglas C. Montgomery, John Wiley and Sons, Inc. 2001.

Median-Range, Individual-Range - "SPC Simplified – Practical Steps to Quality" by Robert T. Amsden, Productivity Inc., 1998.

SPC Control Chart Nomenclature

UCL = Upper Control Limit

LCL = Lower Control Limit

Center line = The target value for the process

$\bar{X}$ = Xbar - Mean of a sample subgroup of size M

$$\bar{X} = \left(\frac{1}{M}\right)\sum_{j=1}^{M} \bar{X}_j = \frac{\left(X_1 + X_2 + \cdots + X_M\right)}{M}$$

$\bar{\bar{X}}$ = X double-bar - Mean of sample subgroup means (also called the grand average), where N is the number of sample subgroups

$$\bar{\bar{X}}_i = \left(\frac{1}{N}\right)\sum_{i=1}^{N} \bar{X}_i = \frac{\left(\bar{X}_1 + \bar{X}_2 + \cdots + \bar{X}_N\right)}{N}$$

$\bar{R}$ = Rbar – Mean of sample subgroup ranges, where N is the number of sample subgroups

$$\bar{R} = \left(\frac{1}{N}\right)\sum_{i=1}^{N} R_i = \frac{\left(R_1 + R_2 + \cdots + R_N\right)}{N}$$

$\tilde{\bar{R}}$ = R-Median – Median of sample subgroup ranges

S = Sigma – sample standard deviation for subgroup size M

$$S = \sqrt{\frac{1}{(M-1)}\sum_{i=1}^{M}\left(X_i - \bar{X}\right)^2}$$

$\bar{S}$ = Sigma-bar – Average of sample subgroup sigma's, for N subgroups

$$\bar{S} = \left(\frac{1}{N}\right)\sum_{i=1}^{N} S_i = \frac{\left(S_1 + S_2 + \cdots + S_N\right)}{N}$$

$\tilde{\bar{M}}$ = Median of sample subgroup medians

X-Bar R Chart – Also known as the Mean (or Average) and Range Chart

Control Limits for the X-Bar Chart

$$UCL = \bar{\bar{X}} + A_2 * \bar{R}$$

$$Center\ line = \bar{\bar{X}}$$

$$LCL = \bar{\bar{X}} - A_2 * \bar{R}$$

Control Limits for the R-Chart

$$UCL = \bar{R} + D_4 * \bar{R}$$

$$Center\ line = \bar{R}$$

$$LCL = \bar{R} - D_3 * \bar{R}$$

Where the constants A_2, D_3 and D_4 are tabulated in every SPC textbook for various sample sizes.

Table of XBar-R Chart Factors

Subgroup Size	A2	D3	D4
2	1.88	0	3.267
3	1.023	0	2.574
4	0.729	0	2.282
5	0.577	0	2.114
6	0.483	0	2.004
7	0.419	0.076	1.924
8	0.373	0.136	1.864
9	0.337	0.184	1.816
15	0.223	0.348	1.652
20	0.18	0.414	1.586
25	0.153	0.459	1.541

X-Bar Sigma – Also known as the X-Bar S Chart

Control Limits for the X-Bar Chart

$$UCL = \bar{\bar{X}} + A_3 * \bar{S}$$

$$Center\ line = \bar{\bar{X}}$$

$$LCL = \bar{\bar{X}} - A_3 * \bar{S}$$

Control Limits for the Sigma-Chart

$$UCL = \bar{B}_4 * \bar{S}$$

$$Center\ line = \bar{S}$$

$$LCL = \bar{B}_3 * \bar{S}$$

Where the constants A_3, B_3 and B_4 are tabulated in every SPC textbook for various sample sizes.

Median Range – Also known as the Median and Range Chart

Control Limits for the Median Chart

$$UCL = \tilde{M} + \tilde{A}_2 * \tilde{R}$$

$$Center\ line = \tilde{M}$$

$$LCL = \tilde{M} - \tilde{A}_2 * \tilde{R}$$

Control Limits for the R-Chart

$$UCL = \tilde{R} + \tilde{D}_4 * \tilde{R}$$

$$Center\ line = \tilde{R}$$

$$LCL = \tilde{R} - \tilde{D}_3 * \tilde{R}$$

The constants A_2, D_3 and D_4 for median-range charts are different than those for mean-range charts. A brief tabulation of the median-range chart specific values appears below

Subgroup Size	**A2**	**D3**	**D4**
2	2.22	0	3.87
3	1.26	0	2.75
4	0.83	0	2.38
5	0.71	0	2.18

Individual Range Chart – Also known as the X-R Chart

Since the subgroup size for Individual Range charts is one, the $\overline{X}$ value is the mean of all of the individual sample values.

The R-value (Range value) for a given sample interval (i) is calculated as:

$$R_i = \text{Abs}(X_i - X_{i-1})$$

starting with the second sample interval. The first sample interval R-value is undefined because there is no X-1 value for the first sample interval. Let (R_2, R_3, ... R_N) be the ranges of the (N-1) sample intervals, calculated this way. The average range $\overline{R}$ (or Rbar) is just the mean of these R-Values.

$$\bar{R} = \left(\frac{1}{N-1}\right) * \sum_{i=2}^{N} R_i = \frac{(R_2 + R_3 + \ldots + R_N)}{(N-1)}$$

Control Limits for the Individuals-Chart

$$UCL = \bar{X} + \frac{3 * \bar{R}}{d_2} = \bar{X} + 2.66 * \bar{R}$$

$$Center\ line = \bar{X}$$

$$LCL = \bar{X} - \frac{3 * \bar{R}}{d_2} = \bar{X} - 2.66 * \bar{R}$$

where the constant d2 is found in the Table of XBar-R Chart Factors in the XBar-R chart page. Since the range calculation for all I-R charts uses two values ($X_i - X_{i-1}$), a sample subgroups size of 2 is used in the d2 lookup, resulting in a value of 1.128. So the constant 1.128 can be substituted in all cases for the constant d2 in the formula above. Since (3/1.128)=2.66, that is where the 2.66 comes from in final reduction of the formulas above.

Control Limits for the R-Chart

$$UCL = D_4 * \bar{R} = 3.267 * \bar{R}$$

$$Center\ line = \bar{R}$$

$$LCL = 0$$

$\overline{R}$ in this case is the average of the moving ranges.

$\overline{X}$ in this case is the mean of the samples

The constant d4 is found in the Table of XBar-R Chart Factors in the XBar-R chart page. Since the range calculation for all I-R charts uses two values ($X_i - X_{i-1}$), a sample subgroups size of 2 is used in the d4 lookup, resulting in a value of 3.267. That is where the 3.267 comes from in final reduction of the UCL formula above. The LCL value for Individual-Range charts is always 0.0.

Levey-Jennings Chart

Control Limits

+Sigma 1	=	Mean + SD
+Sigma 2	=	Mean + 2 * SD
+Sigma 3	=	Mean + 3 *SD
Center line	=	Mean
-Sigma 1	=	Mean - SD
-Sigma 2	=	Mean - 2 * SD
-Sigma 3	=	Mean - 3 *SD

SD = the standard deviation of the sample population

Mean = the mean of the sample population

EWMA Chart – Exponentially Weighted Moving Average

The current value (z) for an EWMA chart is calculated as an exponentially weighted moving average of all previous samples.

$$z_i = \lambda * x_i + (1-\lambda) * z_{i-1}$$

where x_i is the sample value for time interval i, the smoothing value λ has the permissible range of $0 < \lambda <= 1$ and the starting value (required with the first sample at i = 0) is the process target value, μ_0 .

Control Limits for the EWMA Chart

$$UCL = \mu_0 + L * \sigma * \sqrt{\frac{\lambda}{(2-\lambda)} * (1-(1-\lambda)^{2i})}$$

$$Center\ line = \mu_0$$

$$LCL = \mu_0 + L * \sigma * \sqrt{\frac{\lambda}{(2-\lambda)} * \left(1-(1-\lambda)^{2i}\right)}$$

μ_0 is the process mean

σ is the process standard deviation, also known as sigma

λ is the user specified smoothing value. A typical value for λ is 0.05, 0.1 or 0.2

L is the width of the control limits. The typical value for L is in the range of 2.7 to 3.0 (corresponding to the usual three-sigma control limits).

The software does not calculate optimal λ and L values; that is up to you, the programmer to supply, based on your experience with EWMA charts.

Note that the term $(1- (1- \lambda)^{2i})$ approaches unity as i increases. The implies that the control limits of an EWMA chart will reach approximate steady state values defined by:

$$UCL = \mu_0 + L * \sigma * \sqrt{\frac{\lambda}{(2-\lambda)}}$$

$$LCL = \mu_0 - L * \sigma * \sqrt{\frac{\lambda}{(2-\lambda)}}$$

It is best if you use the exact equations that take into account the sample period, so that an out of control process can be detected using the tighter control limits that are calculated for small i.

If the EWMA chart is used with subgroup sample sizes greater than 1, the value of x_i is replace by the mean of the corresponding sample subgroup, and the value of σ is replaced by the value σ/sqrt(n), where in is the sample subgroup size.

You specify λ and L immediately after the initialization call **newSPCBatchVariableControlChart...** . See the example VariableControlCharts.BuildEWMAChart. Specify λ using the **EWMA_Lambda** property. You can optionally set the EWMA starting value (**EWMA_StartingValue**), normally set to the process mean value, and whether or not to use the steady-state EWMA control limits (**UseSSLimits**).

Extracted from the VariableControlCharts.BuildEWMAChart example.

[JavaScript]

```
   var ewmachart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
.
.
.
    var chartdata = ewmachart.getChartData();
    if (chartdata) {

        chartdata.setEWMA_Lambda(0.2);
        chartdata.setEWMA_StartingValue(30);
        chartdata.setTitle (charttitle);
        chartdata.setChartDescriptor("EWMA");
    }
```

[TypeScript]

```
   let ewmachart: QCSPCChartTS.SPCBatchVariableControlChart=
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
.
.
.
    let chartdata: QCSPCChartTS.SPCControlChartData | null =
ewmachart.getChartData();
    if (chartdata) {
        chartdata.setEWMA_Lambda(0.2);
        chartdata.setEWMA_StartingValue(30);
        chartdata.setTitle (charttitle);
        chartdata.setChartDescriptor("EWMA");
    }
```

MA Chart – Moving Average

The current value (z) for a MA chart is calculated as a weighted moving average of the N most recent samples.

$$z_i = (x_i + x_{i-1} + x_{i-2} + \cdots x_{i-N+1})/N$$

where x_i is the sample value for time interval i, and N is the length of the moving average.

Calculate the μ_0 value as the mean of the actual data values. Or you can use a μ_0 value that you know from previous runs.

$$\mu_0 = \bar{X} = \frac{1}{N}\sum_{i=1}^{N} \bar{X}_i = \frac{(X_1 + X_2 + \cdots + X_N)}{N}$$

The standard deviation value (σ for the process is calculated using the sample standard deviation formula, seen below. Or you can use a σ value that you know from previous runs.

$$\sigma = \sqrt{\frac{1}{(N-1)} \sum_{i=1}^{N} (X_i - \bar{X})^2}$$

Control Limits for the MA Chart

$$UCL = \mu_0 + 3 * \frac{\sigma}{\sqrt{(M)}}$$

$$Center\ line = \mu_0$$

$$LCL = u_0 - 3 * \frac{\sigma}{\sqrt{M}}$$

μ_0 is the process mean

σ is the process standard deviation, also known as sigma

M is the length of the moving average used to calculate the current chart value

Control Limits for the MR part of the MAMR (Moving Average/Moving Range Chart

Control Limits for the R-Chart

$$UCL = \bar{R} + D_4 * \bar{R}$$

$$Center\ line = \bar{R}$$

$$LCL = 0$$

$\bar{\bar{R}}$ in this case is the average of the moving ranges.

Where the constant D_4 is tabulated in every SPC textbook for various sample sizes.

Control Limits for the MS part of the MAMS
(Moving Average/Moving Sigma Chart)

$$UCL = \bar{B}_4 * \bar{S}$$

$$Center\ line = \bar{S}$$

$$LCL = \bar{B}_3 * \bar{S}$$

$\bar{S}$ in this case is the average of the moving sigmas.

Where the constant B_4 is tabulated in every SPC textbook for various sample sizes.
The software does not calculate an optimal N value; that is up to you, the programmer to supply, based on your past experience with MA charts.

For the values of z_i where i < N-1, the weighted average and control limits are calculated using the actual number of samples used in the average, rather than N. This results in expanded values for the control limits for small i < N-1.

You specify N, the length of the moving average, immediately after the initialization call **newSPCBatchVariableControl...**. Set the process mean and process sigma used in the control limit calculations using the **ProcessMean** and **ProcessSigma** properties.
See the examples VariableControlCharts.BuildMAChart, and VariableControlCharts.BuildMAChart. Specify N using the **MA_w** property.

Extracted from the VariableControlCharts.MAChart example.

[JavaScript]

```
var htmlcanvas = document.getElementById(canvasid);
var spccharttype = QCSPCChartTS.SPCControlChartData.MA_CHART;
var subgroupsize = 1;
var numberpointsinview = 12;
var charttitle = "MAMR Example";

var machart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);

.
.
.
    var chartdata = machart.getChartData();
    if (chartdata) {
```

```
      chartdata.setMA_w(5);
      chartdata.setTitle (charttitle);
      chartdata.setChartDescriptor("MA");
    };
```

[TypeScript]

```
let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
let spccharttype: number = QCSPCChartTS.SPCControlChartData.MA_CHART;
let subgroupsize: number = 1;
let numberpointsinview: number = 12;
let charttitle: string = "MA Example";

let machart: QCSPCChartTS.SPCBatchVariableControlChart=
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
.
.
.

    let chartdata: QCSPCChartTS.SPCControlChartData | null =
machart.getChartData();
    if (chartdata) {
      chartdata.setMA_w(5);
      chartdata.setTitle (charttitle);
      chartdata.setChartDescriptor("MA");
    };
```

CuSum Chart – Tabular, one-sided, upper and lower cumulative sum

The tabular cusum works by accumulating deviations from the process mean, μ_0. Positive deviations are accumulated in the one sided upper cusum statistic, C^+, and negative deviations are accumulated in the one sided lower cusum statistic, C^-. The statistics are calculated using the following equations:

$$C_i^+ = max\left(0, x_i - (\mu_0 + K) + C_{i-1}^+\right)$$

$$C_i^- = max\left(0, (\mu_0 - K) - x_i + C_{i-1}^-\right)$$

where the starting values are $C^+_0 = C^-_0 = 0$

μ_0 is the process mean

$$\mu_0 = \bar{\bar{X}} = \frac{1}{N}\sum_{i=1}^{N} \bar{X}_i = \frac{(X_1 + X_2 + \cdots + X_N)}{N}$$

K is the reference (or slack value) that is usually selected to be one-half the magnitude of the difference between the target value, μ_0 , and the out of control process mean value, μ_1, that you are trying to detect.

$$K = \frac{|(\mu_1 - \mu_0)|}{2}$$

The control limits used by the chart are +-H. If the value of either C^+ or C^- exceed +- H, the process is considered out of control.

The software does not calculate an optimal H or K value; that is up to you, the programmer to supply, based on your past experience with CuSum charts. Typically H is set equal to 5 times the process standard deviation, σ. Typically K is selected to be one-half the magnitude of the difference between the target value, μ_0 , and the out of control process mean value, μ_1, that you are trying to detect. You specify H and K in the initialization call **newSPCBatchCusumControlChart, or initSPCEventCusumControlChar**. See the examples VariableControlCharts.BuildCusumChart.

[JavaScript]

```
  var htmlcanvas = document.getElementById(canvasid);
    var spccharttype  = QCSPCChartTS.SPCControlChartData.TABCUSUM_CHART;
    var subgroupsize = 1;
    var numberpointsinview  = 12;
    var charttitle = "Cusum Example";
    var processMean  = 10;
    var kValue  = 0.5;
    var hValue  = 5;

    var cusumchart =

QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchCusumControlChart(htmlcanvas,
spccharttype, subgroupsize, numberpointsinview, processMean, kValue, hValue);
```

[TypeScript]

```
    let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
    let spccharttype: number  = QCSPCChartTS.SPCControlChartData.TABCUSUM_CHART;
    let subgroupsize: number = 1;
    let numberpointsinview: number  = 12;
    let charttitle: string = "Cusum Example";
    let processMean: number  = 10;
    let kValue: number  = 0.5;
    let hValue: number  = 5;

    let cusumchart: QCSPCChartTS.SPCBatchVariableControlChart=

QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchCusumControlChart(htmlcanvas,
spccharttype, subgroupsize, numberpointsinview, processMean, kValue, hValue);
```

Variable SPC Control Limits

SPC control limits can be either fixed or variable. In actuality, the limits are always variable, but if you assign a set of limits and don't modify them, the limits will look fixed.

There are at least three ways to enter new SPC limit values. See the example program VariableControlLimits.BuildVariableLimitsXBarRChart
for an example of all three methods. First, you can use the method **chartdata.setControlLimitValues** method.

Extracted from the VariableControlLimits.BuildVariableLimitsXBarRChart example

[JavaScript]

```
// Explicitly set control limits, before any update calls.
var initialControlLimits = [30, 25, 35, 10, 0, 20];
if (chartdata)
   chartdata.setControlLimitValues(initialControlLimits);
.
.
.
var  changeControlLimits = [28, 23, 33, 9, 0, 18];
// Change limits at sample subgroup 10
if (i== 10)
{
    chartdata.setControlLimitValues(changeControlLimits);
}
chartdata.addNewSampleRecordBatchNumberDateSamples (batchnum, timestamp, samples,
notesstring);
```

[TypeScript]

```
//Explicitly set control limits, before any update calls.
  let initialControlLimits: number[] = [30, 25, 35, 10, 0, 20];
  if (chartdata)
    chartdata.setControlLimitValues(initialControlLimits);

.
.
.
.


  let changeControlLimits: number[] = [28, 23, 33, 9, 0, 18];


        if (currentcount > 0) // start date at the previous ending date plut time
increment
        {
        let ts : Date = chartdata.getTimeValue(currentcount-1);
        timestamp = ts;
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }

for (i = 0; i < count; i++) {

         // Simulate a sample subgroup record
```

```
        let samples: QCSPCChartTS.DoubleArray =
chartdata.simulateMeasurementRecordMeanRange(mean, sigma);


        batchCounter = currentcount + i;
        if  (batchCounter == 10)
          chartdata.setControlLimitValues(changeControlLimits);

        // Add a new sample record to the chart data
         chartdata.addNewSampleRecordBatchNumberDateSamples (batchCounter,
timestamp, samples);
        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }
```

Second, you can use the **autoCalculateControlLimits** method. You must have already added a collection of sampled data values to the charts **ChartData** SPC data object before you can call this method, since the method uses the internal **ChartData** object to provide the historical values needed in the calculation.

Extracted from the VariableControlLimits.BuildVariableLimitsIRChart example

[JavaScript]

```
.
.
.

function SimulateIRChartData(spcchart, count, mean, sigma)  {
  // batch number for a given sample subgroup
  var batchCounter = 0;
      var i = 0;
      var timestamp = new Date();
      var chartdata = spcchart.getChartData();
      if (!chartdata) return;
      var currentcount = chartdata.getCurrentNumberRecords();

      if (currentcount > 0) // start date at the previous ending date plut time
increment
      {
          var ts  = chartdata.getTimeValue(currentcount-1);
          timestamp = ts;
          QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
      }
  for (i = 0; i < count; i++) {
      // Simulate a sample subgroup record
      var samples = chartdata.simulateMeasurementRecordMeanRange(mean, sigma);
      // Update chart data using i as the batch number
      batchCounter = currentcount + i;
      if (batchCounter > 10)
      {
        spcchart.autoCalculateControlLimits();
      }
      chartdata.addNewSampleRecordBatchNumberDateSamples (i, timestamp, samples);
      // Simulate passage of timeincrementminutes minutes
      QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
  }

}
```

[TypeScript]

```
.
.
.
SimulateIRChartData(spcchart: QCSPCChartTS.SPCChartBase, count: number, mean:
number, sigma: number)  {
  // batch number for a given sample subgroup
  let batchCounter: number = 0;
      let i: number = 0;
      let timestamp: Date = new Date();
      let chartdata: QCSPCChartTS.SPCControlChartData | null =
spcchart.getChartData();
      if (!chartdata) return;
      let currentcount = chartdata.getCurrentNumberRecords();

      if (currentcount > 0) // start date at the previous ending date plut time
increment
      {
          let ts : Date = chartdata.getTimeValue(currentcount-1);
          timestamp = ts;
          QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
      }
  for (i = 0; i < count; i++) {
      // Simulate a sample subgroup record
      let samples: QCSPCChartTS.DoubleArray =
chartdata.simulateMeasurementRecordMeanRange(mean, sigma);
      // Update chart data using i as the batch number
      batchCounter = currentcount + i;
      if (batchCounter > 10)
      {
        spcchart.autoCalculateControlLimits();
      }
      chartdata.addNewSampleRecordBatchNumberDateSamples (i, timestamp, samples);
      // Simulate passage of timeincrementminutes minutes
      QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
  }

}
```

Third, you can enter the SPC control limits with every new sample subgroup record, using one of the methods that include a control limits array parameter.

Extracted from the VariableControlLimits.BuildVariableLimitsXBarSigmaChart example

[JavaScript]

```
function SimulateXBarSigmaData(spcchart, count, mean,  sigma) {
  // batch number for a given sample subgroup
  var batchCounter = 0;
  var i = 0;
  var timestamp = new Date();
  if (!spcchart) return;
  var chartdata = spcchart.getChartData();
  if (!chartdata) return;
  var currentcount = chartdata.getCurrentNumberRecords();
  var charttype = spcchart.getSPCChartType();
  var changeControlLimits = [28, 23, 33, 9, 0, 18];

  var variableControlLimits  = chartdata.getControlLimitValues();
  if (currentcount > 0) // start date at the previous ending date plut time
increment
  {
      var ts  = chartdata.getTimeValue(currentcount-1);
      timestamp = ts;
      QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
```

```
  }
  for (i = 0; i < count; i++) {

      var samples = chartdata.simulateMeasurementRecordMeanSigma(mean,sigma);
      batchCounter = i + currentcount;

      if (batchCounter== 10)
            variableControlLimits =
QCSPCChartTS.DoubleArray.newDoubleArrayArray(changeControlLimits);

      // Add a new sample record
      if (variableControlLimits)
         chartdata.addNewSampleRecordBatchNumberDateSamplesControlLimits
(batchCounter, timestamp, samples,variableControlLimits);

      // Simulate passage of timeincrementminutes minutes
      QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
  }

}
```

[TypeScript]

```
SimulateXBarSigmaData(spcchart: QCSPCChartTS.SPCChartBase, count: number, mean:
number,  sigma: number) {
  // batch number for a given sample subgroup
  let batchCounter: number = 0;
  let i: number = 0;
  let timestamp: Date = new Date();
  if (!spcchart) return;
  let chartdata: QCSPCChartTS.SPCControlChartData | null =
spcchart.getChartData();
  if (!chartdata) return;
  let currentcount: number = chartdata.getCurrentNumberRecords();
  let charttype: number = spcchart.getSPCChartType();
  let changeControlLimits: number[] = [28, 23, 33, 9, 0, 18];

  let variableControlLimits:  QCSPCChartTS.DoubleArray  =
chartdata.getControlLimitValues();
  if (currentcount > 0) // start date at the previous ending date plut time
increment
  {
      let ts : Date = chartdata.getTimeValue(currentcount-1);
      timestamp = ts;
      QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
  }
  for (i = 0; i < count; i++) {

      let samples: QCSPCChartTS.DoubleArray =
chartdata.simulateMeasurementRecordMeanSigma(mean,sigma);
      batchCounter = i + currentcount;

      if (batchCounter== 10)
            variableControlLimits =
QCSPCChartTS.DoubleArray.newDoubleArrayArray(changeControlLimits);

      // Add a new sample record
      if (variableControlLimits)
         chartdata.addNewSampleRecordBatchNumberDateSamplesControlLimits
(batchCounter, timestamp, samples,variableControlLimits);

      // Simulate passage of timeincrementminutes minutes
      QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
  }
```

```
}
```

Fourth, if you are dealing with multiple limits, such as in the display of the +-1, +-2 and +-3-sigma lines, rather than set each line to an explicit value, it is easier if you use the method. **updateControlLimitsUsingMeanAndSigma**. That way you can specify just the new mean and sigma value, and all of the control limit lines in the chart will update accordingly. If you are using one of the charts with both Primary and Secondary charts (Xbar-R, Xbar-Sigma, I-R, Median-Range, etc.), you will need to call the updateControlLimitsUsingMeanAndSigma for both the Primary and Secondary charts.

Extracted from the VariableControlLimits.BuildVariableLimitsLeveyJenningsChart example

[JavaScript]

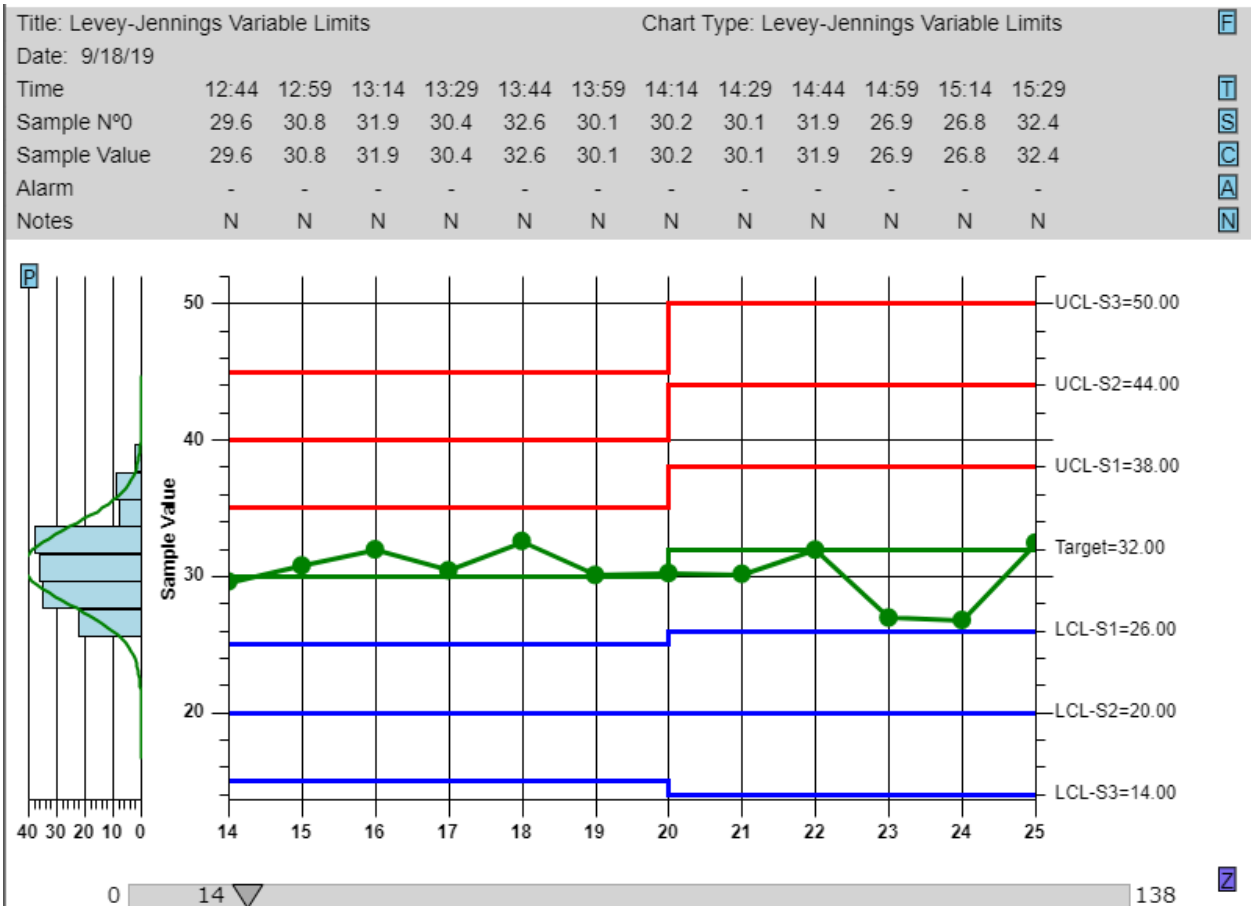
```
function SimulateLeveyJenningsData(spcchart, count, mean, sigma)  {
    // batch number for a given sample subgroup
    var batchCounter = 0;
        var i = 0;
        var timestamp = new Date();
        var chartdata = spcchart.getChartData();
        if (!chartdata) return;
        var currentcount = chartdata.getCurrentNumberRecords();


        if (currentcount > 0) // start date at the previous ending date plut time
increment
        {
            var ts  = chartdata.getTimeValue(currentcount-1);
            timestamp = ts;
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }

    for (i = 0; i < count; i++) {

             // Simulate a sample subgroup record
        var samples = chartdata.simulateMeasurementRecordMeanRange(mean, sigma);

        batchCounter = currentcount + i;
        if  (batchCounter == 20)
        {
            var chartmean = 32;
            var chartsigma = 6;
            if (chartdata)
            {

chartdata.updateControlLimitsUsingMeanAndSigma(QCSPCChartTS.SPCChartObjects.PRIMAR
Y_CHART,chartmean, chartsigma);
            // if you have both primary and secondary charts, you will also need
to call updateControlLimitsUsingMeanAndSigma for the secondary chart
            // The Levey-Jennings chart does NOT have a Secondary chart.
          //  chartmean = 6;
          //  chartsigma = 2;
          //
chartdata.updateControlLimitsUsingMeanAndSigma(QCSPCChartTS.SPCChartObjects.SECOND
ARY_CHART,chartmean, chartsigma);
        }

        }
        // Add a new sample record to the chart data
```

```
         chartdata.addNewSampleRecordBatchNumberDateSamples (batchCounter,
timestamp, samples);

        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }

}
```

[TypeScript]

```
SimulateLeveyJenningsData(spcchart: QCSPCChartTS.SPCChartBase, count: number,
mean: number, sigma: number)  {
    // batch number for a given sample subgroup
    let batchCounter: number = 0;
        let i: number = 0;
        let timestamp: Date = new Date();
        let chartdata: QCSPCChartTS.SPCControlChartData | null =
spcchart.getChartData();
        if (!chartdata) return;
        let currentcount = chartdata.getCurrentNumberRecords();


        if (currentcount > 0) // start date at the previous ending date plut time
increment
        {
            let ts : Date = chartdata.getTimeValue(currentcount-1);
            timestamp = ts;
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }

    for (i = 0; i < count; i++) {

             // Simulate a sample subgroup record
        let samples: QCSPCChartTS.DoubleArray =
chartdata.simulateMeasurementRecordMeanRange(mean, sigma);

        batchCounter = currentcount + i;
        if  (batchCounter == 20)
        {
            let chartmean: number = 32;
            let chartsigma: number = 6;
            if (chartdata)
            {

chartdata.updateControlLimitsUsingMeanAndSigma(QCSPCChartTS.SPCChartObjects.PRIMAR
Y_CHART,chartmean, chartsigma);
            // if you have both primary and secondary charts, you will also need
to call updateControlLimitsUsingMeanAndSigma for the secondary chart
            // The Levey-Jennings chart does NOT have a Secondary chart.
          //  chartmean = 6;
          //  chartsigma = 2;
          //
chartdata.updateControlLimitsUsingMeanAndSigma(QCSPCChartTS.SPCChartObjects.SECOND
ARY_CHART,chartmean, chartsigma);
        }

        }
        // Add a new sample record to the chart data
         chartdata.addNewSampleRecordBatchNumberDateSamples (batchCounter,
timestamp, samples);

        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }
```

}



In this example, the control limits are changed in mid course.

For a complete example, see the example VariableControlLimits.BuildVariableLimitsLeveyJenningsChart
.

Multiple SPC Control Limits

The normal SPC control limit displays at the 3-sigma level, both high and low. A common standard is that if the process variable under observation falls outside of the +-3-sigma limits the process is out of control. The default setup of our variable control charts have a high limit at the +3-sigma level, a low limit at the -3-sigma level, and a target value. There are situations where the quality engineer also wants to display control limits

at the 1-sigma and 2-sigma level. The operator might receive some sort of preliminary warning if the process variable exceeds a 2-sigma limit.

You are able to add additional control limit lines to a variable control chart, as in the example program MultiLimitCharts.BuildMultiLimitXBarRChart.

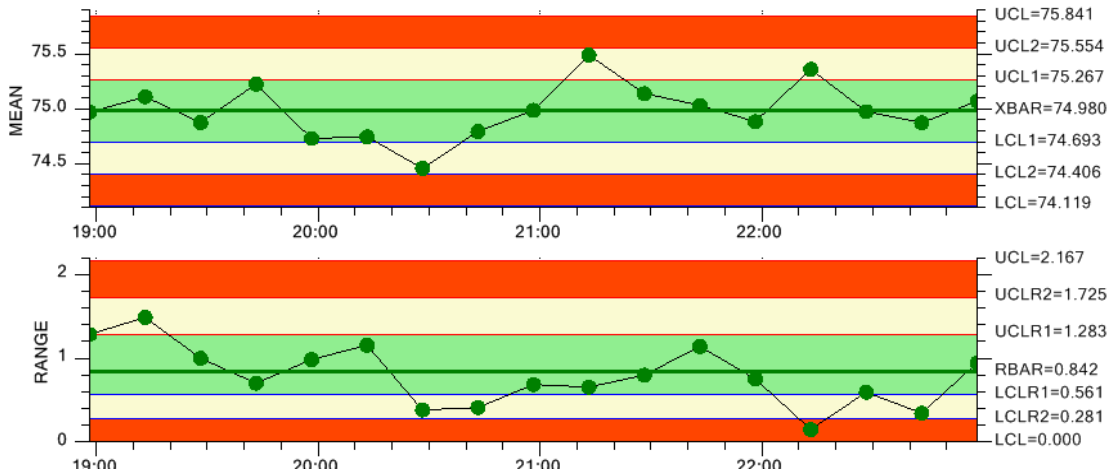
We also added a method (add3SigmaControlLimits) which will generate multiple control limits, for +-1, 2, and 3- sigma levels, based on an initial specification of the target value, and the +-3 sigma control limits. This is most useful if you want to generate +-1, 2 and 3-sigma control limits in order to fill in between them with a zone fill color. See the MultiLimitCharts.BuildMultiLimitXBarRChart example. If you call the **autoCalculateControlLimits** method, the initial +-1,2 and 3-sigma control limit values will be altered to the new, calculated values, but the control limit lines remain, with their new values. Since you do not normally want to be generating alarm messages for excursions into the +-1 and 2-sigma limit areas, the add3SigmaControl limits has the option of disabling alarm notification in the case of +-1 and +-2 alarm conditions.

[JavaScript]

```
var target = 75;
var lowlim = 74;
var highlim = 76;
var limitcheck = false;
if (primarychart)
{
  primarychart.add3SigmaControlLimits(target, lowlim, highlim, limitcheck);
  primarychart.setControlLimitLineFillMode (true);
}


if (secondarychart)
{
  target = 1; lowlim = 0; highlim = 2;
  secondarychart.add3SigmaControlLimits(target, lowlim, highlim, limitcheck);
  secondarychart.setControlLimitLineFillMode(true);
}
```

[TypeScript]

```
let target: number = 75;
let lowlim: number = 74;
let highlim: number = 76;
let limitcheck: boolean = false;

if (primarychart)
{
  primarychart.add3SigmaControlLimits(target, lowlim, highlim, limitcheck);
  primarychart.setControlLimitLineFillMode (true);
}

if (secondarychart)
{
  target = 1; lowlim = 0; highlim = 2;
  secondarychart.add3SigmaControlLimits(target, lowlim, highlim, limitcheck);
  secondarychart.setControlLimitLineFillMode(true);
}
```



Control Limit Fill Option used with +-1, 2 and 3-sigma control limits

Special Note - We view the technique described below, for adding additional control limits, much too complicated. We include it here only because it was included in the manuals for the other versions of QCSPCChart.

You can also add additional control limits one at a time. By default you get the +-3-sigma control limits. So additional control limits should be considered +-2-sigma and +-1-sigma control limits. Do not confuse control limits with specification limits, which must be added using the **addSpecLimit** method. There are two steps to adding additional control limits: creating a SPCControlLimitRecord object for the new control limit, and adding the control limit to the chart using the charts addAdditionalControlLimit method. It is critical that you add them in a specific order, that order being:

Primary Chart	SPC_LOWER_CONTROL_LIMIT_2	(2-sigma lower limit)
Primary Chart	SPC_UPPER_CONTROL_LIMIT_2	(2-sigma lower limit)
Primary Chart	SPC_LOWER_CONTROL_LIMIT_1	(1-sigma lower limit)
Primary Chart	SPC_UPPER_CONTROL_LIMIT_1	(1-sigma lower limit)
Secondary Chart	SPC_LOWER_CONTROL_LIMIT_2	(2-sigma lower limit)
Secondary Chart	SPC_UPPER_CONTROL_LIMIT_2	(2-sigma lower limit)
Secondary Chart	SPC_LOWER_CONTROL_LIMIT_1	(1-sigma lower limit)
Secondary Chart	SPC_UPPER_CONTROL_LIMIT_1	(1-sigma lower limit)

[JavaScript]

```
var sigma2 = 2.0;
var sigma1 = 1.0;

var primarychart = xbarrchart.getPrimaryChart();
var secondarychart = xbarrchart.getSecondaryChart();

if (!primarychart) return;
if (!secondarychart) return;
```

```
if (!chartdata) return;

var lcllimit =
primarychart.getControlLimitDataIndex(QCSPCChartTS.SPCChartObjects.SPC_LOWER_CONTR
OL_LIMIT);
if (lcllimit)  lcllimit.setLimitValue(74);

var ucllimit =
primarychart.getControlLimitDataIndex(QCSPCChartTS.SPCChartObjects.SPC_UPPER_CONTR
OL_LIMIT);
if (ucllimit) ucllimit.setLimitValue(76);

var target =
primarychart.getControlLimitDataIndex(QCSPCChartTS.SPCChartObjects.SPC_CONTROL_TAR
GET);
if (target) target.setLimitValue(75);

// Create multiple limits
// For PrimaryChart
var lcl2 =
QCSPCChartTS.SPCControlLimitRecord.newSPCControlLimitRecordProcessVarAlarmTypeAlar
mValueAlarmMessages(chartdata,
QCSPCChartTS.SPCControlLimitRecord.SPC_LOWERTHAN_LIMIT, 74.33,"LCL2", "LCL2");
var ucl2 =
QCSPCChartTS.SPCControlLimitRecord.newSPCControlLimitRecordProcessVarAlarmTypeAlar
mValueAlarmMessages(chartdata,
QCSPCChartTS.SPCControlLimitRecord.SPC_GREATERTHAN_LIMIT, 75.66,"UCL2", "UCL2");

primarychart.addAdditionalControlLimit(lcl2,
QCSPCChartTS.SPCChartObjects.SPC_LOWER_CONTROL_LIMIT_2, sigma2);
primarychart.addAdditionalControlLimit(ucl2,
QCSPCChartTS.SPCChartObjects.SPC_UPPER_CONTROL_LIMIT_2, sigma2);
var lcl3 =
QCSPCChartTS.SPCControlLimitRecord.newSPCControlLimitRecordProcessVarAlarmTypeAlar
mValueAlarmMessages(chartdata,
QCSPCChartTS.SPCControlLimitRecord.SPC_LOWERTHAN_LIMIT, 74.66,"LCL1", "LCL1");
var ucl3 =
QCSPCChartTS.SPCControlLimitRecord.newSPCControlLimitRecordProcessVarAlarmTypeAlar
mValueAlarmMessages(chartdata,
QCSPCChartTS.SPCControlLimitRecord.SPC_GREATERTHAN_LIMIT, 75.33,"UCL1", "UCL1");

primarychart.addAdditionalControlLimit(lcl3,
QCSPCChartTS.SPCChartObjects.SPC_LOWER_CONTROL_LIMIT_1, sigma1);
primarychart.addAdditionalControlLimit(ucl3,
QCSPCChartTS.SPCChartObjects.SPC_UPPER_CONTROL_LIMIT_1, sigma1);


lcllimit =
secondarychart.getControlLimitDataIndex(QCSPCChartTS.SPCChartObjects.SPC_LOWER_CON
TROL_LIMIT);
if (lcllimit)  lcllimit.setLimitValue(2);

ucllimit =
secondarychart.getControlLimitDataIndex(QCSPCChartTS.SPCChartObjects.SPC_UPPER_CON
TROL_LIMIT);
if (ucllimit) ucllimit.setLimitValue(0);

target =
secondarychart.getControlLimitDataIndex(QCSPCChartTS.SPCChartObjects.SPC_CONTROL_T
ARGET);
if (target) target.setLimitValue(1);


var lcl4 =
QCSPCChartTS.SPCControlLimitRecord.newSPCControlLimitRecordProcessVarAlarmTypeAlar
mValueAlarmMessages(chartdata,
QCSPCChartTS.SPCControlLimitRecord.SPC_LOWERTHAN_LIMIT, 0.33,"LCL2", "LCL2");
var ucl4 =
QCSPCChartTS.SPCControlLimitRecord.newSPCControlLimitRecordProcessVarAlarmTypeAlar
mValueAlarmMessages(chartdata,
QCSPCChartTS.SPCControlLimitRecord.SPC_GREATERTHAN_LIMIT, 1.66,"UCL2", "UCL2");
```

```
secondarychart.addAdditionalControlLimit(lcl4,
QCSPCChartTS.SPCChartObjects.SPC_LOWER_CONTROL_LIMIT_2, sigma2);
secondarychart.addAdditionalControlLimit(ucl4,
QCSPCChartTS.SPCChartObjects.SPC_UPPER_CONTROL_LIMIT_2, sigma2);

var lcl5 =
QCSPCChartTS.SPCControlLimitRecord.newSPCControlLimitRecordProcessVarAlarmTypeAlar
mValueAlarmMessages(chartdata,
QCSPCChartTS.SPCControlLimitRecord.SPC_LOWERTHAN_LIMIT, 0.66,"LCL1", "LCL1");
var ucl5 =
QCSPCChartTS.SPCControlLimitRecord.newSPCControlLimitRecordProcessVarAlarmTypeAlar
mValueAlarmMessages(chartdata,
QCSPCChartTS.SPCControlLimitRecord.SPC_GREATERTHAN_LIMIT, 1.33,"UCL1", "UCL1");

secondarychart.addAdditionalControlLimit(lcl5,
QCSPCChartTS.SPCChartObjects.SPC_LOWER_CONTROL_LIMIT_1, sigma1);
secondarychart.addAdditionalControlLimit(ucl5,
QCSPCChartTS.SPCChartObjects.SPC_UPPER_CONTROL_LIMIT_1, sigma1);

primarychart.setControlLimitLineFillMode( true);

secondarychart.setControlLimitLineFillMode( true);
```

[TypeScript]

```
let sigma2: number = 2.0;
let sigma1: number = 1.0;

let primarychart: QCSPCChartTS.SPCChartObjects | null =
xbarrchart.getPrimaryChart();
let secondarychart: QCSPCChartTS.SPCChartObjects | null =
xbarrchart.getSecondaryChart();

if (!primarychart) return;
if (!secondarychart) return;
if (!chartdata) return;

let lcllimit: QCSPCChartTS.SPCControlPlotObjectData | null =
primarychart.getControlLimitDataIndex(QCSPCChartTS.SPCChartObjects.SPC_LOWER_CONTR
OL_LIMIT);
if (lcllimit)  lcllimit.setLimitValue(74);

let ucllimit: QCSPCChartTS.SPCControlPlotObjectData | null =
primarychart.getControlLimitDataIndex(QCSPCChartTS.SPCChartObjects.SPC_UPPER_CONTR
OL_LIMIT);
if (ucllimit) ucllimit.setLimitValue(76);

let target: QCSPCChartTS.SPCControlPlotObjectData | null =
primarychart.getControlLimitDataIndex(QCSPCChartTS.SPCChartObjects.SPC_CONTROL_TAR
GET);
if (target) target.setLimitValue(75);

// Create multiple limits
// For PrimaryChart
let lcl2: QCSPCChartTS.SPCControlLimitRecord =
QCSPCChartTS.SPCControlLimitRecord.newSPCControlLimitRecordProcessVarAlarmTypeAlar
mValueAlarmMessages(chartdata,
QCSPCChartTS.SPCControlLimitRecord.SPC_LOWERTHAN_LIMIT, 74.33,"LCL2", "LCL2");
let ucl2: QCSPCChartTS.SPCControlLimitRecord =
QCSPCChartTS.SPCControlLimitRecord.newSPCControlLimitRecordProcessVarAlarmTypeAlar
mValueAlarmMessages(chartdata,
QCSPCChartTS.SPCControlLimitRecord.SPC_GREATERTHAN_LIMIT, 75.66,"UCL2", "UCL2");

primarychart.addAdditionalControlLimit(lcl2,
QCSPCChartTS.SPCChartObjects.SPC_LOWER_CONTROL_LIMIT_2, sigma2);
primarychart.addAdditionalControlLimit(ucl2,
QCSPCChartTS.SPCChartObjects.SPC_UPPER_CONTROL_LIMIT_2, sigma2);
let lcl3: QCSPCChartTS.SPCControlLimitRecord =
QCSPCChartTS.SPCControlLimitRecord.newSPCControlLimitRecordProcessVarAlarmTypeAlar
```

```
mValueAlarmMessages(chartdata,
QCSPCChartTS.SPCControlLimitRecord.SPC_LOWERTHAN_LIMIT, 74.66,"LCL1", "LCL1");
let ucl3: QCSPCChartTS.SPCControlLimitRecord =
QCSPCChartTS.SPCControlLimitRecord.newSPCControlLimitRecordProcessVarAlarmTypeAlar
mValueAlarmMessages(chartdata,
QCSPCChartTS.SPCControlLimitRecord.SPC_GREATERTHAN_LIMIT, 75.33,"UCL1", "UCL1");

primarychart.addAdditionalControlLimit(lcl3,
QCSPCChartTS.SPCChartObjects.SPC_LOWER_CONTROL_LIMIT_1, sigma1);
primarychart.addAdditionalControlLimit(ucl3,
QCSPCChartTS.SPCChartObjects.SPC_UPPER_CONTROL_LIMIT_1, sigma1);


lcllimit =
secondarychart.getControlLimitDataIndex(QCSPCChartTS.SPCChartObjects.SPC_LOWER_CON
TROL_LIMIT);
if (lcllimit)  lcllimit.setLimitValue(2);

lcllimit =
secondarychart.getControlLimitDataIndex(QCSPCChartTS.SPCChartObjects.SPC_UPPER_CON
TROL_LIMIT);
if (ucllimit) ucllimit.setLimitValue(0);

target =
secondarychart.getControlLimitDataIndex(QCSPCChartTS.SPCChartObjects.SPC_CONTROL_T
ARGET);
if (target) target.setLimitValue(1);


let lcl4: QCSPCChartTS.SPCControlLimitRecord =
QCSPCChartTS.SPCControlLimitRecord.newSPCControlLimitRecordProcessVarAlarmTypeAlar
mValueAlarmMessages(chartdata,
QCSPCChartTS.SPCControlLimitRecord.SPC_LOWERTHAN_LIMIT, 0.33,"LCL2", "LCL2");
let ucl4: QCSPCChartTS.SPCControlLimitRecord =
QCSPCChartTS.SPCControlLimitRecord.newSPCControlLimitRecordProcessVarAlarmTypeAlar
mValueAlarmMessages(chartdata,
QCSPCChartTS.SPCControlLimitRecord.SPC_GREATERTHAN_LIMIT, 1.66,"UCL2", "UCL2");

secondarychart.addAdditionalControlLimit(lcl4,
QCSPCChartTS.SPCChartObjects.SPC_LOWER_CONTROL_LIMIT_2, sigma2);
secondarychart.addAdditionalControlLimit(ucl4,
QCSPCChartTS.SPCChartObjects.SPC_UPPER_CONTROL_LIMIT_2, sigma2);

let lcl5: QCSPCChartTS.SPCControlLimitRecord =
QCSPCChartTS.SPCControlLimitRecord.newSPCControlLimitRecordProcessVarAlarmTypeAlar
mValueAlarmMessages(chartdata,
QCSPCChartTS.SPCControlLimitRecord.SPC_LOWERTHAN_LIMIT, 0.66,"LCL1", "LCL1");
let ucl5: QCSPCChartTS.SPCControlLimitRecord =
QCSPCChartTS.SPCControlLimitRecord.newSPCControlLimitRecordProcessVarAlarmTypeAlar
mValueAlarmMessages(chartdata,
QCSPCChartTS.SPCControlLimitRecord.SPC_GREATERTHAN_LIMIT, 1.33,"UCL1", "UCL1");

secondarychart.addAdditionalControlLimit(lcl5,
QCSPCChartTS.SPCChartObjects.SPC_LOWER_CONTROL_LIMIT_1, sigma1);
secondarychart.addAdditionalControlLimit(ucl5,
QCSPCChartTS.SPCChartObjects.SPC_UPPER_CONTROL_LIMIT_1, sigma1);

primarychart.setControlLimitLineFillMode( true);

secondarychart.setControlLimitLineFillMode( true);
```

Special Note – When you create a **SPCControlLimitRecord** object, you can specify an actual limit level. If you do not call the charts **autoCalculateControlLimits** method, the control limit will be displayed at that value. If you do call **autoCalculateControlLimits** method, the auto-calculated value overrides the initial value (0.0 in the examples above). When you call the charts **addAdditionalControlLimits** method, you specify the sigma level that is used by the **autoCalculateControlLimits** to calculate the control limit level.

If you want the control limits displayed as filled areas, set the charts **ControlLimitLineFillMode** property True.

[JavaScript / TypeScript]

```
primarychart.setControlLimitLineFillMode(true);
```

This will fill each control limit line from the limit line to the target value of the chart. In order for the fill to work properly, you must set this property after you define all additional control limits. Also, you must add the outer most control limits (SPC_UPPER_CONTROL_LIMIT_3 and SPC_LOWER_CONTROL_LIMIT_3) first, followed by the next outer most limits (SPC_UPPER_CONTROL_LIMIT_2 and SPC_LOWER_CONTROL_LIMIT_2), followed by the inner most control limits (SPC_UPPER_CONTROL_LIMIT_1 and SPC_LOWER_CONTROL_LIMIT_1). This way the fill of the inner limits will partially cover the fill of the outer limits, creating the familiar striped look you want to see.

Western Electric (WE) Runtime Rules

The normal SPC control limit rules display at the 3-sigma level, both high and low. In this case, a simple threshold test determines if a process is in, or out of control. Other, more complex tests rely on more complicated decision-making criteria. The most popular of these are the Western Electric Rules, also know as the WE Rules, or WE Runtime Rules. These rules utilize historical data for the eight most recent sample intervals and look for a non-random pattern that can signify that the process is out of control, before reaching the normal +-3 sigma limits. A processed is considered out of control if any of the following criteria are met:

*Starting with Revision 3.0, we have added additional, industry standard rules to the software: Nelson, AAIG, Juran, Hughes, Gitlow, Westgard, and Duncan. In addition, you can mix and match rules from the different rule sets, and you can create your own custom rules using our standardize rule templates. See Chapter 8, for more information.

1. **The most recent point plots outside one of the 3-sigma control limits.** If a point lies outside either of these limits, there is only a 0.3% chance that this was caused by the normal process.

2. **Two of the three most recent points plot outside and on the same side as one of the 2-sigma control limits.** The probability that any point will fall outside the warning limit is only 5%. The chances that two out of three points in a row fall outside the warning limit is only about 1%.

3. **Four of the five most recent points plot outside and on the same side as one of the 1-sigma control limits.** In normal processing, 68% of points fall within one sigma of the mean, and 32% fall outside it. The probability that 4 of 5 points fall outside of one sigma is only about 3%.

4. **Eight out of the last eight points plot on the same side of the center line, or target value.** Sometimes you see this as 9 out of 9, or 7 out of 7. There is an equal chance that any given point will fall above or below the mean. The chances that a point falls on the same side of the mean as the one before it is one in two. The odds that the next point will also fall on the same side of the mean is one in four. The probability of getting eight points on the same side of the mean is only around 1%.

These rules apply to both sides of the center line at a time. Therefore, there are eight actual alarm conditions: four for the above center line sigma levels and four for the below center line sigma levels.

There are also additional WE Rules for trending. These are often referred to as WE Supplemental Rules. Don't rely on the rule number, often these are listed in a different order.

5. **Six points in a row increasing or decreasing.** The same logic is used here as for rule 4 above. Sometimes this rule is changed to seven points rising or falling.

6. **Fifteen points in a row within one sigma.** In normal operation, 68% of points will fall within one sigma of the mean. The probability that 15 points in a row will do so, is less than 1%.

7. **Fourteen points in a row alternating direction.** The chances that the second point is always higher than (or always lower than) the preceding point, for all seven pairs is only about 1%.

8. **Eight points in a row outside one sigma.** Since 68% of points lie within one sigma of the mean, the probability that eight points in a row fall outside of the one-sigma line is less than 1%.

While the techniques in the previous section can be used to draw multiple SPC control limit lines on the graph, at the +-1, 2, 3 sigma levels for example, they do not provide for the (x out of y) control criteria used in evaluating the WE rules. The software can be explicitly flagged to evaluate out of control alarm conditions according to the WE Rules, instead of the default +-3 sigma control criteria. It will create alarm lines at the +-1, 2, and 3-sigma control limits and the center line. It will also automatically establish the eight alarm conditions associated with the WE rules. Set the WE rules flag using the PrimaryChart (or SecondaryChart) **useWERuntimeRules** method. When the variable control charts **autoCalculatedControlLimits** method is called, the software automatically calculates all of the appropriated control limits, based on the current data.

The example below is extracted from the NamedControlRules.BuildWECOChart example program.

If you want to include the WE Trending (Supplemental) rules, in addition to the regular WE Runtime rules, call useWERuntimeAndSupplementalRules in place of useWERuntimeRules.

[JavaScript]

```
 var htmlcanvas = document.getElementById(canvasid);
    var spccharttype = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART; // XBar-
R
    var subgroupsize = 5;
    var numberpointsinview = 12;
    var charttitle = "XBar-R Example";


    var xbarrchart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);

.
.
.


    xbarrchart.getChartData().setTitle (charttitle);
    xbarrchart.getChartData().setChartDescriptor("WECO Rules");
    xbarrchart.getPrimaryChart().useWERuntimeRules();

    var numssampleintervals = 100;
    var chartmean = 30;
    var chartsigma = 5;

    SimulateData(xbarrchart, numssampleintervals, chartmean, chartsigma);


    // Calculate the SPC control limits for both graphs of the current SPC chart
(X-Bar R)
    xbarrchart.autoCalculateControlLimits();
```

[TypeScript]

```
    let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
    let spccharttype: number  = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
// XBar-R
    let subgroupsize: number  = 5;
    let numberpointsinview: number  = 12;
    let charttitle: string = "XBar-R Example";


    let xbarrchart: QCSPCChartTS.SPCChartBase  =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
.
.
.


    let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarrchart.getChartData();
    if (chartdata) {
        chartdata.setTitle (charttitle);
```

```
        chartdata.setChartDescriptor("WECO Rules");
    }
    let primarychart: QCSPCChartTS.SPCChartObjects|null =
xbarrchart.getPrimaryChart();
    if (primarychart)
      primarychart.useWERuntimeRules();

    let  numssampleintervals: number = 100;
    let chartmean: number = 30;
    let chartsigma: number = 5;

    this.SimulateData(xbarrchart,numssampleintervals, chartmean, chartsigma);


    // Calculate the SPC control limits for both graphs of the current SPC chart
(X-Bar R)
    xbarrchart.autoCalculateControlLimits();
```

If you have setup a method for processing alarm events, the software will call the classes alarm processing method, where you can take appropriate action. If a time interval has multiple alarms, i.e. more than one of the four WR Runtime rules are broken, only the one with the lowest WE rule number is vectored to the alarm event processing routine.

[JavaScript]

```
import * as QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';

function alarmEventChanged( source,  e)
{

   var alarm = e.getEventAlarm();
   if (alarm)
   {
     var alarmlimitvalue = alarm.getControlLimitValue();
     var alarmlimitvaluestring =  alarmlimitvalue.toString();
     var spcData  = alarm.getSPCProcessVar();
   }
}

export async function BuildXBarRChart(canvasid) {

    var htmlcanvas = document.getElementById(canvasid);
    var spccharttype = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    var subgroupsize = 5;
    var numberpointsinview = 12;
    var charttitle = "XBar-R Example";


    var xbarrchart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
.
.
.


chartdata.addAlarmTransitionEventListener(alarmEventChanged);
chartdata.setAlarmTransitionEventEnable(true);


    primarychart.useWERuntimeRules();
    // Must have data loaded before any of the Auto.. methods are called
    this.SimulateData(100, 20);
```

```
    // Calculate the SPC control limits
    xbarrchart.autoCalculateControlLimits();

    // throw out values used to calculate limits
    chartdata.resetSPCChartData();

    // generate alarms now
    chartdata.setAlarmStateEventEnable( true);
    // Must have data loaded before any of the Auto.. methods are called
    // Check this data against rules
    this.SimulateData(100, 23);
}
```

[TypeScript]

```
export class VariableControlCharts implements QCSPCChartTS.SPCAlarmEventListener {


    public constructor() {

    }

alarmEventChanged(source: QCSPCChartTS.SPCControlChartData, e:
QCSPCChartTS.SPCControlLimitAlarmArgs): void
 {

    let alarm: QCSPCChartTS.SPCControlLimitRecord | null = e.getEventAlarm();
    if (alarm != null)
    {
      let alarmlimitvalue: number = alarm.getControlLimitValue();
      let alarmlimitvaluestring: string =  alarmlimitvalue.toString();
       let spcData: QCSPCChartTS.SPCControlChartData | null  =
alarm.getSPCProcessVar();
    }
}



public async  BuildXBarRChart(canvasid: string) {

    let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
    let spccharttype: number  = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    let subgroupsize: number = 5;
    let numberpointsinview: number  = 12;
    let charttitle: string = "XBar-R Example";
.
.
.

 chartdata.addAlarmTransitionEventListener(this);
chartdata.setAlarmTransitionEventEnable(true);


    primarychart.useWERuntimeRules();
    // Must have data loaded before any of the Auto.. methods are called
    this.SimulateData(100, 20);

    // Calculate the SPC control limits
    xbarrchart.autoCalculateControlLimits();

    // throw out values used to calculate limits
    chartdata.resetSPCChartData();

    // generate alarms now
    chartdata.setAlarmStateEventEnable( true);
```

```
    // Must have data loaded before any of the Auto.. methods are called
    // Check this data against rules
    this.SimulateData(100, 23);
}
```

Note that in the case of JavaScript, the **alarmEventChanged** function is just another function, with a specific parameter list signature. In JavaScript the **alarmEventChanged** function name is passed in as the argument to the SPC chart **addAlarmTransitionEventListener** method. That method is called when an alarm occurs. In the case of TypeScript, the entire encompassing class is marked as implementing **SPCAlarmEventListener**. When a class implements **SPCAlarmEventListener** it must have an **alarmEventChanged** function with the signature parameter list. For TypeScript, you pass in the *this* parameter to the **addAlarmTransitionEventListener** method**.** Since the class implements **alarmEventChanged**, the alarm event code looks up the **alarmEventChanged** method for the class, and that is what is executed when an alarm occurs.

If you want multiple alarms for a time interval vectored to the alarm processing routine (i.e. it is possible that a time period has WE1, WE2, WE3 and WE4 alarms), set the SPCControlChartData.AlarmReportMode property to SPCControlChartData.REPORT_ALL_ALARMS.

```
chartdata.setAlarmReportMode( QCSPCChartTS.SPCControlChartData.REPORT_ALL_ALARMS);
```

The resulting X-Bar R SPC Chart with WE Runtime Rules looks something like this. In this example, the WR Rules violations are processed by the **SPCControlLimitAlarm** method, where the alarm condition is added to the Notes record for the appropriate sample interval. The Y in the Notes line indicates that an alarm record has been saved for that time interval, and you can click on the Y to see the note describing the alarm condition.

Specification Limits

Specification limits are not to be confused with the SPC Control Limits discussed in the previous sections. Specification limits are imposed externally and are not calculated based on the manufacturing process under control. They represent the maximum deviation allowable for the process variable being measured. They are calculated based on input from customers and/or engineering. Usually specification limits are going to be

wider than the SPC 3-sigma limits, because you want the SPC control limits to trip before you get to the specification limits. The SPC control limits give you advance notice that the process is going south before you start rejecting parts based on specification limits. You can display specification limits in the same chart as SPC control limits. Use the addSpecLimit method of the PrimaryChart or SecondaryChart.

[JavaScript]

```
if (primarychart)
{
var initialLSL = 25;
var initialHSL = 35;

primarychart.addSpecLimit(QCSPCChartTS.SPCChartObjects.SPC_LOWER_SPEC_LIMIT,
initialLSL, "L SPEC",
QCSPCChartTS.ChartAttribute.newChartAttributeColorWidth(QCSPCChartTS.ChartColor.GR
EEN, 3.0));
primarychart.addSpecLimit(QCSPCChartTS.SPCChartObjects.SPC_UPPER_SPEC_LIMIT,
initialHSL, "H
SPEC",QCSPCChartTS.ChartAttribute.newChartAttributeColorWidth(QCSPCChartTS.ChartCo
lor.YELLOW, 3.0));

}
```

[TypeScript]

```
if (primarychart)
{
  let initialLSL: number = 25;
  let initialHSL: number = 35;

  primarychart.addSpecLimit(QCSPCChartTS.SPCChartObjects.SPC_LOWER_SPEC_LIMIT,
initialLSL, "L SPEC",
QCSPCChartTS.ChartAttribute.newChartAttributeColorWidth(QCSPCChartTS.ChartColor.GR
EEN, 3.0));
  primarychart.addSpecLimit(QCSPCChartTS.SPCChartObjects.SPC_UPPER_SPEC_LIMIT,
initialHSL, "H
SPEC",QCSPCChartTS.ChartAttribute.newChartAttributeColorWidth(QCSPCChartTS.ChartCo
lor.YELLOW, 3.0));

}
```

Chart Y-Scale

You can set the minimum and maximum values of the two charts y-scales manually using the **PrimaryChart.MinY**, **PrimaryChart.MaxY**, **SecondaryChartMinY** and **SecondaryChartMaxY** properties.

[JavaScript / TypeScript]

```
// Set initial scale of the y-axis of the mean chart
// If you are calling autoScalePrimaryChartYRange this isn't really needed
    primarychart.setMinY( 0);
    primarychart.setMaxY (40);

// Set initial scale of the y-axis of the range chart
// If you are calling autoScaleSecondaryChartYRange this isn't really needed
    secondarychart.setMinY(0);
```

```
    secondarychart.setMaxY(10);
```

It is easiest to just call the auto-scale routines after the chart has been initialized with data, and any control limits calculated.

[JavaScript]

```
// Must have data loaded before any of the Auto.. methods are called
    SimulateData(xbarrchart,numssampleintervals, chartmean, chartsigma);

// Calculate the SPC control limits for both graphs of the current SPC chart
    xbarrchart.autoCalculateControlLimits();

// Scale the y-axis of the X-Bar chart to display all data and control limits
    xbarrchart.autoScalePrimaryChartYRange();
// Scale the y-axis of the Range chart to display all data and control limits
    xbarrchart.autoScaleSecondaryChartYRange();
```

[TypeScript]

```
// Must have data loaded before any of the Auto.. methods are called
    this.SimulateData(xbarrchart,numssampleintervals, chartmean, chartsigma);

// Calculate the SPC control limits for both graphs of the current SPC chart
    xbarrchart.autoCalculateControlLimits();

// Scale the y-axis of the X-Bar chart to display all data and control limits
    xbarrchart.autoScalePrimaryChartYRange();
// Scale the y-axis of the Range chart to display all data and control limits
    xbarrchart.autoScaleSecondaryChartYRange();
```

Once all of the graph parameters are set, call the method **rebuildChartUsingCurrentData**.

[JavaScript / TypeScript]

```
// Rebuild the chart using the current data and settings
xbarrchart.rebuildChartUsingCurrentData();
```

If, at any future time you change any of the chart properties, you will need to call **rebuildChartUsingCurrentData** to force a rebuild of the chart, taking into account the current properties. **rebuildChartUsingCurrentData** also invalidates the chart and forces a redraw. Our examples that update dynamically demonstrate this technique. The chart is setup with some initial settings and data values. As data is added in real-time to the graph, the chart SPC limits, and y-scales are constantly recalculated to take into account new data values. The following code is extracted from the **RealTimeSPCChartUpdate** example. The actual example is a little more complicated because it takes into account two running charts.

[JavaScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
```

```
    var timerId = 0;
    var spcrtvarchart = null;
    var spcrtattribchart = null;
    var ontopspcchart = null;
    var timercount = 0;
    var timercountlimit = 0;


    export async function  BuildRealTimeVarSPCChart(canvasid) {

        var htmlcanvas = document.getElementById(canvasid);
        var spccharttype = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
        var subgroupsize = 5;
        var numberpointsinview = 12;
        var charttitle = "Real-Time SPC Chart";
        var xbarrchart  =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
        if (!xbarrchart) return;

        ontopspcchart = spcrtvarchart = xbarrchart;

.
.
.

    }


    function TimerTickProc()
    {
        var numssampleintervals = 1;
        var chartmean = 30;
        var chartsigma = 5;
        var fractiondefective = 0.134;
        var subgroupsize = 100;
         if (spcrtvarchart)
         {
            SimulateVarData(spcrtvarchart, numssampleintervals, chartmean,
chartsigma);
            if (ontopspcchart == spcrtvarchart)
              spcrtvarchart.rebuildChartUsingCurrentData();
         }
         if (spcrtattribchart)
         {
            SimulateAttribData(spcrtattribchart, numssampleintervals,
fractiondefective, subgroupsize);
            if (ontopspcchart == spcrtattribchart)
              spcrtattribchart.rebuildChartUsingCurrentData();
         }
         timercount++;
         if (timercount > timercountlimit )
            StopTimer();
    }

    export async function StartTimer (milliseconds, count)
    {
       if (timerId != 0)
         StopTimer();
       timercountlimit = count;
       timerId = setInterval(() =>
       TimerTickProc(),
       milliseconds);
    }

    export async function  StopTimer ()
    {
        clearInterval(timerId);
        timerId = 0;
    }
```

```
function SimulateVarData(spcchart  , count, mean, sigma) {
        // batch number for a given sample subgroup
        var batchCounter = 0;
        var i = 0;
        var timestamp = new Date();
        if (!spcchart) return;
        var chartdata= spcchart.getChartData();
        if (!chartdata) return;
        var currentcount = chartdata.getCurrentNumberRecords();
        if (currentcount > 0) // start date at the previous ending date plut time
increment
        {
            var ts  = chartdata.getTimeValue(currentcount-1);
            timestamp = ts;
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }
        for (i = 0; i < count; i++) {
            // Simulate a sample subgroup record
            var samples = chartdata.simulateMeasurementRecordMeanRange(mean,
sigma);
            // Update chart data using i as the batch number
            batchCounter = currentcount + i;
            var note = "";
            if ((i % 5) == 0) note = "This is a note";
            // Add a new sample record to the chart data
            chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, note);
            // Simulate passage of timeincrementminutes minutes
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }

    }
```

[TypeScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';


export class RealTimeSPCChartUpdate {

    timerId: number = 0;
    spcrtvarchart: QCSPCChartTS.SPCChartBase | null = null;
    spcrtattribchart: QCSPCChartTS.SPCChartBase | null = null;
    ontopspcchart: QCSPCChartTS.SPCChartBase | null = null;
    timercount: number = 0;
    timercountlimit: number = 0;

    public constructor() {

    }


    public async  BuildRealTimeVarSPCChart(canvasid: string) {

        let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
        let spccharttype: number =
QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
        let subgroupsize: number = 5;
        let numberpointsinview: number = 12;
        let charttitle: string = "Real-Time SPC Chart";
        let xbarrchart: QCSPCChartTS.SPCChartBase  =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
        if (!xbarrchart) return;

        this.ontopspcchart = this.spcrtvarchart = xbarrchart;
```

```
.
.
.
 }


    TimerTickProc()
    {
        let numssampleintervals: number = 1;
        let chartmean: number = 30;
        let chartsigma: number = 5;
        let fractiondefective: number = 0.134;
        let subgroupsize: number = 100;
         if (this.spcrtvarchart)
         {
            this.SimulateVarData(this.spcrtvarchart, numssampleintervals,
chartmean, chartsigma);
            if (this.ontopspcchart == this.spcrtvarchart)
              this.spcrtvarchart.rebuildChartUsingCurrentData();
         }
         if (this.spcrtattribchart)
         {
            this.SimulateAttribData(this.spcrtattribchart, numssampleintervals,
fractiondefective, subgroupsize);
            if (this.ontopspcchart == this.spcrtattribchart)
              this.spcrtattribchart.rebuildChartUsingCurrentData();
         }
         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;
    }

    SimulateVarData(spcchart: QCSPCChartTS.SPCChartBase | null , count: number,
mean: number, sigma: number) {
        // batch number for a given sample subgroup
        let batchCounter = 0;
        let i = 0;
        let timestamp = new Date();
        if (!spcchart) return;
        let chartdata: QCSPCChartTS.SPCControlChartData | null =
spcchart.getChartData();
        if (!chartdata) return;
        let currentcount: number = chartdata.getCurrentNumberRecords();
        if (currentcount > 0) // start date at the previous ending date plut time
increment
        {
            let ts : Date = chartdata.getTimeValue(currentcount-1);
            timestamp = ts;
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }
```

```
        for (i = 0; i < count; i++) {
            // Simulate a sample subgroup record
            let samples: QCSPCChartTS.DoubleArray =
chartdata.simulateMeasurementRecordMeanRange(mean, sigma);
            // Update chart data using i as the batch number
            batchCounter = currentcount + i;
            let note: string = "";
            if ((i % 5) == 0) note = "This is a note";
            // Add a new sample record to the chart data
            chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, note);
            // Simulate passage of timeincrementminutes minutes
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }

    }

}
```

Updating Chart Data

The real-time example above demonstrates how the SPC chart data is updated, using one of the **chartdata.addNewSampleRecord...** methods. In this case, the chart data updates with each timer tick event, though it could just as easily be any other type of event. If you have already collected all of your data and just want to plot it all at once, use a simple loop like most of our examples do to update the data.

[JavaScript]

```
function SimulateData(spcchart, count, mean, sigma) {
    // batch number for a given sample subgroup
    var batchCounter = 0;
    var i;
    var timestamp = new Date();
    if (!spcchart) return;
    if (!spcchart.getChartData()) return;
    var da = new QCSPCChartTS.DoubleArray();

    var currentcount = spcchart.getChartData().getCurrentNumberRecords();
    for (i = 0; i < count; i++) {
        // Simulate a sample subgroup record
        var samples =
spcchart.getChartData().simulateMeasurementRecordMeanRange(mean, sigma);
        // Update chart data using i as the batch number
        batchCounter = currentcount + i;
        var note = "";
        if ((i % 5) == 0) note = "This is a note";
        // Add a new sample record to the chart data

spcchart.getChartData().addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter
, timestamp, samples, note);
        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }

}
```

[TypeScript]

```
SimulateData(spcchart: QCSPCChartTS.SPCChartBase, count: number, mean: number,
sigma: number)  {
    // batch number for a given sample subgroup
    let batchCounter: number = 0;
    let i: number = 0;
    let timestamp: Date = new Date();
```

```
    let chartdata: QCSPCChartTS.SPCControlChartData | null =
spcchart.getChartData();
    if (!chartdata) return;
    let currentcount = chartdata.getCurrentNumberRecords();

    for (i = 0; i < count; i++) {
        // Simulate a sample subgroup record
        let samples: QCSPCChartTS.DoubleArray =
chartdata.simulateMeasurementRecordMeanRange(mean, sigma);
        // Update chart data using i as the batch number
        batchCounter = currentcount + i;
        let note: string = "";
        if ((i % 5) == 0) note = "This is a note";
        // Add a new sample record to the chart data
        chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, note);
        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }

}
```

In this example the sample data and the time stamp for each sample record is simulated. In your application, you will probably be reading the sample record values from some sort of database or file, along with the actual time stamp for that data.

If you want to include a text note in the sample record, use one of the **chartdata.addNewSampleRecord...** methods that have a *notes* parameter.

[JavaScript]

```
function SimulateData(spcchart, count, mean, sigma) {
    // batch number for a given sample subgroup
    var batchCounter = 0;
    var i;
    var timestamp = new Date();
    if (!spcchart) return;
    if (!spcchart.getChartData()) return;
;

    var currentcount = spcchart.getChartData().getCurrentNumberRecords();
    for (i = 0; i < count; i++) {
        var notesstring = "";
       var timestamp = startTime;
       // Use the ChartData sample simulator to make an array of sample data
       var samples = chartdata.simulateMeasurementRecord(mean, sigma);
        batchCounter = currentcount + i;
       var r = QCSPCChartTS.ChartSupport.getRandomDouble();
       if (r < 0.1) // make a note on every tenth item, on average
         notesstring = "Note for sample subgroup #" +
                     i.toString() +
                     ". This sample is flagged as having some sort of problem";
         else
           notesstring = "";
               // Add the new sample subgroup to the chart
        chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, notestring);
       // increment simulated time by timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }
}
```

[TypeScript]

```
SimulateData(spcchart: QCSPCChartTS.SPCChartBase, count: number, mean: number,
sigma: number)  {
    // batch number for a given sample subgroup
    let batchCounter: number = 0;
    let i: number = 0;
    let timestamp: Date = new Date();
    let chartdata: QCSPCChartTS.SPCControlChartData | null =
spcchart.getChartData();
    if (!chartdata) return;
    let currentcount = chartdata.getCurrentNumberRecords();

    for (i = 0; i < count; i++) {
        // Simulate a sample subgroup record
        let samples: QCSPCChartTS.DoubleArray =
chartdata.simulateMeasurementRecordMeanRange(mean, sigma);
        // Update chart data using i as the batch number
        batchCounter = currentcount + i;
        let note: string = "";
        if ((i % 5) == 0) note = "This is a note";
        // Add a new sample record to the chart data
        chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, note);
        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }

}
```

There are situations where you might want to add, change, modify, or append a note for a sample subgroup after the **addNewSampleRecord** method has already been called for the sample subgroup. This can happen if the **addNewSampleRecord** method call generates an alarm event. In the alarm event processing routine, you can add code that adds a special note to the sample subgroup that generated the alarm. Use the **chartdata.setNotesString** or **chartdata.appendNotesString** methods to add notes to the current record, separate from the **addNewSampleRecord** method.

[JavaScript]

```
function alarmEventChanged( source,  e)
 {

    var alarm: QCSPC = e.getEventAlarm();
    if (alarm != null)
    {
      var alarmlimitvalue= alarm.getControlLimitValue();
      var alarmlimitvaluestring =  alarmlimitvalue.toString();
       var spcData = alarm.getSPCProcessVar();

      var spcSource = e.getSPCSource();
      if (spcSource && spcData)
      {
         var calculatedvaluestring = spcSource.getCalculatedValue().toString();

        var message  = alarm.getAlarmMessage();
        var timestamp = spcData.getTimeStamp();
        var timestampstring = timestamp.toString();
```

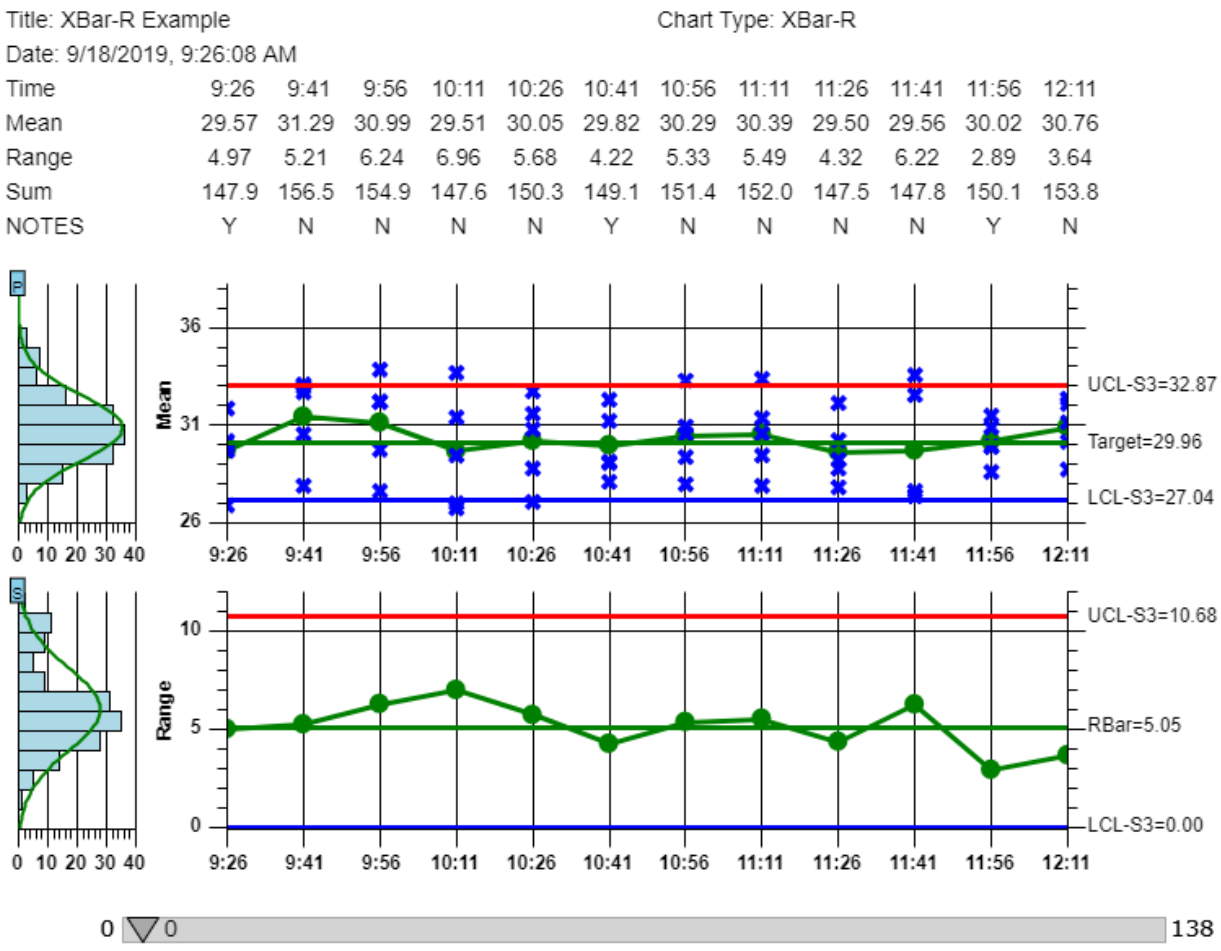
```
      var notesstring = "/n" + timestampstring + " " + message + "=" +
        "/n" +  alarmlimitvaluestring + " Current Value" + "=" +
        calculatedvaluestring;

      // Append a notes string to the current record.
      if (alarm.getAlarmState())
        chartdata.appnedNotesString(notesstring, true);
      }
    }
}
```

[TypeScript]

```
 alarmEventChanged(source: QCSPCChartTS.SPCControlChartData, e:
QCSPCChartTS.SPCControlLimitAlarmArgs): void
 {

    let alarm: QCSPCChartTS.SPCControlLimitRecord | null = e.getEventAlarm();
    if (alarm != null)
    {
      let alarmlimitvalue: number = alarm.getControlLimitValue();
      let alarmlimitvaluestring: string =  alarmlimitvalue.toString();
      let spcData: QCSPCChartTS.SPCControlChartData | null  =
alarm.getSPCProcessVar();

      let spcSource: SPCCalculatedValueRecord | null = e.getSPCSource();
      if (spcSource && spcData)
      {
         let calculatedvaluestring : string =
spcSource.getCalculatedValue().toString();

        let message :string = alarm.getAlarmMessage();
        let timestamp: Date = spcData.getTimeStamp();
        let timestampstring: string = timestamp.toString();

      let notesstring: string= "/n" + timestampstring + " " + message + "=" +
        "/n" +  alarmlimitvaluestring + " Current Value" + "=" +
        calculatedvaluestring;

      // Append a notes string to the current record.
      if (alarm.getAlarmState())
        chartdata.appnedNotesString(notesstring, true);
      }
    }
}
```

Scatter Plots of the Actual Sampled Data

If you want the actual sample data plotted along with the mean or median of the sample data, set the **PrimaryChart.PlotMeasurementValues** property to true.



Scatter plot of raw data for the chart

[JavaScript / TypeScript]

```
// Plot individual sampled values as a scatter plot
primarychart.setPlotMeasurementValues (true);
```

Enable the Chart ScrollBar

Set the **EnableScrollBar** property true to enable the chart scrollbar. You will then be able to window in on 8-20 sample subgroups at a time, from a much larger collection of measurement data representing hundreds or even thousands of subgroups, and use the scrollbar to move through the data, similar to paging through a spreadsheet.

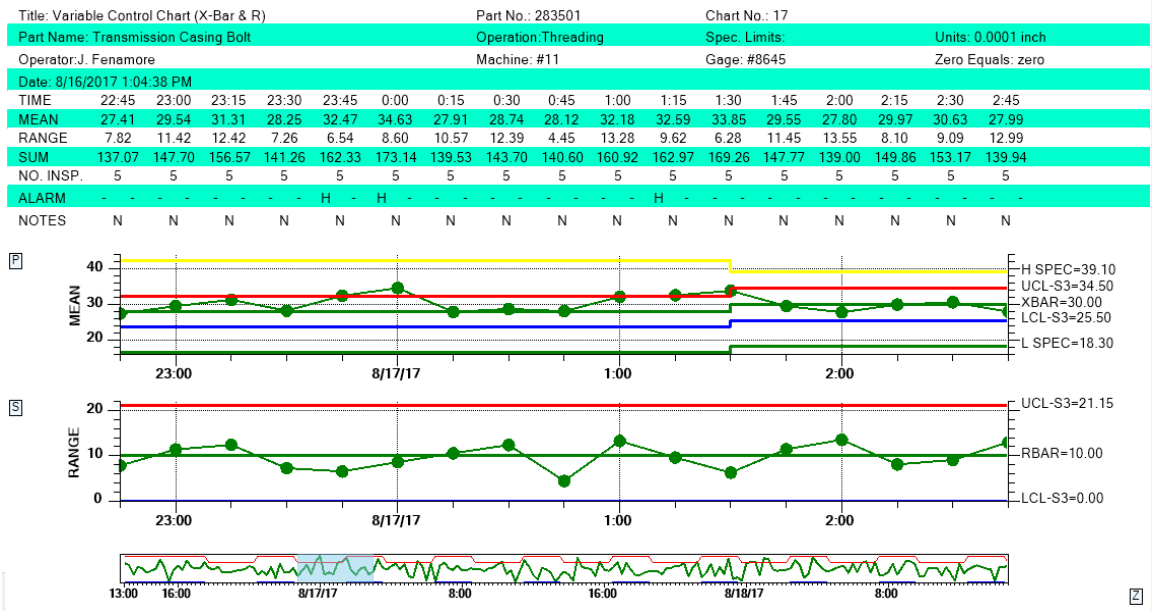
[JavaScript / TypeScript]

```
// enable scroll bar
xbarchart.setEnableScrollBar(true);
```

Once you have initialized the chart with data, and the scrollbar has a range associated with it, you can access the scrollbar using the charts **`HScrollBar1`** property.

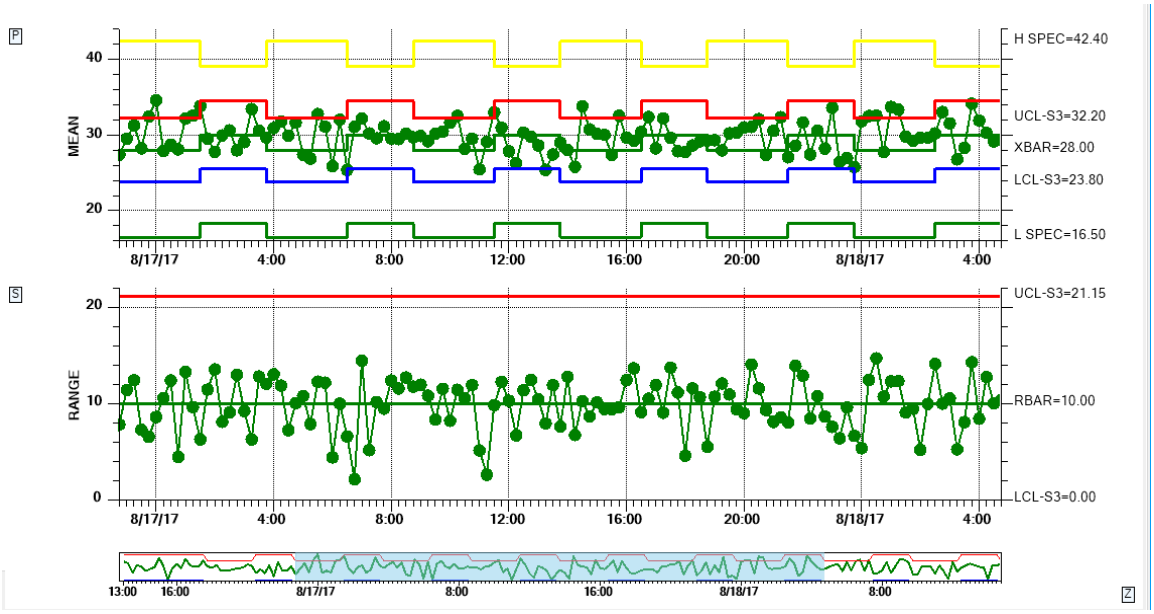
Zooming as an option for the scrollbar

The horizontal scrollbar can be replaced (toggled on/off actually) using the UI, with a zoom window, by clicking on the z-button in the lower right corner. The user will be able to scroll the charts by dragging the transparent blue zoom window left or right, or expand the x-axis scale by dragging the left or right edge of the zoom window.



Click on the Z button in the lower right corner and a zoom window will replace the scrollbar. Use the mouse to scroll the charts by dragging the transparent blue zoom window left or right, or expand the x-axis scale by dragging the left or right edge of the zoom window.

If the number of points in the display exceeds the initial setup value for the number of data points, the table is temporarily removed to prevent overlap of the table columns. If the number of data points is reduced to <= the initial number of points, the table will reappear.



If you use the zoom window to display a lot of points, the table at the top will disappear to prevent the table columns from overlapping.

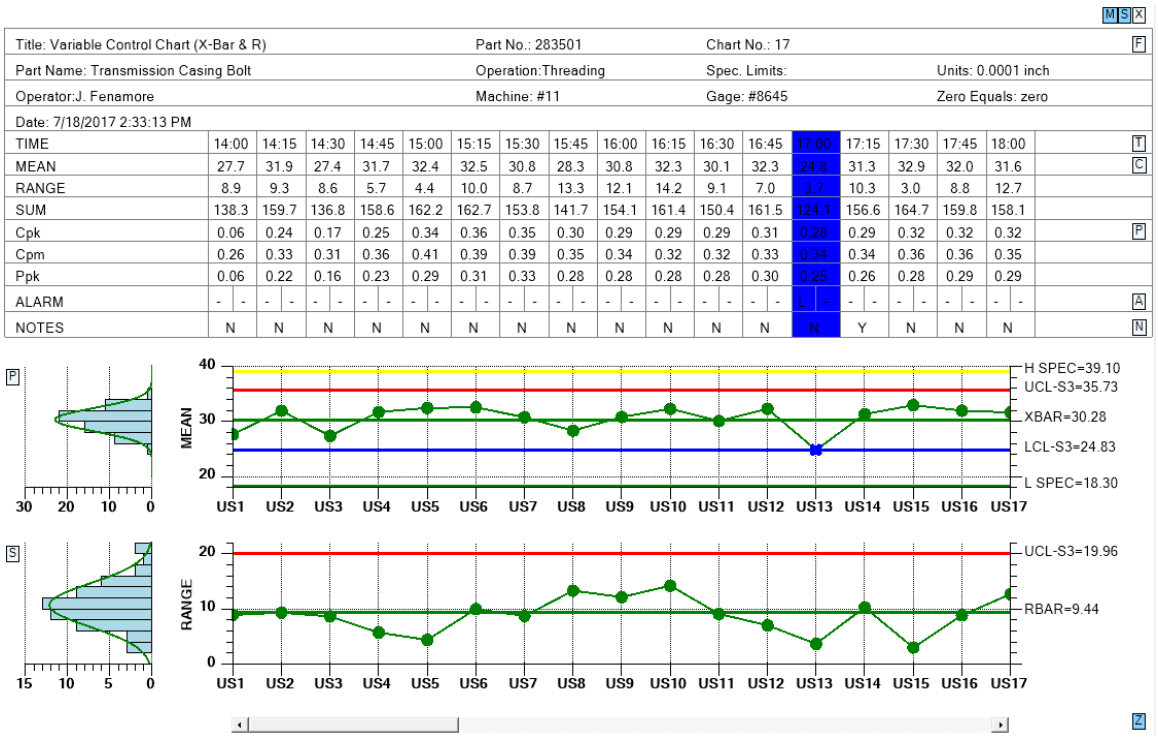
The default display display for all chart types uses the scrollbar. The zoom option is seen as a small button with the character 'Z' in the lower right corner of the display. You enter/ exit the zoom mode by clicking on that button. If you do not want the button to show at all, effectively making the zoom option inaccessible to the end user, set EnableZoomToggles property false.

.[JavaScript / TypeScript]

```
xbarrchart.setEnableZoomToggles(false);
```

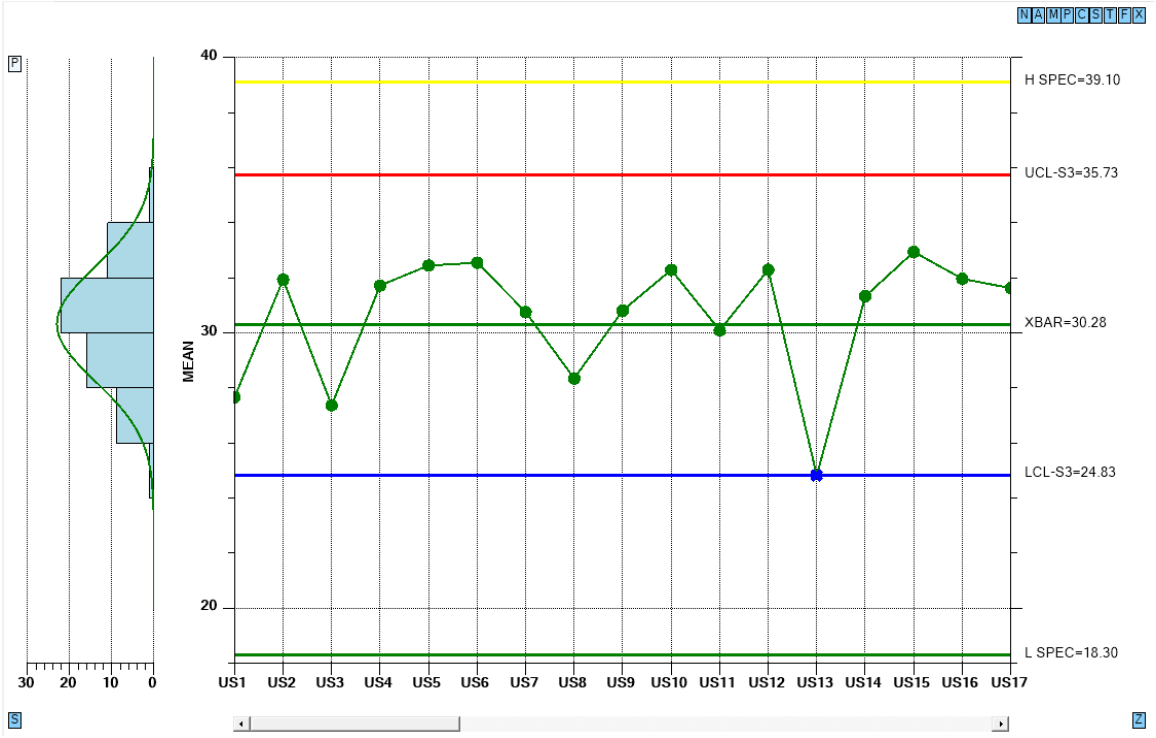
Collapsible Items

Like the Zoom option described in the previous section, there are also UI buttons which can toggle on and off rows in the table, and the Primary and Secondary charts. Like the Zoom button, these UI buttons can be hidden from the end user if you don't want them to show. See the end of this section for examples. On the top and right of the table buttons will selectively collapse/restore rows of the table, freeing up more space for charts. Buttons to the left and bottom of the Primary and Secondary charts also permit the user to collapse/restore either of those charts, allowing the remaining chart to display in all of the remaining space.



Buttons on the top, right and left of the display permit the user to selectively collapse portions of the chart.

Click on the N, A, M, P C and S buttons and shrink the table to following size.



In the example above, all of the chart options except for the Primary chart, have been toggled off using the buttons.

The buttons at the right of the table (and at the top right if toggled off) use the following ID's.

X	Turn on/off all table items at once
F	Turn on/off form data
T	Turn on/off sample interval time stamp data
S	Turn on/off sample value data
C	Turn on/off calculated value (mean, range, sum, etc.) data
P	Turn on/off process capability data
M	Turn on/off number of samples data
A	Turn on/off alarm data
N	Turn on/off notes data
Z	Bottom right - Turn on/off zoom control – the only button not affected by the X button above

The buttons at the left of the primary and secondary charts use the following ID's.

P	Turn on/off the Primary chart
S	Turn on/off the Secondary chart

If you do not want the buttons to show at all, effectively making these UI options inaccessible to the end user, set these properties false.

.

Hide the chart buttons on the left.

.[JavaScript / TypeScript]

```
xbarrchart.setEnableChartToggles(false);
```

Hide the table buttons on the right.

.[JavaScript / TypeScript]

```
xbarrchart.setEnableTableRowToggles(false);
```

You can hide ALL of the UI buttons (zoom, chart, and table) using the property EnableDisplayOptionToggles. This is what you use if you do not want the end user to have any control over the display of the table and chart.

.[JavaScript / TypeScript]

```
xbarrchart.setEnableDisplayOptionToggles(false);
```

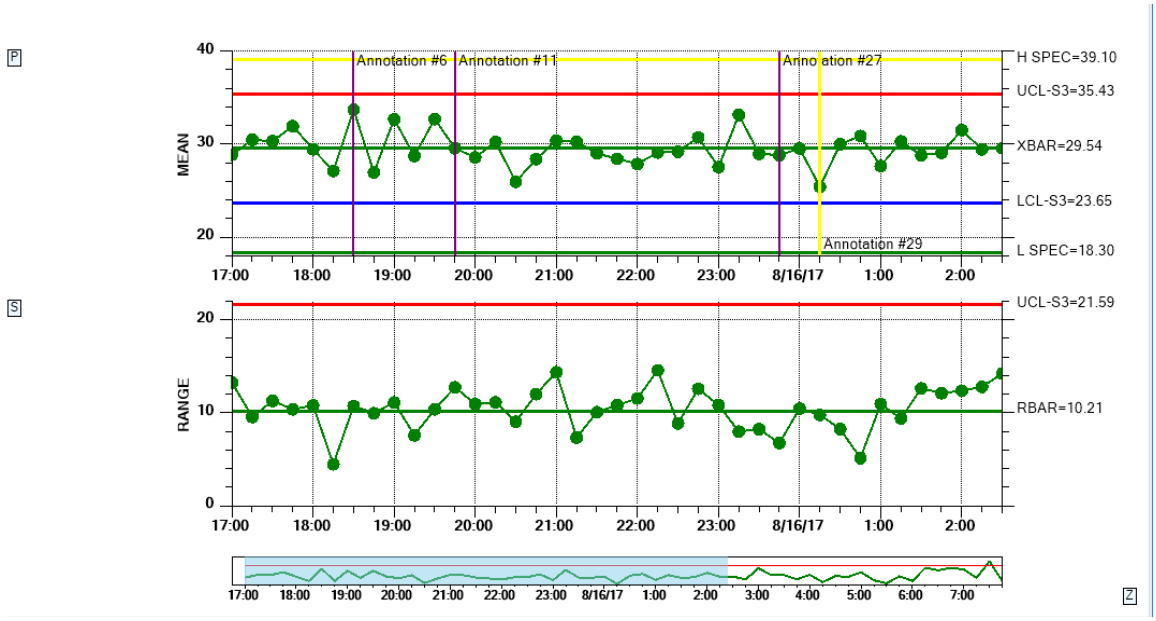
If you want to selectively enable options, first set EnableDisplayOptionToggles true. The other button display enables won't work unless this option is true. Then disable the options you do not want to show. In the example below, only the zoom option is left visible.

.[JavaScript / TypeScript]

```
xbarrchart.setEnableDisplayOptionToggles (true);
xbarrchart.setEnableTableRowToggles(false);
xbarrchart.setEnableChartToggles(false)
xbarrchart.setEnableZoomToggle(true);
```

Enhanced Annotations

The chart annotations have been enhanced with a vertical line with accompanying text using programmer specified justification.



The enhanced chart annotations include a vertical line to mark the data point, and many justification options (top, middle, bottom, left, right and center).

addAnnotation

Add an annotation to a data point in the specified SPC chart.

```
public addAnnotationChartNumIndexString(chart: number, datapointindex: number,
text: string): number

public addAnnotationChartNumIndexChartText(chart: number, datapointindex: number,
textobj: ChartText): number
```

```
public addAnnotationChartNumXDateYNumChartText(chart: number, x: Date, y: number,
textobj: ChartText): number

public addAnnotationChartNumIndexStringJustAttrib(chart: number, datapointindex:
number, text: string, just: number, attrib: ChartAttribute): number

public addAnnotationChartNumStringJustAttrib(chart: number, text: string, just:
number, attrib: ChartAttribute): number
```

where:

chart	Specifies whether the annotation is added to the primary, or secondary chart. Use one of the SPChartObjects constants: SPCChartObjects.PRIMARY_CHART or SPCChartObjects.SECONDARY_CHART.
datapointindex	The index of the data point the annotation is for.
text	A string string representing the annotation.
just	The justification for the text to the x-position of the annotation. Use one of the annotation justification constants: ANNOTATION_UPPER_RIGHT, ANNOTATION_UPPER_LEFT, ANNOTATION_LOWER_RIGHT, ANNOTATION_LOWER_LEFT, ANNOTATION_UPPER_CENTER, ANNOTATION_LOWER_CENTER, ANNOTATION_DATAPOINT_RIGHT, ANNOTATION_DATAPOINT_LEFT
attrib	A the attribute of the vertical line.

You call the addAnnotation method immediately after the addNewSampleRecord method call. For Example:

[JavaScript]

```
                // Add the new sample subgroup to the chart
      chartdata.addNewSampleRecordDateSamplesNotes(timestamp, samples,
notesstring);

      var index = chartdata.getCurrentNumberRecords() - 1;
      var annotstring = "Annotation #" + index.toString();
      var lineattrib = QCSPCChartTS.ChartAttribute.newChartAttributeColorWidth
(QCSPCChartTS.ChartColor.PURPLE, 2);

      // Adjust justification to minimize overlap of adjacent annotations
      var annotjust = QCSPCChartTS.SPCAnnotation.ANNOTATION_UPPER_LEFT;

      xbarrchart.addAnnotation(QCSPCChartTS.SPCChartObjects.PRIMARY_CHART, index,
        annotstring, annotjust, lineattrib);
```

[TypeScript]

```
                // Add the new sample subgroup to the chart
```

```
      chartdata.addNewSampleRecordDateSamplesNotes(timestamp, samples,
notesstring);

      let index: number = chartdata.getCurrentNumberRecords() - 1;
      let annotstring: string = "Annotation #" + index.toString();
      let lineattrib: QCSPCChartTS.ChartAttribute =
QCSPCChartTS.ChartAttribute.newChartAttributeColorWidth
(QCSPCChartTS.ChartColor.PURPLE, 2);

      // Adjust justification to minimize overlap of adjacent annotations
      let annotjust: number = QCSPCChartTS.SPCAnnotation.ANNOTATION_UPPER_LEFT;

      xbarrchart.addAnnotation(QCSPCChartTS.SPCChartObjects.PRIMARY_CHART, index,
        annotstring, annotjust, lineattrib);
```

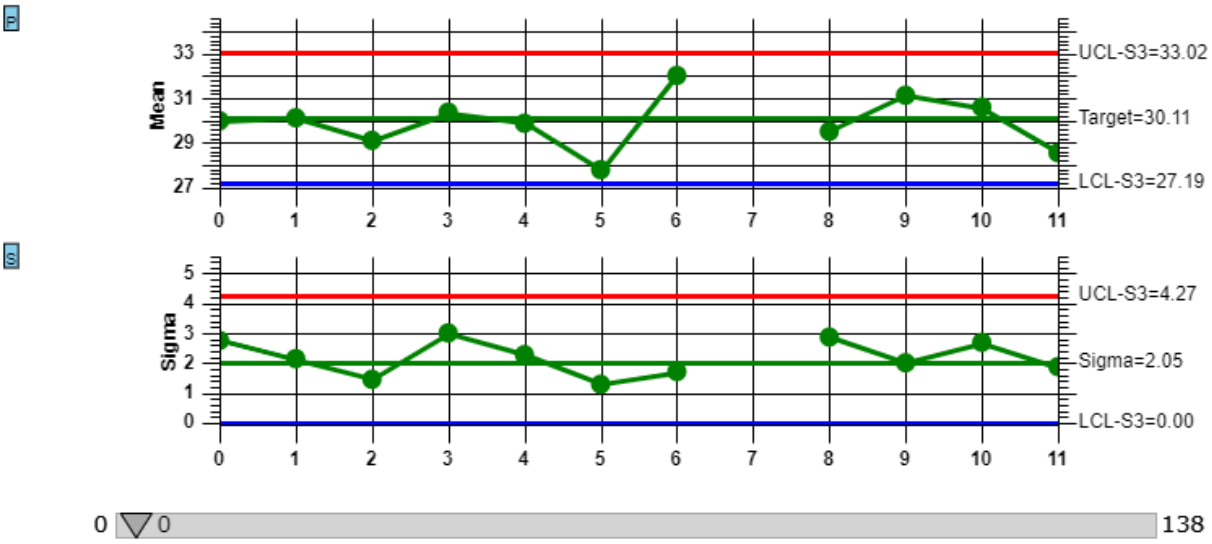
SPC Chart Histograms

Viewing frequency histograms of both the variation in the primary variable (Mean, Median, count, or actual value), and the secondary variable (Range, Sigma or Moving Range) side-by-side with the SPC control chart makes it even easier to find out whether the variations are the result of natural variations or the result of some systemic change in the process. You can turn on integrated frequency histograms for either chart using the **PrimaryChart.DisplayFrequencyHistogram** and **SecondaryChart.DisplayFrequencyHistogram** properties of the chart.

The histograms are turned off by default, but you can turn them off if you want.

Title: XBar-Sigma Example Chart Type: XBar-Sigma
Date: 9/18/19

Time	9:31	9:46	10:01	10:16	10:31	10:46	11:01	11:16	11:31	11:46	12:01	12:16
Mean	29.98	30.12	29.10	30.37	29.87	27.78	32.02	29.84	29.51	31.13	30.60	28.54
Sigma	2.76	2.15	1.46	3.01	2.28	1.29	1.72	1.42	2.87	2.01	2.70	1.88
Sum	149.9	150.6	145.5	151.9	149.4	138.9	160.1	149.2	147.5	155.6	153.0	142.7
Cpk	0.445	0.448	0.460	0.432	0.427	0.404	0.469	0.485	0.453	0.474	0.471	0.454
Cpm	0.543	0.542	0.567	0.535	0.530	0.522	0.571	0.586	0.557	0.572	0.566	0.556
Ppk	0.360	0.436	0.438	0.424	0.432	0.379	0.416	0.435	0.420	0.440	0.444	0.427
Alarm	-	-	-	-	-	-	-	-	-	-	-	-
Notes	Y	N	N	N	N	Y	N	N	N	N	Y	N



Xbar-Sigma chart with the Histograms turned off

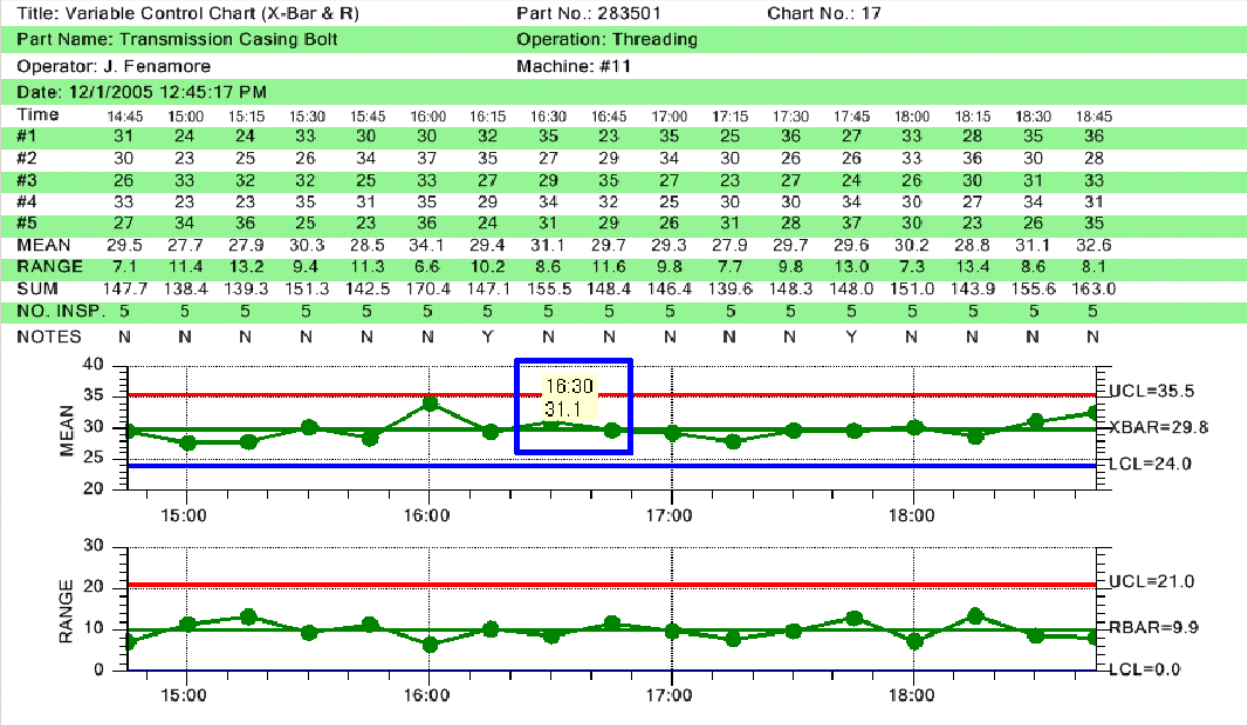
.[JavaScript / TypeScript]

```
//  frequency histogram for both charts
primarychart.setDisplayFrequencyHistogram( false);
secondarychart.setDisplayFrequencyHistogram( false);
```

SPC Chart Data and Notes Tooltips

You can invoke two types of tooltips using the mouse. The first is a data tooltip. When you hold the mouse button down over one of the data points, in the primary or secondary chart, the x and y values for that data point display in a popup tooltip.

Data Tooltip



In the default mode, the data tooltip displays the x,y value of the data point nearest the mouse click. If the x-axis is a time axis then the x-value is displayed as a time stamp; otherwise, it is displayed as a simple numeric value, as is the y-value. You can optionally display subgroup information (sample values, calculated values, process capability values and notes) in the data tooltip window, under the x,y value, using enable flags in the primary charts tooltip property.

Extracted from the MultiDivCharts.BuildXBarRChart example.

[JavaScript]

```
if (primarychart)
{
    var datatooltip = primarychart.getDatatooltip();
    if (datatooltip)
    {
      datatooltip.setEnableCategoryValues( true);
      datatooltip.setEnableProcessCapabilityValues( true);
      datatooltip.setEnableCalculatedValues(true);
      datatooltip.setEnableNotesString ( true);
    }
}
```

[TypeScript]

```
let primarychart: QCSPCChartTS.SPCChartObjects | null =
xbarrchart.getPrimaryChart();
if (primarychart)
{
    let datatooltip: QCSPCChartTS.SPCDataToolTip | null =
primarychart.getDatatooltip();
```

```
    if (datatooltip)
    {
      datatooltip.setEnableCategoryValues( true);
      datatooltip.setEnableProcessCapabilityValues( true);
      datatooltip.setEnableCalculatedValues(true);
      datatooltip.setEnableNotesString ( true);
    }
}
```

where

setEnableCategoryValues
setEnableProcessCapabilityValues
setEnableCalculatedValues
setEnableNotesStrings

Display the category (subgroup sample values) in the data tooltip.

```
datatooltip.setEnableCategoryValues(true);
```

Display the calculated values used in the chart (Mean, range and sum for an Mean-Range chart).

```
datatooltip.setEnableCalculatedValues(true);
```

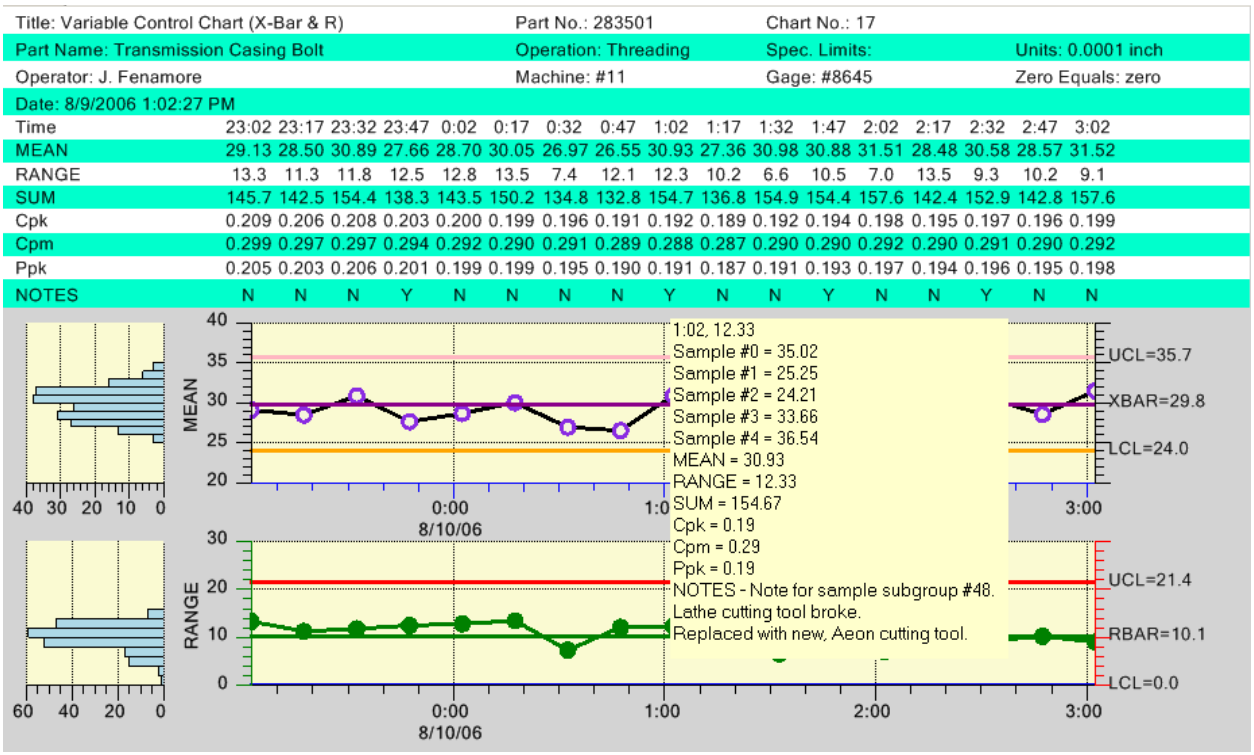
Display the process capability (Cp, Cpl, Cpu, Cpk, Cpm, Pp, Ppl, Ppu and Ppk) statistics currently being calculated for the chart.

```
datatooltip.setEnableProcessCapabilityValues(true);
```

Display the current notes string for the sample subgroup.

```
datatooltip.setEnableCategoryValues(true);
```

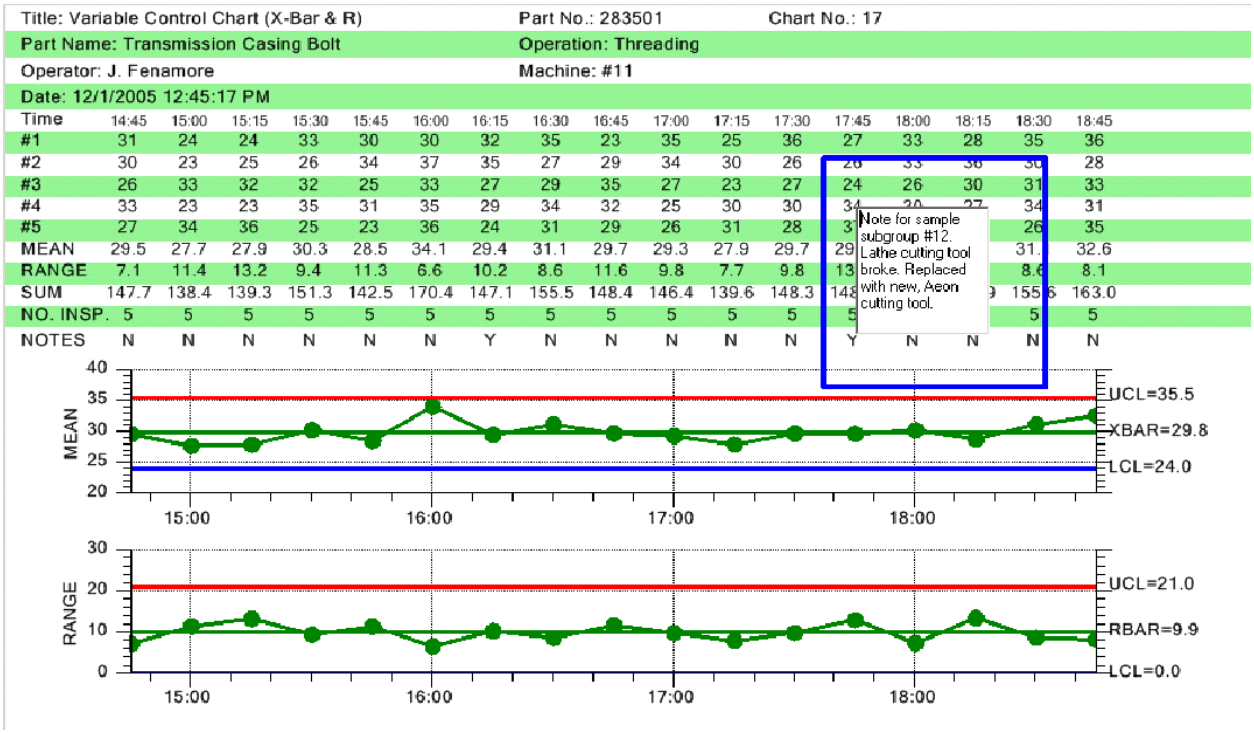
The variable control chart below displays a tooltip with all of the enable options above set true.



Data Tooltip with optional display items

If you are displaying the Notes line in the table portion of the chart, the Notes entry for a sample subgroup will display “Y” if a note was recorded for that sample subgroup, or “N” if no note was recorded. Notes are recorded using one of the **chartdata.addNewSampleRecord...** methods that include a notes parameter, or by using the **chartdata.setNotes**, or **chartdata.appendNotes** methods. See the section *Updating Chart Data*. If you click on a “Y” in the Notes row for a sample subgroup, the complete text of the note for that sample subgroup will display in a edit text box, immediately above the “Y”. You can actually edit the notes in the edit text box.

Notes Tooltip



[JavaScript]

```
function SimulateData(spcchart, count, mean, sigma) {
    // batch number for a given sample subgroup
    var batchCounter = 0;
    var i=0;
    var timestamp = new Date();
    if (!spcchart) return;
    if (!spcchart.getChartData()) return;

    var currentcount = spcchart.getChartData().getCurrentNumberRecords();
    for (i = 0; i < count; i++) {
        // Simulate a sample subgroup record
        var samples =
spcchart.getChartData().simulateMeasurementRecordMeanRange(mean, sigma);
        // Update chart data using i as the batch number
        batchCounter = currentcount + i;
        var note = "";
        if ((i % 5) == 0) note = "This is a note";
        // Add a new sample record to the chart data

spcchart.getChartData().addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter
, timestamp, samples, note);
        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }

}
```

[TypeScript]

```
SimulateData(spcchart: QCSPCChartTS.SPCChartBase, count: number, mean: number,
sigma: number)  {
    // batch number for a given sample subgroup
    let batchCounter: number = 0;
        let i: number = 0;
        let timestamp: Date = new Date();
        let chartdata: QCSPCChartTS.SPCControlChartData | null =
spcchart.getChartData();
        if (!chartdata) return;
        let currentcount = chartdata.getCurrentNumberRecords();

    for (i = 0; i < count; i++) {
        // Simulate a sample subgroup record
        let samples: QCSPCChartTS.DoubleArray =
chartdata.simulateMeasurementRecordMeanRange(mean, sigma);
        // Update chart data using i as the batch number
        batchCounter = currentcount + i;
        let note: string = "";
        if ((i % 5) == 0) note = "This is a note";
        // Add a new sample record to the chart data
        chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, note);
        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }

}
```

Both kinds of tooltips are on by default. Turn the tooltips on or off in the program using the **EnableDataToolTip** and **EnableNotesToolTip** flags.

[JavaScript / TypeScript]

```
// Enable data and notes tooltips
xbarrchart.setEnableDataToolTip(true);
xbarrchart.setEnableNotesToolTip(true);
```

The notes tooltip has an additional option. In order to make the notes tooltip “editable”, the tooltip, which is a simple edit box, displays on the first click, and goes away on the second click. You can click inside the edit box and not worry the tooltip suddenly disappearing. The notes tooltip works this way by default. If you wish to explicitly set it, or change it so that the tooltip only displays while the mouse button is held down, as the data tooltips do, set the **chartdata.NotesToolTips.ToolTipMode** property to **NotesToolTip.MOUSEDOWN_TOOLTIP**, as in the example below.

[JavaScript]

```
// Enable data and notes tooltips
xbarrchart.setEnableDataToolTip(true);
xbarrchart.setEnableNotesToolTip(true);

var chartdata = xbarrchart.getChartData();
if (chartdata) {
   var tooltip = chartdata.getNotesTooltips();
   if (tooltip)
   {
      tooltip.setButtonMask( QCSPCChartTS.ChartConstants.RIGHT_BUTTON);
     // default is MOUSETOGGLE_TOOLTIP
```

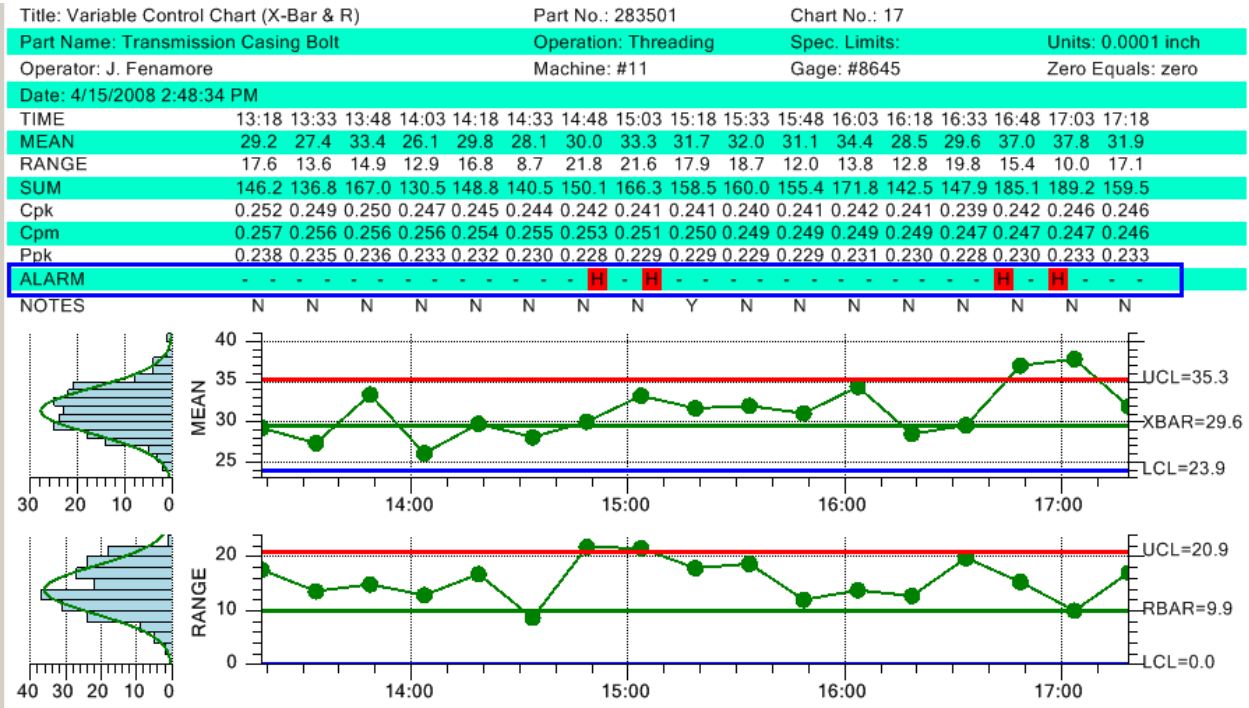
```
        tooltip.setToolTipMode(QCSPCChartTS.NotesToolTip.MOUSEDOWN_TOOLTIP);
    }
}
```

[TypeScript]

```
// Enable data and notes tooltips
xbarrchart.setEnableDataToolTip(true);
xbarrchart.setEnableNotesToolTip(true);

let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarrchart.getChartData();
if (chartdata) {
   let tooltip: QCSPCChartTS.SPCDataTooltip | null = chartdata.getNotesTooltips();
   if (tooltip)
   {
      tooltip.setButtonMask( QCSPCChartTS.ChartConstants.RIGHT_BUTTON);
      // default is MOUSETOGGLE_TOOLTIP
       tooltip.setToolTipMode(QCSPCChartTS.NotesToolTip.MOUSEDOWN_TOOLTIP);
   }
}
```

Enable Alarm Highlighting

EnableAlarmStatusValues



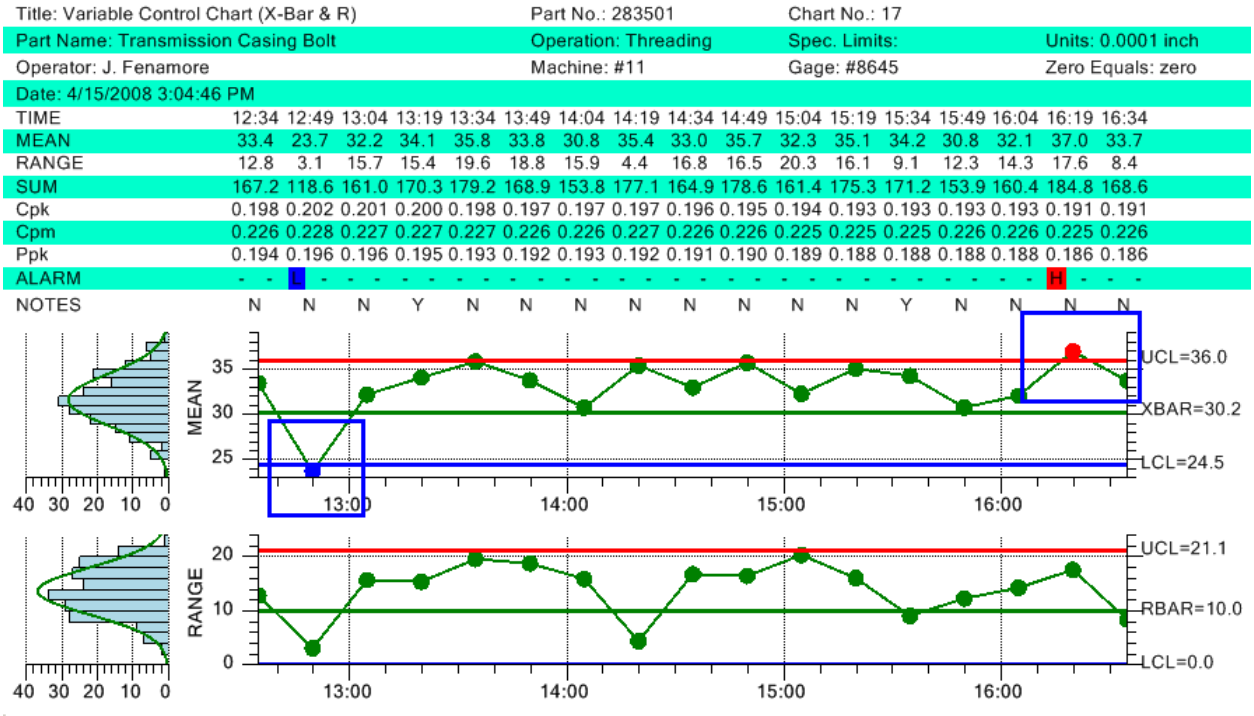
There are several alarm highlighting options you can turn on and off. The alarm status line above is turned on/off using the EnableAlarmStatusValues property. We have set it on by default, so you will have to turn it off if you don't want it. Each sample interval has

two small boxes that are labeled using one of three different characters. An “H” signifies a high alarm, a “L” signifies a low alarm, and a “-“ signifies that there is no alarm.

[JavaScript / TypeScript]

```
// Alarm status line
xbarrchart.setEnableAlarmStatusValues(false);
```

ChartAlarmEmphasisMode



[JavaScript]

```
// Chart alarm emphasis mode
xbarrchart.setChartAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_SYM
BOL);
```

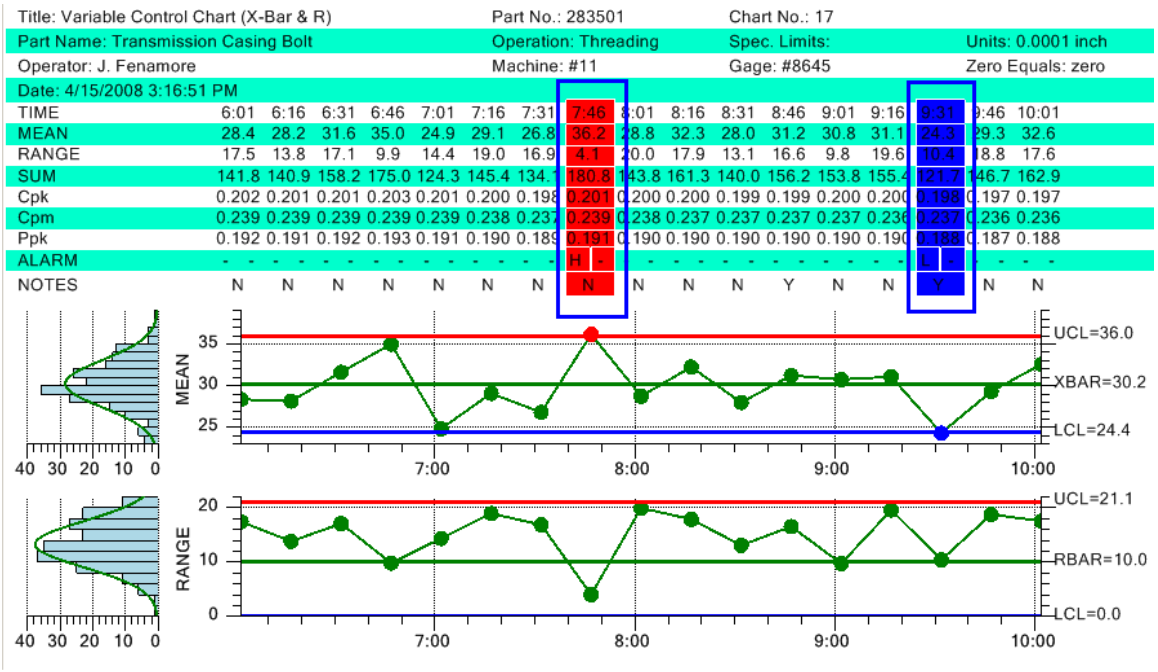
[TypeScript]

```
// Chart alarm emphasis mode
xbarrchart.setChartAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_SYM
BOL);
```

The scatter plot symbol used to plot a data point in the primary and secondary charts is normally a fixed color circle. If you turn on the alarm highlighting for chart symbols the symbol color for a sample interval that is in an alarm condition will change to reflect the color of the associated alarm line. In the example above, a low alarm (blue circle) occurs at the beginning of the chart and a high alarm (red circle) occurs at the end of the chart.

Alarm symbol highlighting is turned on by default. To turn it off use the SPCChartBase.ALARM_NO_HIGHLIGHT_SYMBOL constants.

TableAlarmEmphasisMode -



[JavaScript / TypeScript]

```
// Table alarm emphasis mode
xbarrchart.setTableAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_BAR
);
```

The entire column of the data table can be highlighted when an alarm occurs. There are four modes associated with this property:

ALARM_HIGHLIGHT_NONE	No alarm highlight
ALARM_HIGHLIGHT_TEXT	Text alarm highlight
ALARM_HIGHLIGHT_OUTLINE	Outline alarm highlight
ALARM_HIGHLIGHT_BAR	Bar alarm highlight

The example above uses the ALARM_HIGHLIGHT_BAR mode.

Title: Variable Control Chart (X-Bar & R)							Part No.: 283501				Chart No.: 17						
Part Name: Transmission Casing Bolt							Operation: Threading				Spec. Limits:					Units: 0.0001 inch	
Operator: J. Fenamore							Machine: #11				Gage: #8645					Zero Equals: zero	
Date: 4/15/2008 4:08:49 PM																	
TIME	6:23	6:38	6:53	7:08	7:23	7:38	7:53	8:08	8:23	8:38	8:53	9:08	9:23	9:38	9:53	10:08	10:23
MEAN	32.0	28.2	32.5	23.2	26.5	30.2	26.6	28.4	36.5	28.7	27.7	28.8	29.3	30.0	35.0	27.3	30.0
RANGE	16.7	17.6	16.7	12.3	15.0	14.7	17.8	16.9	15.7	15.9	21.1	9.8	19.3	19.0	11.7	14.5	17.7
SUM	160.2	141.0	162.5	116.1	132.3	151.1	132.9	142.0	182.6	143.3	138.6	143.8	146.4	150.0	175.2	136.5	150.0
Cpk	0.173	0.172	0.173	0.170	0.169	0.169	0.167	0.167	0.169	0.168	0.167	0.167	0.166	0.166	0.168	0.167	0.166
Cpm	0.229	0.228	0.228	0.228	0.227	0.227	0.227	0.226	0.226	0.226	0.225	0.225	0.225	0.224	0.225	0.225	0.224
Ppk	0.168	0.167	0.168	0.165	0.164	0.164	0.162	0.162	0.163	0.163	0.162	0.162	0.161	0.161	0.163	0.162	0.161
ALARM	-	-	-	-	-	-	L	-	-	-	-	-	-	-	H	-	-	-	-	-	-	-	-	-	-	-	-	-	-
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

The example above uses the ALARM_HIGHLIGHT_TEXT mode.

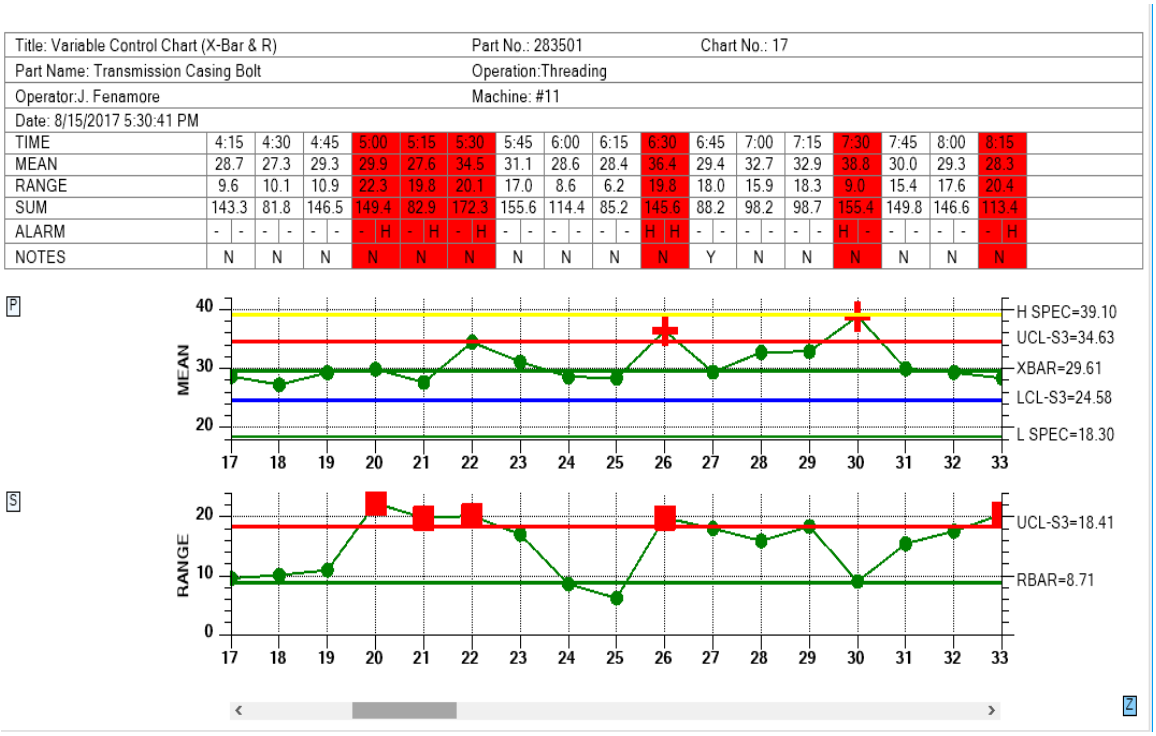
Title: Variable Control Chart (X-Bar & R)							Part No.: 283501				Chart No.: 17						
Part Name: Transmission Casing Bolt							Operation: Threading				Spec. Limits:					Units: 0.0001 inch	
Operator: J. Fenamore							Machine: #11				Gage: #8645					Zero Equals: zero	
Date: 4/15/2008 4:11:27 PM																	
TIME	12:41	12:56	13:11	13:26	13:41	13:56	14:11	14:26	14:41	14:56	15:11	15:26	15:41	15:56	16:11	16:26	16:41
MEAN	24.3	29.8	29.5	33.1	30.4	28.8	37.4	25.5	29.2	24.6	26.2	29.5	31.1	28.6	31.1	27.6	34.7
RANGE	9.2	19.1	17.4	12.7	12.6	12.0	10.5	20.0	16.7	16.4	16.4	13.2	16.9	16.2	12.1	19.3	8.1
SUM	121.6	149.1	147.5	165.6	152.1	143.8	187.1	127.6	145.8	123.2	131.1	147.5	155.3	142.9	155.5	138.1	173.4
Cpk	0.188	0.188	0.187	0.188	0.188	0.188	0.190	0.189	0.188	0.186	0.185	0.184	0.184	0.184	0.184	0.183	0.185
Cpm	0.226	0.226	0.225	0.225	0.225	0.226	0.226	0.225	0.225	0.225	0.224	0.224	0.224	0.224	0.224	0.223	0.224
Ppk	0.184	0.183	0.183	0.184	0.184	0.184	0.186	0.184	0.183	0.181	0.180	0.180	0.180	0.179	0.179	0.178	0.180
ALARM	L	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

The example above uses the ALARM_HIGHLIGHT_OUTLINE mode. In the table above, the column outlines in blue and red reflect what is actually displayed in the chart, whereas in the other TableAlarmEmphasisMode examples the outline just shows where the alarm highlighting occurs.

The default mode is ALARM_HIGHLIGHT_NONE mode.

Enhanced Out of Limit Symbol

One of the SPC chart options is to mark a data point which is outside of control limits by changing the color of the symbol. It is now possible to also also change the size and symbol type for out of limit symbols.



A data point which is outside of control limits can be automatically marked using color, symbol type, and color.

In the example above, the out of control symbol for the Primary chart is an extra large plus sign, while for the the Secondary chart it is an extra large square, both contrasting with the default circle symbol.

The default symbol for the out of control indication is the same as the in control indication, a filled circle. Only a color change signifies out of control. To change the notification symbol, and size, use the OutOfLimitSymbolNumber and OutOfLimitSymbolSize properties, which apply separately to the Primary and Secondary charts.

setOutOfLimitSymbolNumber

Set the symbol type for data points found to be in alarm. Use one of the symbol type constants found in the ChartConstants class: SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE. :

setOutOfLimitSymbolSize

Set the size in pixels of the out of limit symbol:

For example:

[JavaScript]

```
var primarychart = xbarsigmachart.getPrimaryChart();
var secondarychart = xbarsigmachart.getSecondaryChart();
if (primarychart)
{
// Set symbol emphasis type, and size, for primary chart
    primarychart.setOutOfLimitSymbolNumber( QCSPCChartTS.ChartConstants.PLUS);
    primarychart.setOutOfLimitSymbolSize(16);
}
if (secondarychart)
{
// Set symbol emphasis type, and size, for secondary chart
    secondarychart.setOutOfLimitSymbolNumber(QCSPCChartTS.ChartConstants.SQUARE);
    secondarychart.setOutOfLimitSymbolSize(14);
}
```

[TypeScript]

```
let primarychart: QCSPCChartTS.SPCChartObjects | null =
xbarsigmachart.getPrimaryChart();
let secondarychart: QCSPCChartTS.SPCChartObjects | null =
xbarsigmachart.getSecondaryChart();
if (primarychart)
{
// Set symbol emphasis type, and size, for primary chart
    primarychart.setOutOfLimitSymbolNumber( QCSPCChartTS.ChartConstants.PLUS);
    primarychart.setOutOfLimitSymbolSize(16);
}
if (secondarychart)
{
// Set symbol emphasis type, and size, for secondary chart
    secondarychart.setOutOfLimitSymbolNumber( QCSPCChartTS.ChartConstants.SQUARE);
    secondarychart.setOutOfLimitSymbolSize(14);
}
```

See VariableControlCharts.BuildXBarSigmaChart for an example.

AutoLogAlarmsAsNotes

When an alarm occurs, details of the alarm can be automatically logged as a Notes record. Just set the AutoLogAlarmsAsNotes property to true.

[JavaScript / TypeScript]

```
xbarrchart.setAutoLogAlarmsAsNotes( true);
```

Changing the Batch and Event Control Chart X-Axis Labeling Mode

You may find that labeling every subgroup tick mark with a time stamp, or a user-defined string, causes the axis labels to stagger because there is not enough room to display the tick mark label without overlapping its neighbor. In these cases you may wish to reduce the number of sample subgroups you show on the page using the *numdatapointsinview* variable found in all of the example programs.

```
// Number of datapoints in the view
 numdatapointsinview = 13;
```

You can rotate the x-axis labels using the charts XAxisLabelRotation property.

[JavaScript / TypeScript]

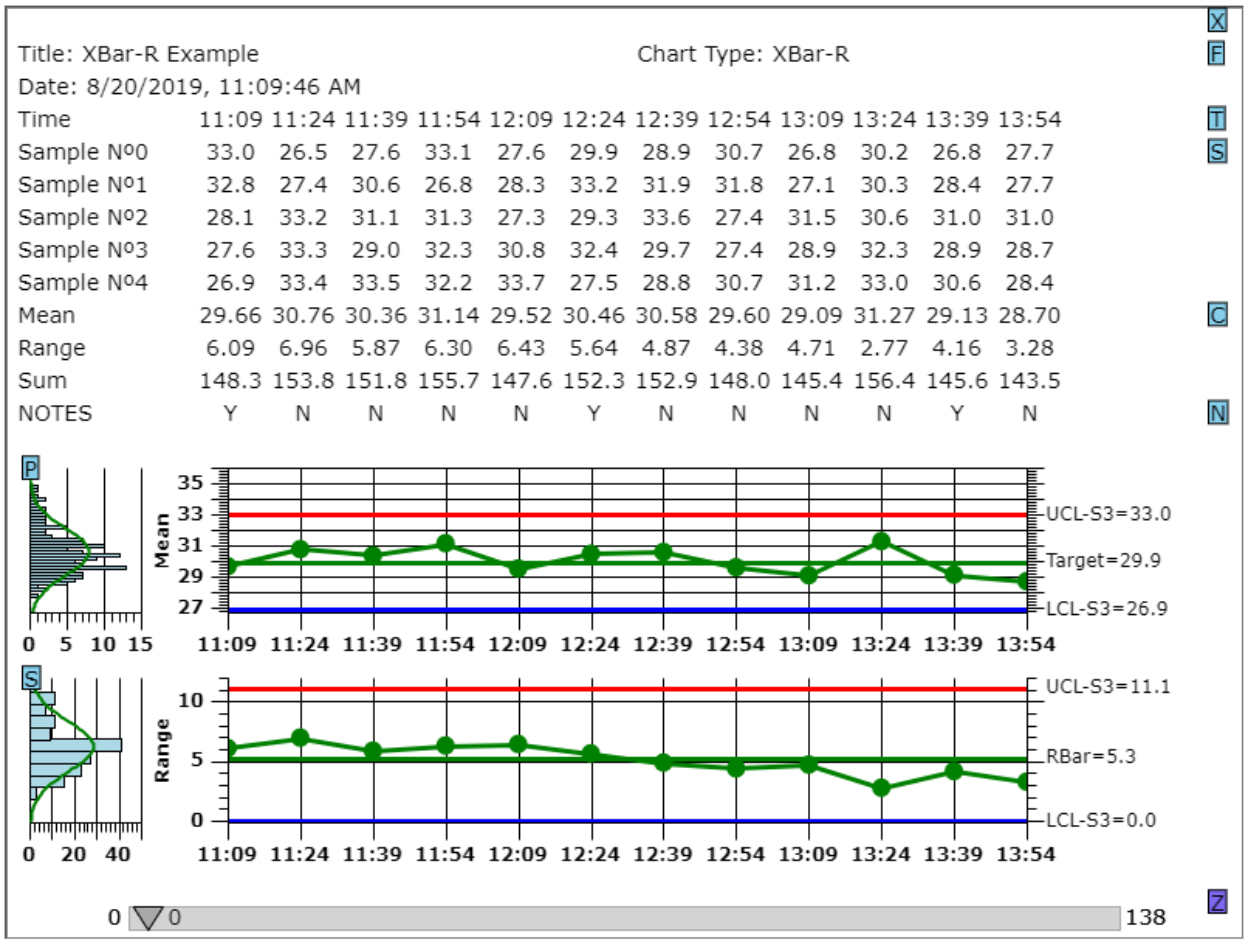
```
xbarrchart.setXAxisLabelRotation(90);
```

If you rotate the x-axis labels you may need to leave more room between the primary and secondary graphs, and at the bottom, to allow for the increased height of the labels.

[JavaScript / TypeScript]

```
xbarrchart.setXAxisLabelRotation(90);
xbarrchart.setInterGraphMargin(0.1);
xbarrchart.setGraphBottomPos(0.85);
```

Batch Control Chart X-Axis Time Stamp Labeling



Batch X-Bar R Chart using time stamp labeling of the x-axis

Set the x-axis labeling mode using the overall charts XAxisStringLabelMode property, setting it `SPCChartObjects.AXIS_LABEL_MODE_TIME.`

[JavaScript / TypeScript]

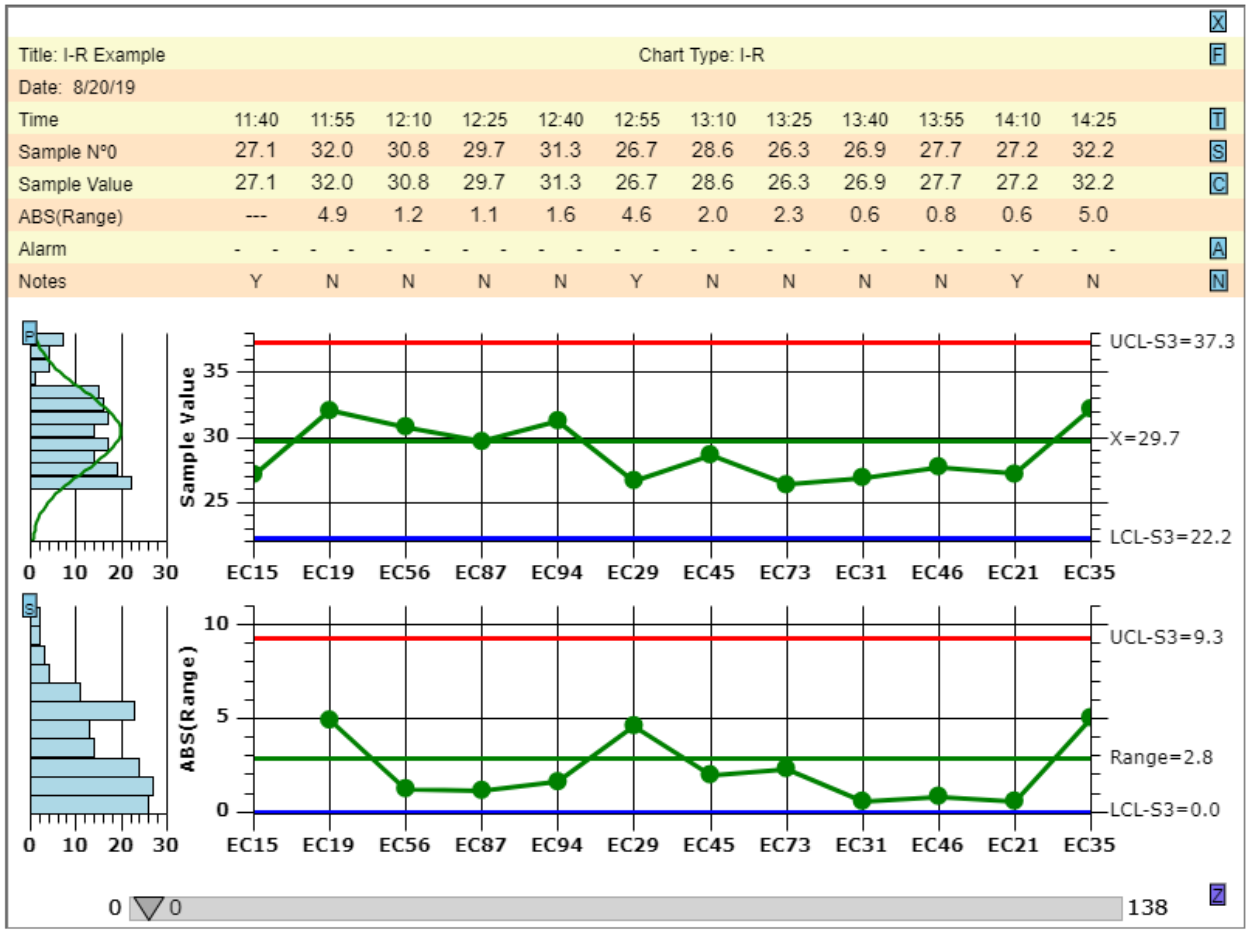
```
xbarrchart.setEnableScrollBar( true);
xbarrchart.setEnableCategoryValues(false);

// Label the tick mark with time stamp of sample group
xbarrchart.setXAxisStringLabelMode(QCSPCChartTS.SPCChartObjects.AXIS_LABEL_MODE_TI
ME);
```

When updating the chart with sample data, use addNewSampleRecord overload that has batch number and a time stamp parameters.
[JavaScript / TypeScript]

```
chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter, timestamp,
samples, note);
```

See the example program VariableControlCharts.BuildXBarRChart for a complete example. Reset the axis labeling mode back to batch number labeling by assigning the XAxisStringLabelMode property to SPCChartObjects.AXIS_LABEL_MODE_DEFAULT.

Batch Control Chart X-Axis User-Defined String Labeling



Batch X-Bar R Chart user-defined string labeling of the x-axis

Set the x-axis labeling mode using the overall charts XAxisStringLabelMode property, setting it `QCSPCChartTS.SPCChartObjects.AXIS_LABEL_MODE_STRING`.

[JavaScript / TypeScript]

```
// Label the tick mark with user-defined strings
```

```
irchart.setXAxisStringLabelMode(QCSPCChartTS.SPCChartObjects.AXIS_LABEL_MODE_STRIN
G);
```

Use the addAxisUserDefinedString method to supply a new string for every new sample subgroup. It must be called every time the addNewSampleRecord method is called, or the user-defined strings will get out of sync with their respective sample subgroup. Reset the axis labeling mode back to batch number labeling by assigning the XAxisStringLabelMode property to SPCChartObjects.AXIS_LABEL_MODE_DEFAULT.

[JavaScript]

```
chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter, timestamp,
samples, note);

var randomnum=  Math.round(100 * QCSPCChartTS.ChartSupport.getRandomDouble());
var batchidstring = "EC" + randomnum.toString();

chartdata.addAxisUserDefinedString(batchidstring);
```

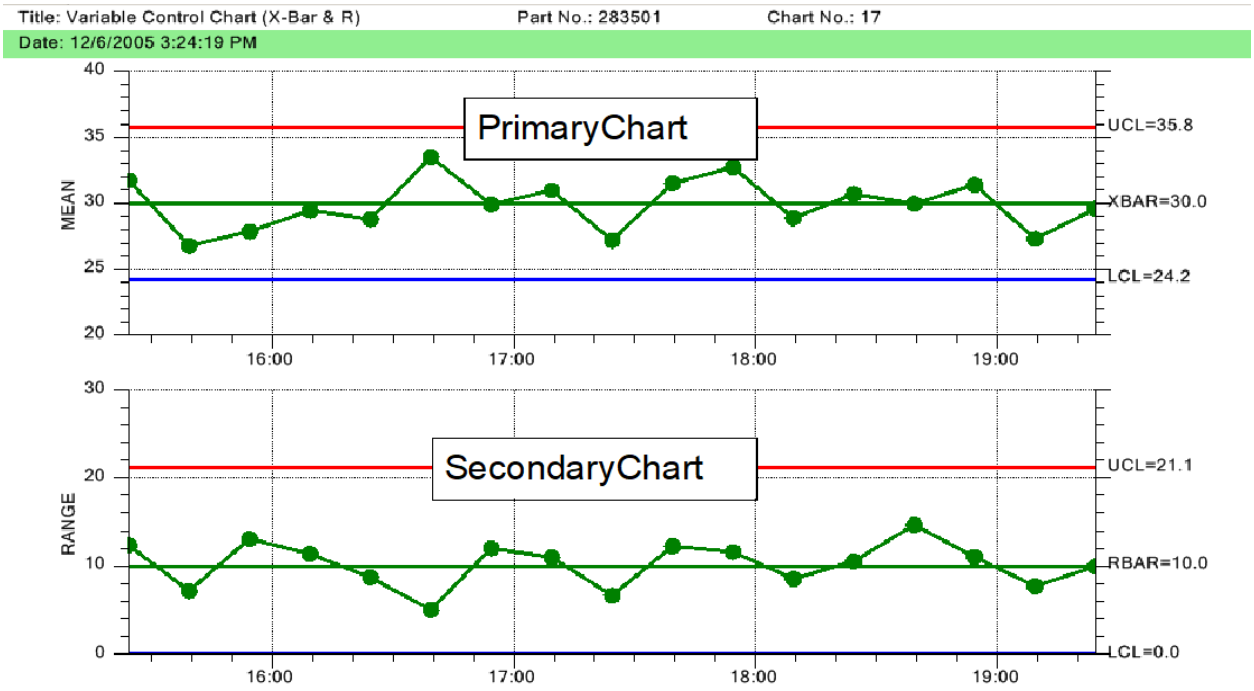
[TypeScript]

```
chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter, timestamp,
samples, note);

let randomnum: number=  Math.round(100 *
QCSPCChartTS.ChartSupport.getRandomDouble());
let batchidstring: string = "EC" + randomnum.toString();

chartdata.addAxisUserDefinedString(batchidstring);
```

See the example program MultiLimitCharts.SimulateVariableControlChartData for a complete example.

Changing Default Characteristics of the Chart

All *Variable Control Charts* have two distinct graphs, each with its own set of properties. The top graph is the Primary Chart, and the bottom graph is the Secondary Chart.



Logically enough, the properties of the objects that make up each of these graphs are stored in properties named **PrimaryChart** and **SecondaryChart**. Once the spc graph is instantiated (using one of the **newSPCBatchVariableControlChart**..., or **newSPCEventVariableControlChart...** constructors) , you can modify the default characteristics of each graph using these properties.

The primary chart, and secondary chart references are assigned to local variables (primarychart and secondarychart) in order to accommodate TypeScript strict null checking. In JavaScript the code can be more relaxed.

[JavaScript]

```
   var xbarrchart: QCSPC =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);

     var primarychart = xbarrchart.getPrimaryChart();
     if (primarychart)
     {
         primarychart.setPlotMeasurementValues(true);
         primarychart.getXAxis().setLineColor( QCSPCChartTS.ChartColor.BLUE);
         primarychart.getXAxis().setLineWidth(3);
     }
     var secondarychart = xbarrchart.getSecondaryChart();
     if (secondarychart)
     {
        secondarychart.getYAxis1().setLineColor( QCSPCChartTS.ChartColor.GREEN);
        secondarychart.getYAxis2().setLineColor(QCSPCChartTS.ChartColor.RED);
        secondarychart.getYAxis1().setLineWidth( 3);
        secondarychart.getYAxis2().setLineWidth(3);
     }
```

[TypeScript]

```
   let xbarrchart: QCSPCChartTS.SPCBatchVariableControlChart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);


     let primarychart: QCSPCChartTS.SPCChartObjects | null =
xbarrchart.getPrimaryChart();
     if (primarychart)
     {
         primarychart.setPlotMeasurementValues(true);
         primarychart.getXAxis().setLineColor( QCSPCChartTS.ChartColor.BLUE);
         primarychart.getXAxis().setLineWidth(3);
     }
     let secondarychart: QCSPCChartTS.SPCChartObjects | null =
xbarrchart.getSecondaryChart();
     if (secondarychart)
     {
        secondarychart.getYAxis1().setLineColor( QCSPCChartTS.ChartColor.GREEN);
        secondarychart.getYAxis2().setLineColor(QCSPCChartTS.ChartColor.RED);
        secondarychart.getYAxis1().setLineWidth( 3);
        secondarychart.getYAxis2().setLineWidth(3);
     }
```

The **PrimaryChart** and **SecondaryChart** objects are both instances of the **SPCChartObjects** class. The **SPCChartObjects** class contains the objects needed to display a single graph. Below you will find a summary of the class properties.

Note - Since JavaScript/TypeScript does not support properties in the same way that C# does, you set and get the values of the properties using the “set” and “get” in front of the property name, i.e. **getAnnotationFont** and **setAnnotationFont.**

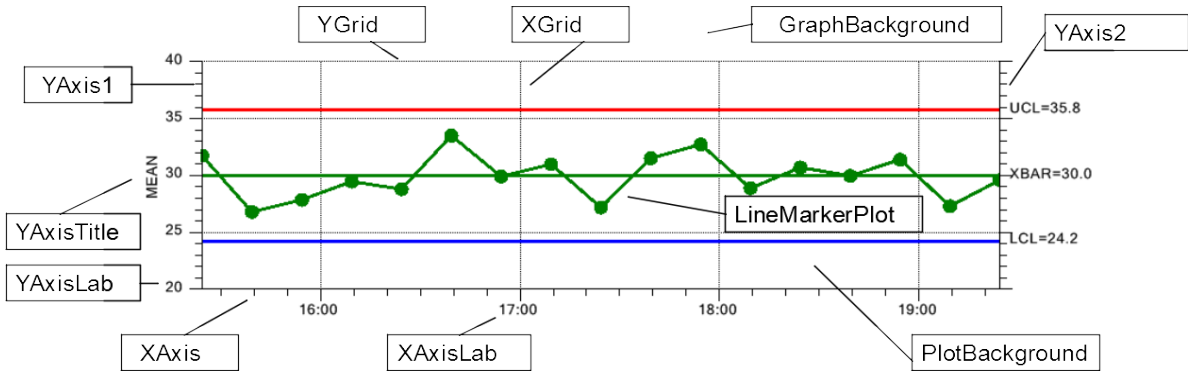
Public Instance Properties

AnnotationArray	set/get the array of TextObject objects, representing the annotations of the chart.
AnnotationFont	set/get annotation font.
AnnotationNudge	set/get the x and y-values use to offset a data points annotation with respect to the actual data point.
AxisLabelFont	set/get the font used to label the x- and y- axes.
AxisTitleFont	set/get the font used for the axes titles.
ControlLabelPosition	set/get that numeric label for a control limit is placed inside, or outside the plot area INSIDE_PLOTAREA.
ControlLimitData	set/get the array of the plot objects associated with control limits.
Datatooltip	set/get a reference to the charts tooltip.
DefaultChartBackgroundColor	set/get the default background color for the graph area.
DefaultNumberControlLimits	set/get the number of control limits in the chart.
DefaultPlotBackgroundColor	set/get the default background color for the plot area.

DisplayChart	set/get to true to enable the drawing of this chart.
DisplayFrequencyHistogram	set/get to true to enable the drawing of the frequency histogram attached to the chart.
FrequencyHistogramChart	set/get a reference to the optional frequency histogram attached to the chart.
GraphBackground	set/get a reference to the charts graph background object.
BatchIncrement	set/get increment between adjacent samples of Batch type charts that use a numeric x-scale.
BatchStartValue	set/get the starting numeric value of the x-scale for Batch type charts that use a numeric x-scale.
BatchStopValue	set/get the ending numeric value of the x-scale for Batch type charts that use a numeric x-scale.
Header	set/get a reference to the charts header.
HeaderFont	set/get the font used for the chart title.
HistogramStartPos	set/get the left edge, using normalized coordinates, of the frequency histogram plotting area.
HistogramOffset	set/get the offset with respect to the GraphStartPosX value, using normalized coordinates, of the frequency histogram plotting area.
MaxY	set/get the maximum value used to scale the y-axis of the chart.
MinY	set/get the minimum value used to scale the y-axis of the chart.
ParentSPCChartBase	set/get that parent SPCChartBase object.
PlotBackground	set/get a reference to the charts plot background object.
PlotMeasurementValues	set/get to true to enable the plotting of all sampled values, as a scatter plot, in addition to the mean or median values.
PPhysTransform1	set/get a reference to the charts physical coordinate system.
ProcessVariableData	Holds a reference to an object encapsulating the plot object data associated with the main variable of the chart.
SampledDataData	set/get the array of the sample data.
SubHead	set/get a reference to the charts subhead.
SubheadFont	set/get the font used for the chart subhead.
TableFont	set/get the font used for the data table.
TextTemplate	set/get the text template for the data tooltip.
TimeIncrementMinutes	set/get the increment between adjacent samples

	of charts that use a numeric x-scale.
ToolTipFont	set/get tooltip font.
ToolTipSymbol	set/get a reference to the charts tooltip symbol.
XAxis	set/get a reference to the charts x-axis.
XAxisLab	set/get a reference to the charts x-axis labels.
XGrid	set/get a reference to the charts x-axis grid.
XValueTemplate	set/get the x-value template for the data tooltip.
YAxis1	set/get a reference to the charts left y-axis.
YAxis2	set/get a reference to the charts right y-axis.
YAxisLab	set/get a reference to the charts left y-axis labels.
YAxisTitle	set/get a reference to the charts left y-axis title.
YGrid	set/get a reference to the charts y-axis grid.
YValueTemplate	set/get the y-value template for the data tooltip.

The main objects of the graph are labeled in the graph below.



Chapter 7 - SPC Attribute Control Charts

SPCEventAttributeControlChart
SPCTimeAttributeControlChart
SPCBatchAttributeControlChart

Attribute Control Charts are a set of control charts specifically designed for tracking product defects (also called non-conformities). These types of defects are binary in nature (yes/no), where a part has one or more defects, or it doesn't. Examples of defects are paint scratches, discolorations, breaks in the weave of a textile, dents, cuts, etc. Think of the last car that you bought. The defects in each sample group are counted and run through some statistical calculations. Depending on the type of *Attribute Control Chart*, the number of defective parts are tracked (p-chart and np-chart), or alternatively, the number of defects are tracked (u-chart, c-chart). The difference in terminology "number of defective parts" and "number of defects" is highly significant, since a single part not only can have multiple defect categories (scratch, color, dent, etc), it can also have multiple defects per category. A single part may have 0 – N defects. So keeping track of the number of defective parts is statistically different from keeping track of the number of defects. This affects the way the control limits for each chart are calculated.

p-Chart - Also known as the Percent or Fraction Defective Parts Chart
For a sample subgroup, the number of defective parts is measured and plotted as either a percentage of the total subgroup sample size, or a fraction of the total subgroup sample size. Since the plotted value is a fraction or percent of the sample subgroup size, the size of the sample group can vary without rendering the chart useless.

np-Chart – Also known as the Number Defective Parts Chart
For a sample subgroup, the number of defective parts is measured and plotted as a simple count. Statistically, in order to compare number of defective parts for one subgroup with the other subgroups, this type of chart requires that the subgroup sample size is fixed across all subgroups.

c-Chart - Also known as the Number of Defects or Number of Non-Conformities Chart
For a sample subgroup, the number of times a defect occurs is measured and plotted as a simple count. Statistically, in order to compare number of defects for one subgroup with the other subgroups, this type of chart requires that the subgroup sample size is fixed across all subgroups.

u-Chart – Also known as the Number of Defects per Unit or Number of Non-Conformities per Unit Chart
For a sample subgroup, the number of times a defect occurs is measured and plotted as either a percentage of the total subgroup sample size, or a fraction of the total subgroup sample size. Since the plotted value is a fraction or percent of the sample subgroup size, the size of the sample group can vary without rendering the chart useless.

DPMO Chart – Also known as the Number of Defects per Million Chart
For a sample subgroup, the number of times a defect occurs is measured and plotted as a value normalized to defects per million. Since the plotted value is normalized to a fixed sample subgroup size, the size of the sample group can vary without rendering the chart useless.

Event-Based, Time-Based and Batch-Based SPC Charts

Attribute Control Charts are further categorized as either event-, time- or batch- based. Time-based SPC charts are used when data is collected using a subgroup interval corresponding to a specific time interval. Batch-based (and Event-based) SPC charts are used when the data subgroup interval is a sequential batch number that does not correspond to a uniform time interval. The major difference in these two types of SPC charts is the display of the x-axis. Variable control charts that sample using a uniform time interval will generally use a time-based x-axis, with time/date axis labels. Variable control charts that sample based on batches will generally use a numeric-based x-axis, with numeric axis labels.

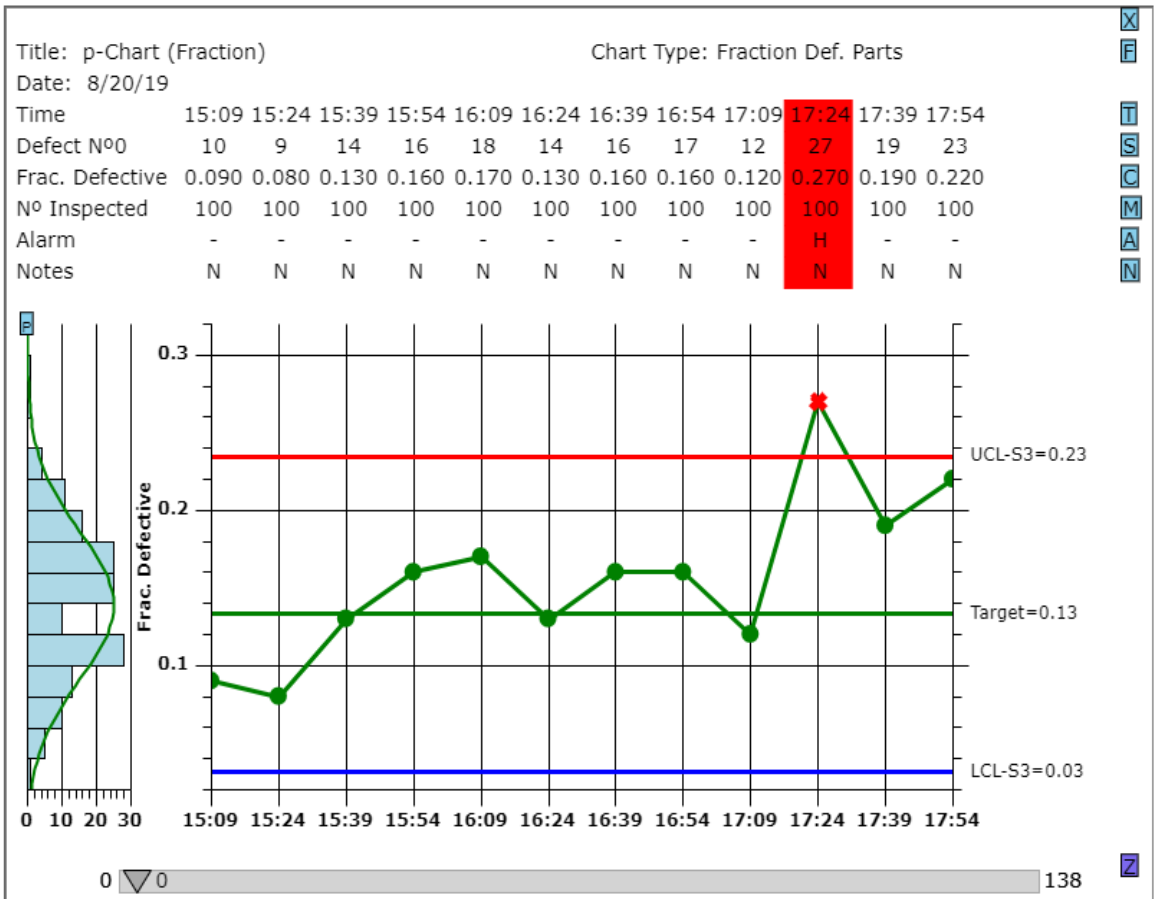
SPECIAL NOTE: The time-based and batch-based SPC charts have been deprecated and replaced by the new event-based charts. In order to maintain backward compatibility, we also keep the old SPCTime... and SPCBatch... control chart classes, but derive them from the new Event-based SPC chart classes. The only difference you might see is an actual benefit. No matter what the time stamp is on a SPCTime... chart, adjacent points will always be equally spaced. So if your sample interval is irregular, or you even skip days or weeks in your sampling, the resulting chart will still display equally spaced adjacent sample records. **Furthermore**, if you are using the SPCTimeAttributeControlChart class, we recommend that you instead use either the SPCBatchAttributeControl (or SPCEventAttributeControlChart class. If you want to see time/date values on the x-axis, set the XAxisStringLabelMode of the chart to AXIS_LABEL_MODE_TIME.

[JavaScript / TypeScript]

```
attribchart.setXAxisStringLabelMode(QCSPCChartTS.SPCChartObjects.AXIS_LABEL_MODE_T
IME)
```

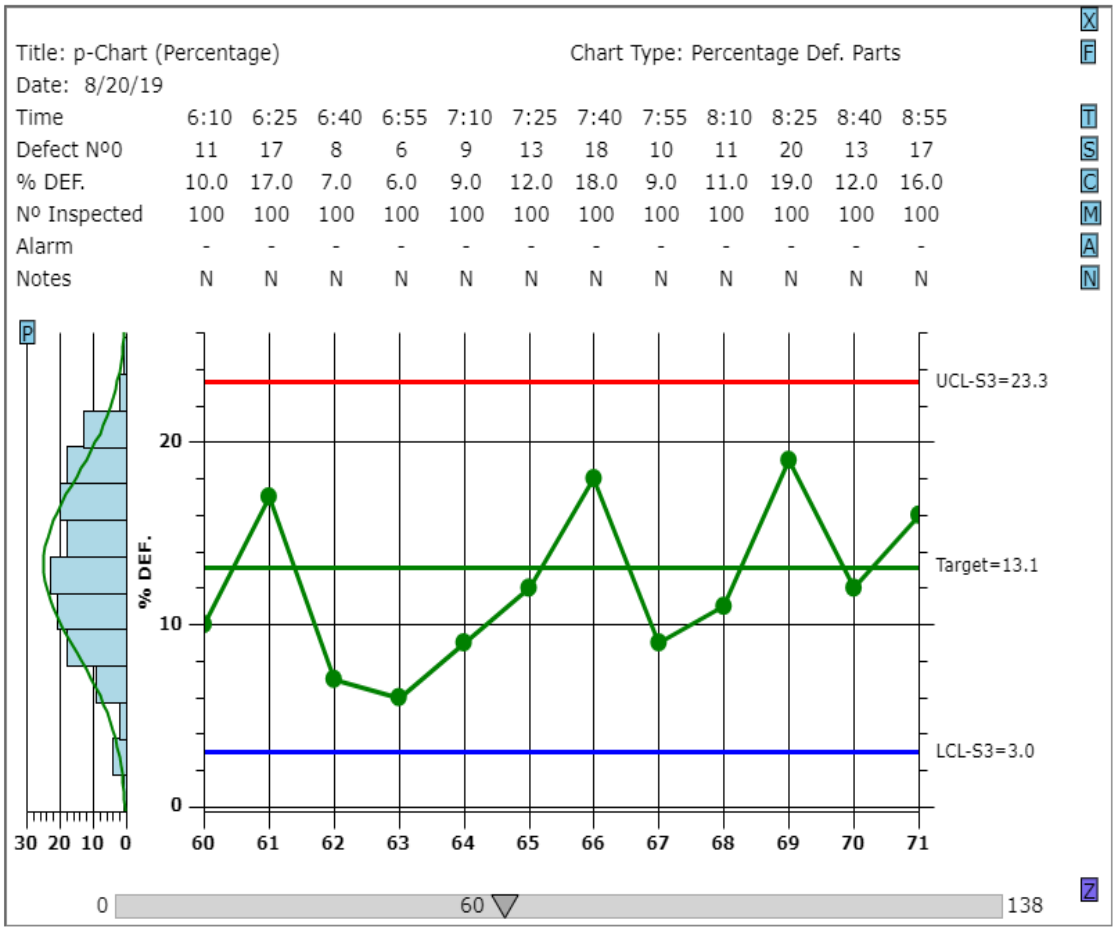
The new class heirarchy looks like:

ChartView
SPCChartBase
SPCEventVariableControlChart
SPCTimeVariableControlChart
SPCBatchVariableControlChart
SPCEventAttributeControlChart
SPCTimeAttributeControlChart
SPCBatchAttributeControlChart



Batch-Based Attribute Control Chart with time labeled x-axis

Note the time-based x-axis. Even though the time stamp values may not have consistent time interval, the data points are spaced evenly by batch number.



Batch-Based Attribute Control Chart using a numeric x-axis

Attribute Control Charts Consist of Only One Graph

Whereas the *Variable Control Charts* contain two different graphs, which we refer to generically as the primary and secondary graphs of the chart, *Attribute Control Charts* only have a single graph, which we refer to generically as the primary graph of the chart.

Creating an Attribute Control Chart

We recommend that you use the **newSPCBatchAttributeControlChart...** static constructor to create a new instance of a **SPCBatchAttributeControlChart**. The example below is extracted from the **AttributeControlCharts.BuildPChartF** example program.

[JavaScript]

```
   export async function BuildPChartF(canvasid) {

    var canvasid = "spcChartCanvas1";
    var htmlcanvas = document.getElementById(canvasid);
    var spccharttype =
QCSPCChartTS.SPCControlChartData.FRACTION_DEFECTIVE_PARTS_CHART;
    var subgroupsize = 100;
    var numberpointsinview = 12;
    var charttitle = " p-Chart (Fraction)";


    var attribchart =
QCSPCChartTS.SPCBatchAttributeControlChart.newSPCBatchAttributeControlChartChartTy
peSubgroupSize(htmlcanvas, spccharttype, 1, subgroupsize, numberpointsinview);

    attribchart.setPreferredSize(800, 600);
attribchart.setXAxisStringLabelMode( QCSPCChartTS.SPCChartObjects.AXIS_LABEL_MODE_
TIME);

.
.
.

    // Calculate the SPC control limits for both graphs of the current SPC chart
(X-Bar R)
    attribchart.autoCalculateControlLimits();

        // Out some more data, different mean and sigma, to simulate out of
control
        fractiondefective = 0.15;
        numssampleintervals = 50;
     SimulateData(attribchart, numssampleintervals, fractiondefective,
subgroupsize);

    // Scale the y-axis of the X-Bar chart to display all data and control limits
    attribchart.autoScalePrimaryChartYRange();
    // Scale the y-axis of the Range chart to display all data and control limits
    attribchart.autoScaleSecondaryChartYRange();
    // Rebuild the chart using the current data and settings
    attribchart.rebuildChartUsingCurrentData();


}
```

[TypeScript]

```
 public async  BuildPChartF(canvasid: string) {

   let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
   let spccharttype: number =
QCSPCChartTS.SPCControlChartData.FRACTION_DEFECTIVE_PARTS_CHART;
   let subgroupsize: number = 100;
   let numberpointsinview: number = 12;
   let charttitle: string = " p-Chart (Fraction)";


   let attribchart: QCSPCChartTS.SPCChartBase =
QCSPCChartTS.SPCBatchAttributeControlChart.newSPCBatchAttributeControlChartChartTy
peSubgroupSize(htmlcanvas, spccharttype, 1, subgroupsize, numberpointsinview);
   if (!attribchart) return;
```

```
   attribchart.setPreferredSize(800, 600);


attribchart.setXAxisStringLabelMode( QCSPCChartTS.SPCChartObjects.AXIS_LABEL_MODE_
TIME);

   attribchart.setGraphStopPosX(0.825);
   attribchart.setGraphStartPosX(0.18);
.
.
.

   // Scale the y-axis of the X-Bar chart to display all data and control limits
   attribchart.autoScalePrimaryChartYRange();
   // Scale the y-axis of the Range chart to display all data and control limits
   attribchart.autoScaleSecondaryChartYRange();
   // Rebuild the chart using the current data and settings
   attribchart.rebuildChartUsingCurrentData();
}
```

SPCBatchAttributeControlChart Members

SPCBatchAttributeControlChart overview

Static Instance Constructors

newSPCBatchAttributeControlChartChartTypeSubgroupSize	Creates a new instance of the SPCBatchAttributeControlChart class and initializes it with the supplied parameters.

Public Instance Constructors

SPCBatchAttributeControlChart	Initializes a new instance of the SPCBatchAttributeControlChart class. You will still need to call initSPCBatchVariableControlChart

Public Instance Functions

initSPCBatchAttributeControlChartCanvasChartTypeSubgroupSize	Initialize the class for a specific SPC chart type.

The SPCBatchAttributeControlChart properties are documented in the QCSPCChartJSTSClassesIndex.html documentation file, located in the docs/docs/ subdirectory.

The control chart type (p-, np-, c- and u-charts) is established when the chart is created, either in one of the newSPCEventAttribute... constructors, or one of the initSPCEventAttributeControlChart... initialization routines.

SPCEventAttributeControlChart.initSPCEventAttributeControlChart Method

This initialization method initializes the most important values in the creation of a SPC chart.

[TypeScript]

```
public static newSPCBatchAttributeControlChartChartTypeSubgroupSize(context: Canvas,
charttype: number, numcategories: number, numsamplespersubgroup: number,
numdatapointsinview: number): SPCBatchAttributeControlChart

public initSPCBatchAttributeControlChartCanvasChartTypeSubgroupSize(context:
Canvas, charttype: number, numcategories: number, numsamplespersubgroup: number,
numdatapointsinview: number)
```

Parameters

canvas

A reference to the the HTML5 canvas object the chart is placed in. Typically the string id of the canvas is passed in, and the reference is looked up using a call document.getElementById(canvasid), as done in each of the example programs.

charttype

Specifies the chart type. Use one of the SPC Attribute Control chart types: PERCENT_DEFECTIVE_PARTS_CHART, FRACTION_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTS_PERUNIT_CHART, NUMBER_DEFECTS_CHART, NUMBER_DEFECTS_PER_MILLION_CHART.

numcategories

In Attribute Control Charts this value represents the number of defect categories used to determine defect counts.

numsamplespersubgroup

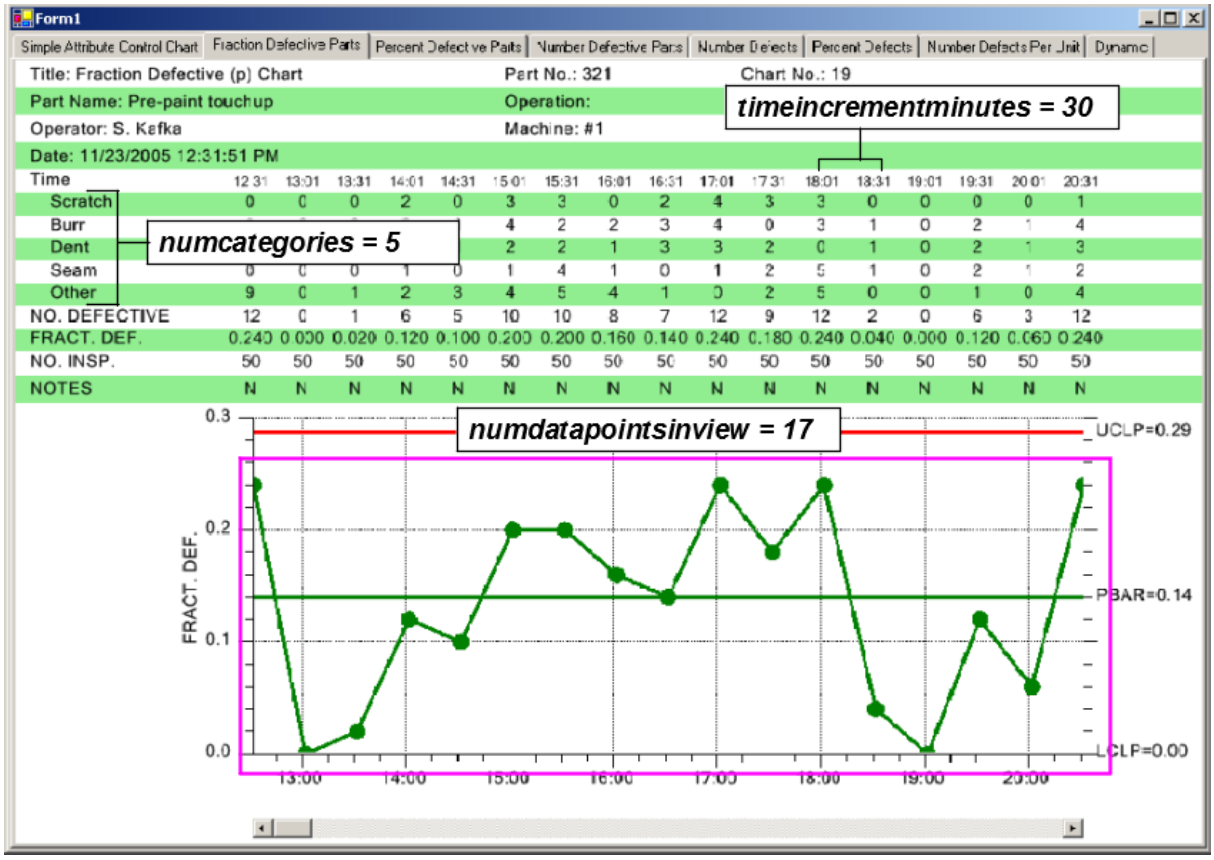
In an Attribute Control chart it represents the total sample size per sample subgroup from which the defect data is counted.

numdatapointsinview

Specifies the number of sample subgroups displayed in the graph at one time.

Note: The *timeincrementminutes* was found in the original intSPCTimeVariableControlChart method, but is not present in the initSPCBatchVariableControlChart method

The image below further clarifies how these parameters affect the attribute control chart.



Once the spc chart has been instantiated, the chart can be further customized using properties inherited from **SPCBaseChart**, described below.

Note - Since JavaScript/TypeScript does not support properties in the same way that C# does, you set and get the values of the properties using the "set" and "get" in front of the property name, i.e. **getBottomLableMargin** and **setBottomLableMargin.**

Public Static Properties

DefaultChartFontString	set/get the default font used in the table display.

Public Instance Properties

AutoLogAlarmsAsNotes	Set to true to automatically log alarm details in the

	sample interval Notes record.
BottomLabelMargin	set/get an additional margin, in normalized coordinates, if only the primary graphs is displayed, allowing for the x-axis labels
ChartData	set/get the object that holds the descriptive text, sampled and calculated values associated with the control chart.
ChartAlarmEmphasisMode	Set to SPCChartBaseALARM_HIGHLIGHT_SYMBOL to highlight the process variable symbol if an alarm condition exists. Set to Set to SPCChartBase.ALARM_NO_HIGHLIGHT_SYMBOL to turn off alarm highlighting.
ChartTable	set/get the object that holds the data table information needed to display the data table along with the chart
DefaultControlLimitSigma	set/get that SPC control limits are to be calculated using the 3 sigma level standard.
EnableAlarmStatusValues	If set true enables the alarm status row of the chart table.
EnableCalculatedValues	If set true enables the calculated values rows of the data table
EnableCategoryValues	If set true enables the category or sample values rows of the data table
EnableDataToolTip	If set true enables data tooltips
EnableInputStringsDisplay	If set true enables the input string rows of the data table
EnableNotes	If set true enables the notes row of the data table
EnableNotesToolTip	If set true enables data tooltips
EnableScrollBar	If set true the scroll bar is added to the bottom of the chart.
EnableTimeValues	If set true enables the time row of the data table
EnableTotalSamplesValues	If set true enables the total of sampled values row of the data table
GraphBottomPos	set/get the bottom edge, using normalized coordinates, of the plotting area for the secondary chart
GraphStartPosX	set/get the left edge, using normalized coordinates, of the plotting area for both primary and secondary charts
GraphStartPosY1	set/get the top edge, using normalized coordinates, of the plotting area for the primary chart
GraphStartPosY2	set/get the top edge, using normalized coordinates, of

	the plotting area for the secondary chart
GraphStopPosX	set/get the right edge, using normalized coordinates, of the plotting area for both primary and secondary charts
GraphStopPosY1	set/get the bottom edge, using normalized coordinates, of the plotting area for the primary chart
GraphStopPosY2	set/get the bottom edge, using normalized coordinates, of the plotting area for the secondary chart
GraphTopTableOffset	set/get the offset of the top of the primary chart from the bottom of the data table, using normalized coordinates
HeaderStringsLevel	set/get the level of header strings to include in the chart. Use one of the SPCControlChartData header strings constants: HEADER_STRINGS_LEVEL0, HEADER_STRINGS_LEVEL1, HEADER_STRINGS_LEVEL2, or HEADER_STRINGS_LEVEL3
InterGraphMargin	set/get the margin, in normalized coordinates, between the primary and secondary charts
MultipleMouseListener	set/get the MultiMouseListener.
PrimaryChart	set/get the object that holds he the chart objects needed to display the primary chart
ScrollBarBottomPosition	set/get the bottom edge, using normalized coordinates, of the optional scroll bar
ScrollBarPixelHeight	set/get the height of the scrollbar in pixels
SecondaryChart	set/get the object that holds he the chart objects needed to display the secondary chart
SPCChartType	Specifies the control chart type: use one of the SPCControlChartData chart type constants: MEAN_RANGE_CHART, MEDIAN_RANGE_CHART, MEAN_SIGMA_CHART, MEAN_SIGMA_CHART_VSS, INDIVIDUAL_RANGE_CHART, CUSTOM_ATTRIBUTE_CONTROL_CHART, PERCENT_DEFECTIVE_PARTS_CHART, FRACTION_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTIVE_PARTS_CHART, NUMBER_DEFECTS_CHART, NUMBER_DEFECTS_PERUNIT_CHART, NUMBER_DEFECTS_PER_MILLION_CHART.
TableAlarmEmphasisMode	set/get the table alarm highlighting to one of the SPCChartBase table highlight constants:

	ALARM_HIGHLIGHT_NONE, ALARM_HIGHLIGHT_TEXT, ALARM_HIGHLIGHT_OUTLINE, ALARM_HIGHLIGHT_BAR
XScaleMode	set/get whether the x-axis is time based, or numeric based.

Public Instance Functions

addAnnotation	Add a simple annotation to a data point in the specified SPC chart.
autoCalculateControlLimits	Using the current sampled data values, high, target and low control limits are calculated for both primary and secondary charts using an algorithm appropriate to the SPC chart type.
autoCalculatePrimaryControlLimits	Using the current sampled data values, high, target and low control limits are calculated for the primary chart using an algorithm appropriate to the SPC chart type.
autoCalculateSecondaryControlLimits	Using the current sampled data values, high, target and low control limits are calculated for the primary chart using an algorithm appropriate to the SPC chart type.
autoScaleChartYRange	Auto-scale the y-range of the SPC chart so that all of the sampled data and chart control limits are within the bounds of the y-axis.
autoScalePrimaryChartYRange	Auto-scale the y-range of the primary SPC chart so that all of the sampled data and chart control limits are within the bounds of the y-axis.
autoScaleSecondaryChartYRange	Auto-scale the y-range of the SPC chart so that all of the sampled data and chart control limits are within the bounds of the y-axis.
rebuildChartUsingCurrentData	Rebuild the graph taking into account the most recent data values.
resetSPCChartData	Reset the history buffers of all of the SPC data objects.

The **SPCBatchAttributeControlChart** properties are documented in the QCSPCChartJSTSClassesIndex.html documentation file, located in the docs/docs/ subdirectory.

Special Note for DPMO Charts

The NUMBER_DEFECTS_PER_MILLION_CHART has an important parameter you may need to set. DPMO charts use an important parameter known is the *defect opportunites per unit*. The default value for the parameter is 1. So if you are using 1 as the the value of *defect opportunites per unit* in your chart, you don't need to do anything. If your value is greater than 1, you need to specify that using code similar to below.

[JavaScript / TypeScript]

```
chartdata.setDefectOpportunitiesPerUnit(5);
```

Adding New Sample Records for Attribute Control Charts.

Attribute Control Chart Cross Reference

p-chart = FRACTION_DEFECTIVE_PARTS_CHART
or
PERCENT_DEFECTIVE_PARTS_CHART

np-chart = NUMBER_DEFECTIVE_PARTS_CHART

c-chart = NUMBER_DEFECTS_CHART

u-chart = NUMBER_DEFECTS_PERUNIT_CHART

DPMO = NUMBER_DEFECTS_PER_MILLION_CHART

Updating p- and np-charts

In attribute control charts, the meaning of the data in the *samples* array varies, depending on whether the attribute control chart measures the number of defective parts (p-, and np-charts), or the total number of defects (u- and c-charts). The major anomaly is that while the p- and np-charts plot the fraction or number of defective parts, the table portion of the chart can display defect counts for any number of defect categories (i.e. paint scratches, dents, burrs, etc.). It is critical to understand that total number of defects, i.e. the sum of the items in the defect categories for a give sample subgroup, do NOT have to add up to the number of defective parts for the sample subgroup. Every defective part not only can have one or more defects, it can have multiple defects of the same defect category. The total number of defects for a sample subgroup will always be equal to or greater than the

number of defective parts. When using p- and np-charts that display defect category counts as part of the table, where N is the *numcategories* parameter in the **newSPCEventAttributeControlChart...** or **newSPCBatchAttributeControlChart...** initialization call, the first N elements of the *samples* array holds the defect count for each category. The N+1 element of the *samples* array holds the total defective parts count. For example, if you initialized the chart with a *numcategories* parameter to five, signifying that you had five defect categories, you would use a *samples* array sized to six, as in the code below:

[JavaScript]

```
var samples = QCSPCChartTS.DoubleArray.newDoubleArrayN(6);
//  Date initialized with current  time by default
var timestamp= new Date();  // use a unique time stamp for every update
batchCounter++; / defined elsewhere but incremented with every update
// Place sample values in array
samples.setElement(0, 3);      // Number of defects for defect category #1
samples.setElement(1, 0);      // Number of defects for defect category #2
samples.setElement(2, 4);      // Number of defects for defect category #3
samples.setElement(3, 2);      // Number of defects for defect category #4
samples.setElement(4, 3);      // Number of defects for defect category #5
samples.setElement(5, 4); // TOTAL number of defective parts in the sample


// Add the new sample subgroup to the chart
    chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, "");
```

[TypeScript]

```
let samples: QCSPCChartTS.DoubleArray =
QCSPCChartTS.DoubleArray.newDoubleArrayN(6);
//  Date initialized with current  time by default
let timestamp: Date = new Date();  // use a unique time stamp for every update
batchCounter++; / defined elsewhere but incremented with every update
// Place sample values in array
samples.setElement(0, 3);      // Number of defects for defect category #1
samples.setElement(1, 0);      // Number of defects for defect category #2
samples.setElement(2, 4);      // Number of defects for defect category #3
samples.setElement(3, 2);      // Number of defects for defect category #4
samples.setElement(4, 3);      // Number of defects for defect category #5
samples.setElement(5, 4); // TOTAL number of defective parts in the sample


// Add the new sample subgroup to the chart
    chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, "");
```

This is obscured in our example programs a bit because we use a special method to simulate defect data for n- and np-charts.

[JavaScript]

```
var samples = chartdata.simulateDefectRecordMeanType(50 * 0.134,
QCSPCChartTS.SPCControlChartData.NUMBER_DEFECTIVE_PARTS_CHART);
```

```
// Add new sample record
chartdata.addNewSampleRecordDateSamples( timestamp, samples);
```

[TypeScript]

```
let samples: QCSPCChartTS.DoubleArray = chartdata.simulateDefectRecordMeanType(50
* 0.134, QCSPCChartTS.SPCControlChartData.NUMBER_DEFECTIVE_PARTS_CHART);
// Add new sample record
chartdata.addNewSampleRecordDateSamples( timestamp, samples);
```

This particular overload for **chartdata.simulateDefectRecord** knows that since it is a NUMBER_DEFECTIVE_PARTS_CHART chart (np-chart), and that since the **ChartData** object was setup with five categories in the **initSPCTimeAttributeControlChart** call, that is should return a **DoubleArray** with (5 + 1 = 6) elements, the first five elements representing simulated defect counts for the five defect categories, and the sixth element the simulated defective parts count. The defect category count data of the *samples* array is only used in the table part of the display; the defect category counts play NO role in the actual SPC chart. The only value that is used in plotting the SPC chart is the last element in the *samples* array, the defective parts count for the sample subgroup.

Updating c- and u-charts

In c- and u-charts the number of defective parts is of no consequence. The only thing that is tracked is the number of defects. Therefore, there is no extra array element tacked onto the end of the *samples* array. Each element of the *samples* array corresponds to the total number of defects for a given defect category. If the *numcategories* parameter in the **initSPCEventAttributeControlChart**, **initSPCTimeAttributeControlChart** or **initSPCBatchAttributeControlChart** is initialized to five, the total number of elements in the *samples* array should be five. For example:

[JavaScript]

```
var samples = QCSPCChartTS.DoubleArray.newDoubleArrayN(5);
//  Date initialized with current  time by default
var timestamp= new Date();  // use a unique time stamp for every update
batchCounter++; / defined elsewhere but incremented with every update
// Place sample values in array
samples.setElement(0, 3);      // Number of defects for defect category #1
samples.setElement(1, 0);      // Number of defects for defect category #2
samples.setElement(2, 4);      // Number of defects for defect category #3
samples.setElement(3, 2);      // Number of defects for defect category #4
samples.setElement(4, 3);      // Number of defects for defect category #5


// Add the new sample subgroup to the chart
    chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, "");
```

[TypeScript]

```
let samples: QCSPCChartTS.DoubleArray =
QCSPCChartTS.DoubleArray.newDoubleArrayN(56);
//  Date initialized with current  time by default
let timestamp: Date = new Date();// use a unique time stamp for every update
batchCounter++; / defined elsewhere but incremented with every update
// Place sample values in array
samples.setElement(0, 3);      // Number of defects for defect category #1
samples.setElement(1, 0);      // Number of defects for defect category #2
samples.setElement(2, 4);      // Number of defects for defect category #3
samples.setElement(3, 2);      // Number of defects for defect category #4
samples.setElement(4, 3);      // Number of defects for defect category #5


// Add the new sample subgroup to the chart
    chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, "");
```

While the table portion of the display can display defect data broken down into categories, only the sum of the defects for a given sample subgroup is used in creating the actual SPC chart. Note that the code below, extracted from the AttributeControlCharts.BuildCChart example, uses a different **chartdata.simulateDefectRecordMeanType** method to simulate the defect data.

Chart Header Information, Measured Data and Calculated Value Table

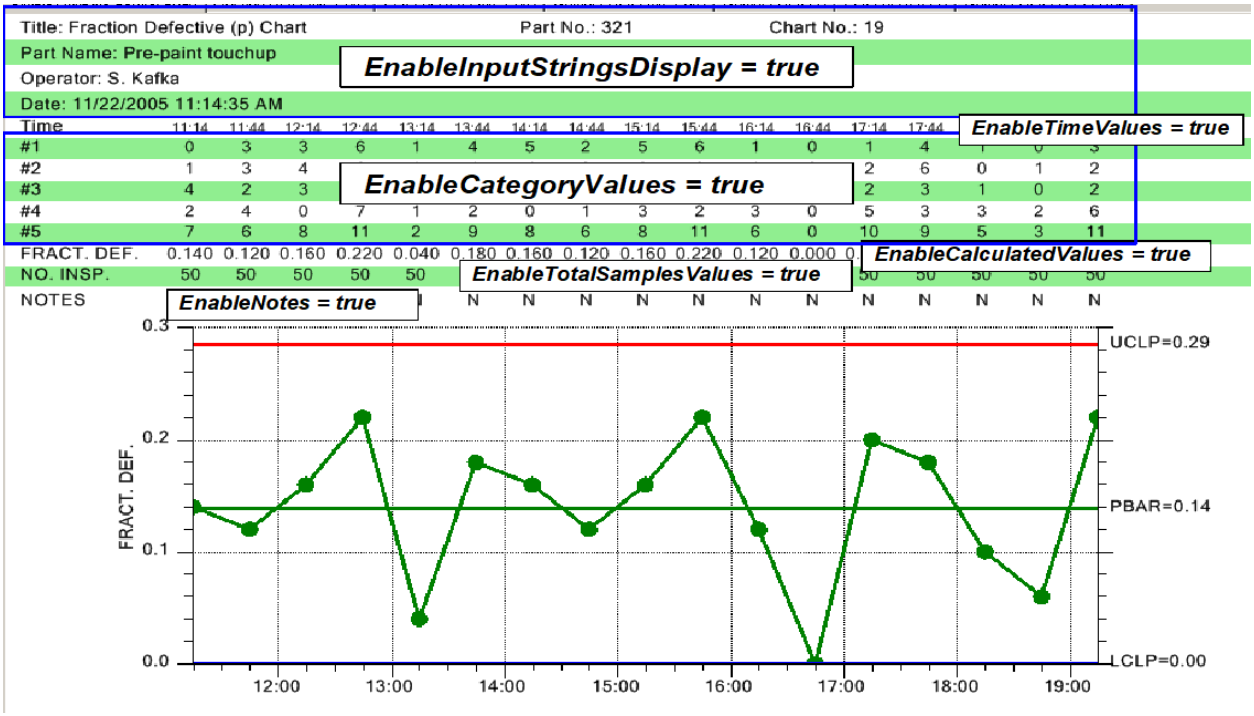
Standard worksheets used to gather and plot SPC data consist of three main parts.

- The first part is the header section, identifying the title of the chart, the monitored process, the machine operator, part number and other important information specific to the chart.
- The second part is the measurement data recording and calculation section, organized as a table recording the sample data and calculated values in a neat, readable fashion.
- The third part plots the calculated SPC values as a SPC chart.

The chart includes options that enable the programmer to customize and automatically include header information along with a table of the measurement and calculated data, in the SPC chart.

The following properties enable sections of the chart header and table:

EnableInputStringsDisplay
EnableCategoryValues
EnableCalculatedValues
EnableTotalSamplesValues
EnableNotes
EnableTimeValues



The example code below is extracted from the **AttributeControlCharts.BuildPChartF** example.

[JavaScript]

```
  attribchart.setPreferredSize(800, 600);


attribchart.setXAxisStringLabelMode( QCSPCChartTS.SPCChartObjects.AXIS_LABEL_MODE_
TIME);

attribchart.setGraphStopPosX(0.825);
attribchart.setGraphStartPosX(0.18);

attribchart.setChartAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_SY
MBOL);
attribchart.setEnableScrollBar(true);
attribchart.setEnableDisplayOptionToggles(true);
attribchart.setEnableCategoryValues(true);
attribchart.setEnableCalculatedValues(true);
attribchart.setEnableAlarmStatusValues(true);
attribchart.setEnableChartToggles(true);
attribchart.setTableAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_BA
R);

attribchart.setHeaderStringsLevel(QCSPCChartTS.SPCControlChartData.HEADER_STRINGS_
LEVEL1);

var chartdata = attribchart.getChartData();
if (chartdata) {
```

```
        chartdata.setTitle(charttitle);
        chartdata.setPartNumber("321");
        chartdata.setChartNumber("19");
        chartdata.setPartName( "Pre-paint touchup");
        chartdata.setOperator("S. Kafka");
        chartdata.setChartDescriptor("Fraction Def. Parts");
}

.
.
.
attribchart.rebuildChartUsingCurrentData();
```

[TypeScript]

```
attribchart.setPreferredSize(800, 600);


attribchart.setXAxisStringLabelMode( QCSPCChartTS.SPCChartObjects.AXIS_LABEL_MODE_
TIME);

attribchart.setGraphStopPosX(0.825);
attribchart.setGraphStartPosX(0.18);

attribchart.setChartAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_SY
MBOL);
attribchart.setEnableScrollBar(true);
attribchart.setEnableDisplayOptionToggles(true);
attribchart.setEnableCategoryValues(true);
attribchart.setEnableCalculatedValues(true);
attribchart.setEnableAlarmStatusValues(true);
attribchart.setEnableChartToggles(true);

attribchart.setTableAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_BA
R);

attribchart.setHeaderStringsLevel(QCSPCChartTS.SPCControlChartData.HEADER_STRINGS_
LEVEL1);

let chartdata: QCSPCChartTS.SPCControlChartData | null =
attribchart.getChartData();
if (chartdata) {
            chartdata.setTitle(charttitle);
            chartdata.setPartNumber("321");
            chartdata.setChartNumber("19");
            chartdata.setPartName( "Pre-paint touchup");
            chartdata.setOperator("S. Kafka");
            chartdata.setChartDescriptor("Fraction Def. Parts");
}

    .
    .
    .
    attribchart.rebuildChartUsingCurrentData();
```

The input header strings display has four sub-levels that display increasing levels of information. The input header strings display level is set using the charts HeaderStringsLevel property. Strings that can be displayed are: Title, PartNumber, ChartNumber, DateString, PartName, Operation, Machine, SpecificationLimits, Gage,

UnitOfMeasure, ZeroEquals and DateString. The four levels and the information displayed is listed below:

HEADER_STRINGS_LEVEL0	Display no header information
HEADER_STRINGS_LEVEL1	Display minimal header information: Title, PartNumber, ChartNumber, DateString
HEADER_STRINGS_LEVEL2	Display most header strings: Title, PartNumber, ChartNumber, PartName, Operation, Operator, Machine, DateString
HEADER_STRINGS_LEVEL3	Display all header strings: Title, PartNumber, ChartNumber, DateString, PartName, Operation, Machine, SpecificationLimits, Gage, UnitOfMeasure, ZeroEquals and DateString

The example program AttributeControlCharts.BuildPChartF demonstrates the use of the **HeaderStringsLevel** property. The example below displays a minimum set of header strings (HeaderStringsLevel = SPCControlChartData.HEADER_STRINGS_LEVEL1).

[JavaScript]

```
  var chartdata = attribchart.getChartData();
  if (chartdata) {
    chartdata.setTitle( "Fraction Defective (p) Chart");
    chartdata.setPartNumber("321");
    chartdata.setChartNumber("19");
    chartdata.setPartName= "Pre-paint touchup");
    chartdata.setTheOperator("S. Kafka");
    var today = new Date();
    chartdata.setDateString(today.toLocaleString());
 }

attribchart.setHeaderStringsLevel(QCSPCChartTS.SPCControlChartData.HEADER_STRINGS_
LEVEL1);
```

[TypeScript]

```
  let chartdata: QCSPCChartTS.SPCControlChartData | null =
attribchart.getChartData();

  if (chartdata) {
    chartdata.setTitle( "Fraction Defective (p) Chart");
    chartdata.setPartNumber("321");
    chartdata.setChartNumber("19");
    chartdata.setPartName= "Pre-paint touchup");
    chartdata.setTheOperator("S. Kafka");
    let  today: Date = new Date();
    chartdata.setDateString(today.toLocaleString());
 }
attribchart.setHeaderStringsLevel(QCSPCChartTS.SPCControlChartData.HEADER_STRINGS_
LEVEL1);
```

The example below displays a maximum set of header strings (HeaderStringsLevel = SPCControlChartData.HEADER_STRINGS_LEVEL3).

[JavaScript]

```
// Set the strings used in the header section of the table
 var chartdata = attribchart.getChartData();
 if (chartdata) {
chartdata.setTitle ( "Fraction Defective (p) Chart");
chartdata.setPartNumber( "283501");
chartdata.setChartNumber("17");
chartdata.setTheOperator("B. Cornwall");
chartdata.setPartName ("Left Front Fender");
chartdata.setOperation( "Painting");
chartdata.setSpecificationLimits("");
chartdata.setMachine("#11");
chartdata.setGage("");
chartdata.setUnitOfMeasure( "");
chartdata.setZeroEquals("");
var  today = new Date();
chartdata.setDateString(today.toLocaleString());

}
attribchart.setHeaderStringsLevel( SPCControlChartData.HEADER_STRINGS_LEVEL3);
```

[TypeScript]

```
// Set the strings used in the header section of the table
 let chartdata: QCSPCChartTS.SPCControlChartData | null =
attribchart.getChartData();
 if (chartdata) {
chartdata.setTitle ( "Fraction Defective (p) Chart");
chartdata.setPartNumber( "283501");
chartdata.setChartNumber("17");
chartdata.setTheOperator("B. Cornwall");
chartdata.setPartName ("Left Front Fender");
chartdata.setOperation( "Painting");
chartdata.setSpecificationLimits("");
chartdata.setMachine("#11");
chartdata.setGage("");
chartdata.setUnitOfMeasure( "");
chartdata.setZeroEquals("");
let  today: Date = new Date();
chartdata.setDateString(today.toLocaleString());

}
attribchart.setHeaderStringsLevel( SPCControlChartData.HEADER_STRINGS_LEVEL3);
```

The identifying string displayed in front of the input header string can be any string that you want, including non-English language string. For example, if you want the input header string for the Title to represent a project name:

Project Name: Project XKYZ for PerQuet

Set the properties:

[JavaScript / TypeScript]

```
chartdata.setTitle ("Project XKYZ for PerQuet");
chartdata.setTitleHeader( "Project Name:");
```

Change other headers using the ChartData properties listed below.

- TitleHeader
- PartNumberHeader
- ChartNumberHeader
- PartNameHeader
- OperationHeader
- OperatorHeader
- MachineHeader
- DateHeader
- SpecificationLimitsHeader
- GageHeader
- UnitOfMeasureHeader
- ZeroEqualsHeader
- NotesHeader

Even though the input header string properties have names like Title, PartNumber, ChartNumber, etc., those names are arbitrary. They are really just placeholders for the strings that are placed at the respective position in the table. You can display any combination of strings that you want, rather than the ones we have selected by default, based on commonly used standardized SPC Control Charts.

Depending on the control chart type, you may want to customize the category header strings. In most of our examples, we use the category header strings: Scratch, Burr, Dent, Seam, and Other, to represent common defect categories. You can change these strings to anything that you want using the **chartdata.setSampleRowHeaderString** method. See the example program AttributeControlCharts.BuildNPChart.

[JavaScript / TypeScript]

```
// Set the table row headers strings for defect categories
chartdata.setSampleRowHeaderString(0, "    Scratch");
chartdata.setSampleRowHeaderString(1, "    Burr");
chartdata.setSampleRowHeaderString(2, "    Dent");
chartdata.setSampleRowHeaderString(3, "    Seam");
chartdata.setSampleRowHeaderString(4, "    Other");
```

The **ChartTable** property of the chart has properties that further customize the chart. The default table background uses the accounting style green-bar striped background. You can change this using the **ChartTable.TableBackgroundMode** property**.** Set the value to one of the TableBackgroundMode constants:

TABLE_NO_COLOR_BACKGROUND	Constant specifies that the table does not use a background color.

TABLE_SINGLE_COLOR_BACKGROUND	Constant specifies that the table uses a single color for the background (backgroundColor1)
TABLE_STRIPED_COLOR_BACKGROUND	Constant specifies that the table uses horizontal stripes of color for the background (backgroundColor1 and backgroundColor2)
TABLE_SINGLE_COLOR_BACKGROUND_GRIDCELL	Constant specifies that the table uses a grid background, with backgroundColor1 the overall background color and backgroundColor2 the color of the grid lines.

Extracted from the AttributeControlCharts.BuildPChartP example program

[JavaScript]

```
    var charttable = attribchart.getChartTable();
    if (charttable)
    {

charttable.setTableBackgroundMode(   QCSPCChartTS.SPCGeneralizedTableDisplay.TABLE
_STRIPED_COLOR_BACKGROUND);
        charttable.setBackgroundColor1( QCSPCChartTS.ChartColor.BISQUE);

charttable.setBackgroundColor2( QCSPCChartTS.ChartColor.LIGHTGOLDENRODYELLOW);
    }
```

[TypeScript]

```
    let charttable: QCSPCChartTS.SPCGeneralizedTableDisplay | null =
attribchart.getChartTable();
    if (charttable)
    {

charttable.setTableBackgroundMode(   QCSPCChartTS.SPCGeneralizedTableDisplay.TABLE
_STRIPED_COLOR_BACKGROUND);
        charttable.setBackgroundColor1( QCSPCChartTS.ChartColor.BISQUE);

charttable.setBackgroundColor2( QCSPCChartTS.ChartColor.LIGHTGOLDENRODYELLOW);
    }
```

Extracted from the AttributeControlCharts.BuildNPChart example program

[JavaScript]

```
    var charttable = attribchart.getChartTable();
```

```
    if (charttable)
    {

charttable.setTableBackgroundMode(   QCSPCChartTS.SPCGeneralizedTableDisplay.TABLE
_SINGLE_COLOR_BACKGROUND );
        charttable.setBackgroundColor1( QCSPCChartTS.ChartColor.LIGHTGRAY);
    }
```

[TypeScript]

```
    let charttable: QCSPCChartTS.SPCGeneralizedTableDisplay | null =
attribchart.getChartTable();
    if (charttable)
    {

charttable.setTableBackgroundMode(   QCSPCChartTS.SPCGeneralizedTableDisplay.TABLE
_SINGLE_COLOR_BACKGROUND );
        charttable.setBackgroundColor1( QCSPCChartTS.ChartColor.LIGHTGRAY);
    }
```

Extracted from the AttributeControlCharts.BuildCChart example program

[JavaScript]

```
    var charttable = attribchart.getChartTable()

    if (charttable)
    {
        charttable.setTableBackgroundMode(QCSPCChartTS.SPCGeneralizedTableDisplay.
TABLE_NO_COLOR_BACKGROUND );
    }
```

[TypeScript]

```
    let charttable: QCSPCChartTS.SPCGeneralizedTableDisplay | null =
attribchart.getChartTable()

    if (charttable)
    {
        charttable.setTableBackgroundMode(QCSPCChartTS.SPCGeneralizedTableDisplay.
TABLE_NO_COLOR_BACKGROUND );
    }
```

Table and Chart Fonts

There are a large number of fonts that you have control over, both the fonts in the table and the fonts in the chart. The programmer can select a default font (as in the case of non-US character set), or select individual fonts for different elements of the table and charts.

Table Fonts

The table fonts are accessed through the charts **ChartTable** property. Below is a list of accessible table fonts:

TimeLabelFont	The font used in the display of time values in the table.
SampleLabelFont	The font used in the display of sample numeric values in the table.

CalculatedLabelFont	The font used in the display of calculated values in the table.
StringLabelFont	The font used in the display of header string values in the table.
NotesLabelFont	The font used in the display of notes values in the table.

Extracted from the example AttributeControlCharts.PercentDefectivePartsControlChart

[JavaScript]

```
    var charttable = attribchart.getChartTable();
    if (charttable)
    {

charttable.setSampleLabelFont( QCSPCChartTS.ChartFont.newChartFont("Arial",
QCSPCChartTS.ChartFont.REGULAR, 14));

charttable.setCalculatedLabelFont( QCSPCChartTS.ChartFont.newChartFont("Arial",
QCSPCChartTS.ChartFont.REGULAR, 14));
    }
```

[TypeScript]

```
let charttable: QCSPCChartTS.SPCGeneralizedTableDisplay | null =
attribchart.getChartTable();
    if (charttable)
    {

charttable.setSampleLabelFont( QCSPCChartTS.ChartFont.newChartFont("Arial",
QCSPCChartTS.ChartFont.REGULAR, 14));

charttable.setCalculatedLabelFont( QCSPCChartTS.ChartFont.newChartFont("Arial",
QCSPCChartTS.ChartFont.REGULAR, 14));
    }
```

The **SPCGeneralizedTableDisplay**
class has a static property, **SPCGeneralizedTableDisplay.DefaultTableFont**, that sets the default font name. Use this if you want to establish a default font for all of the text in a table. This static property must be set BEFORE the chart is instantiated. All of the other fonts in the class are initialized to this default font. The DefaultTableFont is initialized internally using the following code:

```
public static defaultTableFont: ChartFont =
ChartFont.newChartFont3(SPCChartStrings.getString(SPCStringEnum.chartFont),
ChartFont.PLAIN, 14);
```

so you can see that it looks to the value of the chartFont in SPCChartStrings lookup table, obtained by calling SPCChartStrings.getString(SPCStringEnum.chartFont). So if you change the font name in SPCChartStrings, all fonts in the software will use that font type

[JavaScript]

```
QCSPCChartTS.SPCGeneralizedTableDisplay.setDefaultTableFont(
    QCSPCChartTS.ChartFont.newChartFont("Arial",  QCSPCChartTS.ChartFont.REGULAR,
14));
```

```
var attribchart =
QCSPCChartTS.SPCBatchAttributeControlChart.newSPCBatchAttributeControlChartChartTy
peSubgroupSize(htmlcanvas, spccharttype, 1, subgroupsize, numberpointsinview);

.
.
.
  var charttable = attribchart.getChartTable();
    if (charttable)
    {

charttable.setSampleLabelFont( QCSPCChartTS.ChartFont.newChartFont("Arial",
QCSPCChartTS.ChartFont.REGULAR, 14));

charttable.setCalculatedLabelFont( QCSPCChartTS.ChartFont.newChartFont("Arial",
QCSPCChartTS.ChartFont.REGULAR, 14));
    }
```

[TypeScript]

```
QCSPCChartTS.SPCGeneralizedTableDisplay.setDefaultTableFont(
    QCSPCChartTS.ChartFont.newChartFont("Arial",  QCSPCChartTS.ChartFont.REGULAR,
14));
// Initialize the SPCBatchAttributeControlChart
let attribchart: QCSPCChartTS.SPCChartBase =
QCSPCChartTS.SPCBatchAttributeControlChart.newSPCBatchAttributeControlChartChartTy
peSubgroupSize(htmlcanvas, spccharttype, 1, subgroupsize, numberpointsinview);
.
.
let charttable: QCSPCChartTS.SPCGeneralizedTableDisplay | null =
attribchart.getChartTable();
    if (charttable)
    {

charttable.setSampleLabelFont( QCSPCChartTS.ChartFont.newChartFont("Arial",
QCSPCChartTS.ChartFont.REGULAR, 14));

charttable.setCalculatedLabelFont( QCSPCChartTS.ChartFont.newChartFont("Arial",
QCSPCChartTS.ChartFont.REGULAR, 14));
    }


.
```

Chart Fonts

There are default chart fonts that are instance objects in the **SPCChartObjects** class. They establish the default fonts for related chart objects and they are initially set to the value of the SPCChartBase.DefaultChartFontString. But you can change them if you want to.

Property name	Description	Default size
AxisLabelFont	The font used to label the x- and y- axes.	12
AxisTitleFont	The font used for the axes titles.	12
MainTitleFont	The font used for the chart title.	18
SubheadFont	The font used for the chart subhead.	14
ToolTipFont	The tooltip font.	12
FooterFont	The font used for the chart footer.	12
AnnotationFont	The annotation font.	12

ControlLimitLabelFont	The font used to label the control limits	12
LegendFont	The font used to label the legend items	12

Extracted from the example AttributeControlCharts.BuildPChartF

[JavaScript]
```
QCSPCChartTS.SPCChartBase.setDefaultChartFontString("Arial");


// Initialize the SPCBatchAttributeControlChart
var attribchart =
QCSPCChartTS.SPCBatchAttributeControlChart.newSPCBatchAttributeControlChartChartTy
peSubgroupSize(htmlcanvas, spccharttype, 1, subgroupsize, numberpointsinview);
let primarychart: QCSPCChartTS.SPCChartObjects | null =
attribchart.getPrimaryChart();
if (primarychart)
{

primarychart.setAxisTitleFont(
    QCSPCChartTS.ChartFont.newChartFont("Arial",  QCSPCChartTS.ChartFont.REGULAR,
14));
}
```

[TypeScript]
```
QCSPCChartTS.SPCChartBase.setDefaultChartFontString("Arial");


// Initialize the SPCBatchAttributeControlChart
var attribchart =
QCSPCChartTS.SPCBatchAttributeControlChart.newSPCBatchAttributeControlChartChartTy
peSubgroupSize(htmlcanvas, spccharttype, 1, subgroupsize, numberpointsinview);
var primarychart = attribchart.getPrimaryChart();
if (primarychart)
{

primarychart.setAxisTitleFont(
    QCSPCChartTS.ChartFont.newChartFont("Arial",  QCSPCChartTS.ChartFont.REGULAR,
14));
}
```

Font Size (using setPreferredSize)

The fonts all have an initial, default size, usually between 10 and 14 device units, regardless of the window size. If you place the chart in a small window, the fonts size may be too a large for the window and the text within the window may overlap. If you place the chart in a full screen window, there may be a lot of wasted whitespace in the chart. The resolution of the output device will also effect the apparent size of the text, with small phone type displays in portrait mode showing crowded overlapping text, and large computer screens or tablets showing a lot of whitespace. This is the main reason that many web pages have multiple variants the browser can choose from, depending on the resolution of the browser display device. The display output orientation, whether portrait (usually mobile devices) or landscape, can also effect the crowding of text.

In all versions of QCSPCChart we have included a function, setPreferredSize, which allows a degree of device independence when displaying text in charts. It establishes a relative screen size, against which you size your text. In all of our examples we explicitly set a preferred size of (800x600) device units, representing a typical Canvas window size within an HTML page.

```
var spcchart = QCSPCChartTS.SPCBatchAttributeControlChart.newSPCBatchAttributeCont
rolChartChartTypeSubgroupSize(htmlcanvas, spccharttype, 1, subgroupsize, numberpoi
ntsinview);

var spcchart .setPreferredSize(800, 600);
```

All of the fonts used in the software are initialized to a size which works pretty good on average for a page size of (800x600). If you place the chart in a much larger Canvas (2400x1800) for example, but leave the **PreferredSize** property still sized for (800x600), all fonts will be scaled up by a factor of 3, taking into account the new, larger display window size. The software assumes that you want the relative size of the charts text to look the same, regardless of whether you displaying the chart in a (800x600) or a (2400x1800) output device. So, with a **PreferredSize** of (800x600), but an actual display size of (2400x1800), a size 12 font will temporarily be increased to a size 36 font for the purposes of output scaling. If you were to call **setPreferredSize**(2400, 1800) in your program, the software would output a size 12 font to the much larger display, making it appear very small with respect to the overall chart size. This simple resizing works best if the aspect ratio (W/H) of the display screen matches the aspect ratio of the **PreferredSize** dimensions. If they don't match, for example a change from landscape orientation to portrait mode, the font size scaling factor will use the scale factor (width or height) which changes the least when calculating (preferred size)/(actual size ratios) for width and height. The actual scale factor calculation routine in the software looks like:

```
    public calcResizedWindowFontMultiplier(preferreddim: ChartDimension,
        actualdim: ChartDimension): number {
        let changewidth: number = actualdim.getWidth() / preferreddim.getWidth();
        let changeheight: number = actualdim.getHeight() /
preferreddim.getHeight();
        let result: number = 1.0;
        if (this.resizeMode == ChartConstants.NO_RESIZE_OBJECTS)
            result = 1.0;
        else if (this.resizeMode == ChartConstants.AUTO_RESIZE_OBJECTS)
            result = Math.min(changewidth, changeheight);
        else if (this.resizeMode == ChartConstants.MANUAL_RESIZE_OBJECTS)
            result = this.resizeMultiplier; // return manually set value
        return result;
    }
```

In that code you will see a resizeMode property, which you can set to either ChartConstants.AUTO_RESIZE_OBJECTS (the default mode), or ChartConstants.NO_RESIZE_OBJECTS, using the setResizeMode method of the chart.

```
spcchart.setResizeMode( QCSPCChartTS.ChartConstants.NO_RESIZE_OBJECTS);
```

If the resizeMode is explicitly set to ChartConstants.AUTO_RESIZE_OBJECTS, the scaling factor for up- or down-scaling font size will always be 1.

SPC Charts without a Table

If you don't want any of the items we have designated table items, just call the **useNoTable** method. That method removes all of the table items, and displays the primary and/or secondary charts with a simple title and optional histograms.

This initialization method initializes the most important values in the creation of a SPC chart.

[TypeScript]

```
public useNoTable(primary: boolean, secondary: boolean,
        histograms: boolean, title: string)
```

Parameters

primarychart
: Set to true to display primary chart.

secondarychart
: Set to true to display secondary chart.

histograms
: Set to true to display chart histograms

title
: Specifies the title for the charts

[JavaScript / TypeScript]

```
    attribchart.useNoTable(true, true, true,"XBar-R Chart");
```

Chart Position

If the SPC chart does not include frequency histograms on the left (they take up about 20% of the available chart width), you can adjust the left and right edges of the chart using the **GraphStartPosX** and **GraphStopPlotX** properties to allow for more room in the display of the data. This also affects the table layout, because the table columns must line up with the chart data points.

[JavaScript / TypeScript]

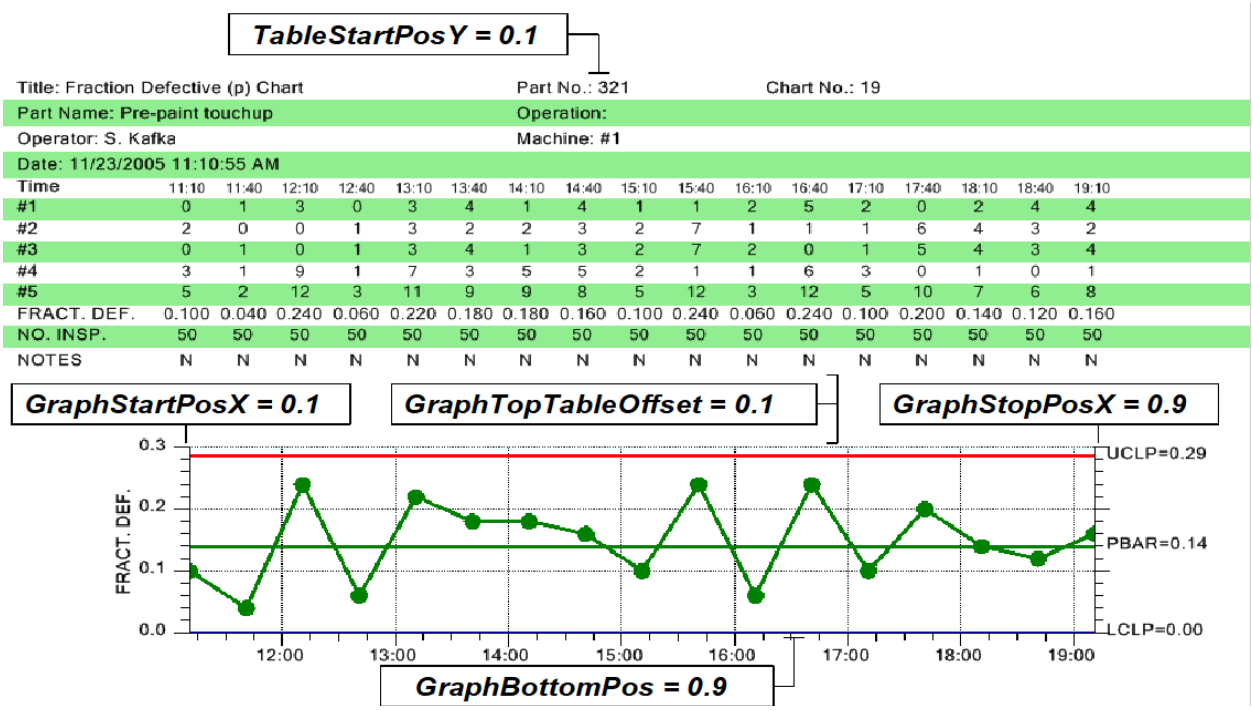
```
attribchart.setGraphStartPosX( 0.1); // start here
attribchart.setGraphStopPosX( 0.875);  // end here
```

There is not much flexibility positioning the top and bottom of the chart. Depending on the table items enabled, the table starts at the position defined by the **TableStartPosY** property, and continues until all of the table items are displayed. It then offsets the top of the primary chart with respect to the bottom of the table by the value of the property **GraphTopTableOffset**. The value of the property **GraphBottomPos** defines the bottom of the graph. The default values for these properties are:

[JavaScript / TypeScript]

```
attribchart.setTableStartPosY( 0.00);
attribchart.setGraphTopTableOffset(0.02);
attribchart.setInterGraphMargin(0.075);
attribchart.setGraphBottomPos(0.925);
```

The picture below uses different values for these properties in order to emphasize the affect that these properties have on the resulting chart.



SPC Control Limits

There are two methods you can use to set the SPC control limit for a chart. The first method explicitly sets the limits to values that you calculate on your own, because of some analysis that a quality engineer does on previously collected data. The second method auto-calculates the limits using the algorithms supplied in this software.

The quick way to set the limit values and limit strings is to use the charts **chartdata.setControlLimitValues** and **chartdata.setControlLimitStrings** methods. This method only works for the default +-3-sigma level control limits, and not any others you may have added using the charts **addAdditionalControlLimit** method discussed in the *Multiple Control Limits* section. The data values in the *controllimitvalues* and *controllimitstrings* arrays used to pass the control limit information must be sorted in the following order:

[SPC_PRIMARY_CONTROL_TARGET,
SPC_PRIMARY_LOWER_CONTROL_LIMIT,
SPC_PRIMARY_UPPER_CONTROL_LIMIT]

[JavaScript]

```
var chartdata = attribchart.getChartData();

var controllimitvalues = [0.123, 0.025, 0.222];
if (chartdata)
    chartdata.setControlLimitValues(controllimitvalues);;


var controllimitstrings: string = ["Target","LCL", "UCL"];

if (chartdata)
  chartdata.setControlLimitStrings(controllimitstrings);
```

[TypeScript]

```
let chartdata: QCSPCChartTS.SPCControlChartData | null =
attribchart.getChartData();
let controllimitvalues: number[] = [0.123, 0.025, 0.222];
if (chartdata)
    chartdata.setControlLimitValues(controllimitvalues);;


let controllimitstrings: string [] = ["Target","LCL", "UCL"];

if (chartdata)
  chartdata.setControlLimitStrings(controllimitstrings);
```

You can also set the control limit values and control limit text one value at a time using the **chartdata.setControlLimitValue** and **chartdata.setControlLimitStrings** methods.

A more complicated way to set the control limits explicitly is to first grab a reference to the **SPCControlLimitRecord** for a given control limit, and then change the value of that

control limit, and the control limit text, if desired. The example below sets the control limit values and text for the three control limits (target value, upper control limit, and lower control limit) of the primary chart, and the three control limit values for the secondary chart.

[JavaScript]

```
//target control limit primary chart
 var chartdata = attribchart.getChartData();

 var primarytarget =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_CONTR
OL_TARGET);
      primarytarget.setControlLimitValue ( 0.123);
      primarytarget.setControlLimitText("TARGET");

      //lower control limit primary chart
      var primarylowercontrollimit =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_LOWER
_CONTROL_LIMIT);
      primarylowercontrollimit.setControlLimitValue(0.025);
      primarylowercontrollimit.setControlLimitText("LCL");

      //upper control limit primary chart
      var primaryuppercontrollimit =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_UPPER
_CONTROL_LIMIT);
      primaryuppercontrollimit.setControlLimitValue( 0.222);
      primaryuppercontrollimit.setControlLimitText( "UCL");
```

[TypeScript]

```
//target control limit primary chart
 let chartdata: QCSPCChartTS.SPCControlChartData | null =
attribchart.getChartData();
 let primarytarget: QCSPCChartTS.SPCControlLimitRecord =

chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_CONTR
OL_TARGET);

      primarytarget.setControlLimitValue ( 0.123);
      primarytarget.setControlLimitText("TARGET");

      //lower control limit primary chart
      let primarylowercontrollimit: QCSPCChartTS.SPCControlLimitRecord =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_LOWER
_CONTROL_LIMIT);
      primarylowercontrollimit.setControlLimitValue(0.025);
      primarylowercontrollimit.setControlLimitText("LCL");

      //upper control limit primary chart
      let primaryuppercontrollimit: QCSPCChartTS.SPCControlLimitRecord =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_UPPER
_CONTROL_LIMIT);
      primaryuppercontrollimit.setControlLimitValue( 0.222);
      primaryuppercontrollimit.setControlLimitText( "UCL");
```

We also added a method (add3SigmaControlLimits) which will generate multiple control limits, for +-1, 2, and 3- sigma levels, based on an initial specification of the target value, and the +-3 sigma control limits. This is most useful if you want to generate +-1, 2 and 3-sigma control limits in order to fill in between them with a zone fill color. See the MultiLimitCharts.BuildMultiLimitXBarRChart. If you call the autoCalculateControlLimits method, the initial +-1,2 and 3-sigma control limit values will be altered to the new, calculated values, but the control limit lines remain, with their new values. Since you do not normally want to be generating alarm messages for excursions into the +-1 and 2-sigma limit areas, the add3SigmaControl limits has the option of disabling alarm notification in the case of +-1 and +-2 alarm conditions.

[JavaScript]

```
var target = 0.123;
var lowlim = 0.025,
var highlim = 0.222;
var limitcheck = false;

var primarychart = attribchart.getPrimaryChart();

if (primarychart)
{
  primarychart.add3SigmaControlLimits(target, lowlim, highlim, limitcheck);
  primarychart.setControlLimitLineFillMode( true);
}
```

[TypeScript]

```
let target: number = 0.123;
let lowlim: number = 0.025,
let highlim: number = 0.222;
let limitcheck: boolean = false;

let primarychart: QCSPCChartTS.SPCChartObjects | null =
attribchart.getPrimaryChart();

if (primarychart)
{
  primarychart.add3SigmaControlLimits(target, lowlim, highlim, limitcheck);
  primarychart.setControlLimitLineFillMode( true);
}
```

The second way to set the control limits is to call the **autoCalculateControlLimits** method. You must have already added a collection of sampled data values to the charts **ChartData** SPC data object before you can call this method, since the method uses the internal **ChartData** object to provide the historical values needed in the calculation.

[JavaScript]

```
// Must have data loaded before any of the Auto.. methods are called
SimulateData(attribchart, numssampleintervals, fractiondefective, subgroupsize);
```

```
// Calculate the SPC control limits for both graphs of the current SPC

attribchart.autoCalculateControlLimits();
```

[TypeScript]

```
// Must have data loaded before any of the Auto.. methods are called
this.SimulateData(attribchart, numssampleintervals, fractiondefective,
subgroupsize);

// Calculate the SPC control limits for both graphs of the current SPC

attribchart.autoCalculateControlLimits()
```

You can add data to the **chartdata** object, auto-calculate the control limits to establish the SPC control limits, and then continue to add new data values. Alternatively, you can set the SPC control limits explicitly as the result of previous runs, using the previously described **chartdata.setControlLimitValues** method, and add new sampled data values to the **ChartData** object, and after a certain number of updates call the **autoCalculateControlLimits** method to establish new control limits.

[JavaScript / TypeScript]

```
updateCount++;
chartdata.addNewSampleRecordDateSamples(timestamp, samples);
if (updateCount > 50) // After 50 sample groups and calculate limits on the fly
{
// Calculate the SPC control limits for the primary chart of the current SPC chart
    attribchart.autoCalculateControlLimits();
    // Scale the y-axis of the primary chart to display all data and control
limits
    attribchart.autoScalePrimaryChartYRange();

}
```

Need to exclude records from the control limit calculation? Call the **chartdata.excludeRecordFromControlLimitCalculations** method, passing in true to exclude the record.

[JavaScript]

```
var i = 0;
for ( i=0; i < 10; i++)
      chartdata.excludeRecordFromControlLimitCalculations(i,true);
```

[TypeScript]

```
let i: number = 0;
for ( i=0; i < 10; i++)
      chartdata.excludeRecordFromControlLimitCalculations(i,true);
```

Formulas Used in Calculating Control Limits for Attribute Control Charts

The SPC control limit formulas used in the software derive from the following source:

Fraction Defective Parts, Number Defective Parts, Number Defects, Number Defects Per Unit - "Introduction to Statistical Quality Control" by Douglas C. Montgomery, John Wiley and Sons, Inc. 2001.

Percent Defective Parts - "SPC Simplified – Practical Steps to Quality" by Robert T. Amsden, Productivity Inc., 1998.

SPC Control Chart Nomenclature

UCL = Upper Control Limit

LCL = Lower Control Limit

Center line = The target value for the process

p = estimate (or average) of the fraction defective (or non-conforming) parts

P = estimate (or average) of the percent defective (or non-conforming) parts

c = estimate (or average) of the number of defects (or nonconformities)

u = estimate (or average) of the number of defects (or nonconformities) per unit

M = number of samples per subgroup

N = number of samples intervals

np = M * p (the sample subgroups size * average fraction defective parts)

dopu = defect opportunities per unit (applies only the DPMO chart)

dpmo = defects per million opportunities (applies only the DPMO chart)
calculated as: dpmo = (1,000,000 * numberOfDefects) / (sampleSize * dopu)

up = estimate (or average) of the dpmo values

Fraction Defective Parts – Also known as Fraction Non-Conforming or p-chart

$$UCL = p + 3 * \sqrt{\frac{p * (1 - p)}{M}}$$

$$Center\ line = p$$

$$LCL = p - 3 * \sqrt{\frac{p * (1 - p)}{M}}$$

Percent Defective Parts – Also known as Percent Non-Conforming or p-chart

$$UCL = p + 3 * \sqrt{\frac{p * (100\% - p)}{M}}$$

$$Center\ line = p$$

$$LCL = p - 3 * \sqrt{\frac{p * (100\% - p)}{M}}$$

Number of Defective Parts – Also known as the Number Nonconforming or np-chart

M = number of samples per subgroup

$p \approx \bar{p} =$ estimate (or average) of the fraction defective (or non-conforming) parts

$$p \approx \bar{p} = \frac{\sum_{j=1}^{N} D_j}{(M * N)}$$

np = M * p (the sample subgroups size * average fraction defective parts)

$$UCL = np + 3 * \sqrt{np * (1 - p)}$$

$$Center\ line = np$$

$$LCL = np - 3 * \sqrt{np * (1 - p)}$$

Number Defects Per Million – Also known as DPMO

$$UCL = up + 3000 * \sqrt{\frac{up}{(dopu * M)}}$$

$$Center\ line = up$$

$$LCL = up - 3000 * \sqrt{\frac{up}{(dopu * M)}}$$

Number of Defects Control Chart – Also known as Number Nonconformities or c-chart

$$UCL = c + 3 * \sqrt{(c)}$$

$$Center\ line = c$$

$$LCL = c - 3 * \sqrt{(c)}$$

Number of Defects per Unit Control Chart – Also known as Number Nonconformities per Unit or u-chart

$$UCL = u + 3 * \sqrt{\frac{u}{M}}$$

$$Center\ line = u$$

$$LCL = u - 3 * \sqrt{\frac{u}{M}}$$

Note that in the u-Chart formulas, the there is no independently calculated sigma value. That is because u-charts in general assume a Poisson distribution about the mean. In a Poisson distribution, the variance value of the distribution is equal to the mean, and the sigma value is the square root of the variance. You find this expression in the formulas for the UCL and LCL control limits.

$$sigma = \sqrt{\frac{u}{M}}$$

Variable SPC Control Limits

SPC control limits can be either fixed or variable. In actuality, the limits are always variable, but if you assign a set of limits and don't modify them, the limits will look fixed.

There are three ways to enter new SPC limit values. See the example program VariableControlLimits.BuildVariableLimitsPChart for an example of all three methods. First, you can use the method **chartdata.setControlLimitValues** method.
Extracted from the VariableControlLimits.BuildVariableLimitsPChart example

[JavaScript]

```
var initialControlLimits = [0.13, 0.0, 0.27];
var changeControlLimits = [0.11, 0.0, 0.25];
.
.
.
// Change limits at sample subgroup 10
if (i== 10)
{
    chartdata.setControlLimitValues(changeControlLimits);
}
chartdata.addNewSampleRecordBatchNumberDateSamples (batchnum, timestamp, samples,
notesstring);
```

[TypeScript]

```
let initialControlLimits: number[] = [0.13, 0.0, 0.27];
let changeControlLimits: number[] = [0.11, 0.0, 0.25];
```

```
.
.
.
// Change limits at sample subgroup 10
if (i== 10)
{
    chartdata.setControlLimitValues(changeControlLimits);
}
chartdata.addNewSampleRecordBatchNumberDateSamples(batchnum, timestamp, samples);
```

Second, you can use the **autoCalculateControlLimits** method. You must have already added a collection of sampled data values to the charts **chartdata** SPC data object before you can call this method, since the method uses the internal **chartdata** object to provide the historical values needed in the calculation.

[JavaScript / TypeScript]

```
.
.
.

// Variable Control Limits re-calculated every update after 10 using
//  autoCalculateControlLimits
if (i > 10)
    attribchart.autoCalculateControlLimits();
chartdata.addNewSampleRecordDateSamples(timestamp, samples);
```

Last, you can enter the SPC control limits with every new sample subgroup record, using one of the methods that include a control limits array parameter.

[JavaScript]

```
var initialControlLimits = [0.13, 0.0, 0.27];
var changeControlLimits  = [0.11, 0.0, 0.25];
var variableControlLimits = new QCSPCChartTS.DoubleArray();
.
.
.
//      Variable Control Limits updated using addNewSampleRecord
if (i== 10) // need to convert changeControlLimits to a DoubleArray
  variableControlLimits =
QCSPCChartTS.DoubleArray.newDoubleArrayArray(changeControlLimits);

chartdata.addNewSampleRecordBatchNumberDateSamplesControlLimitsNotes
 (batchCounter,timestamp, samples, variableControlLimits, notesstring);
```

[TypeScript]

```
let initialControlLimits: number [] = [0.13, 0.0, 0.27];
let changeControlLimits: number[]  = [0.11, 0.0, 0.25];
let variableControlLimits: QCSPCChartTS.DoubleArray = new
QCSPCChartTS.DoubleArray();
.
.
```

```
.
//      Variable Control Limits updated using addNewSampleRecord
if (i== 10) // need to convert changeControlLimits to a DoubleArray
  variableControlLimits =
QCSPCChartTS.DoubleArray.newDoubleArrayArray(changeControlLimits);

chartdata.addNewSampleRecordBatchNumberDateSamplesControlLimitsNotes(batchCounter,
samples, variableControlLimits, notesstring);
```

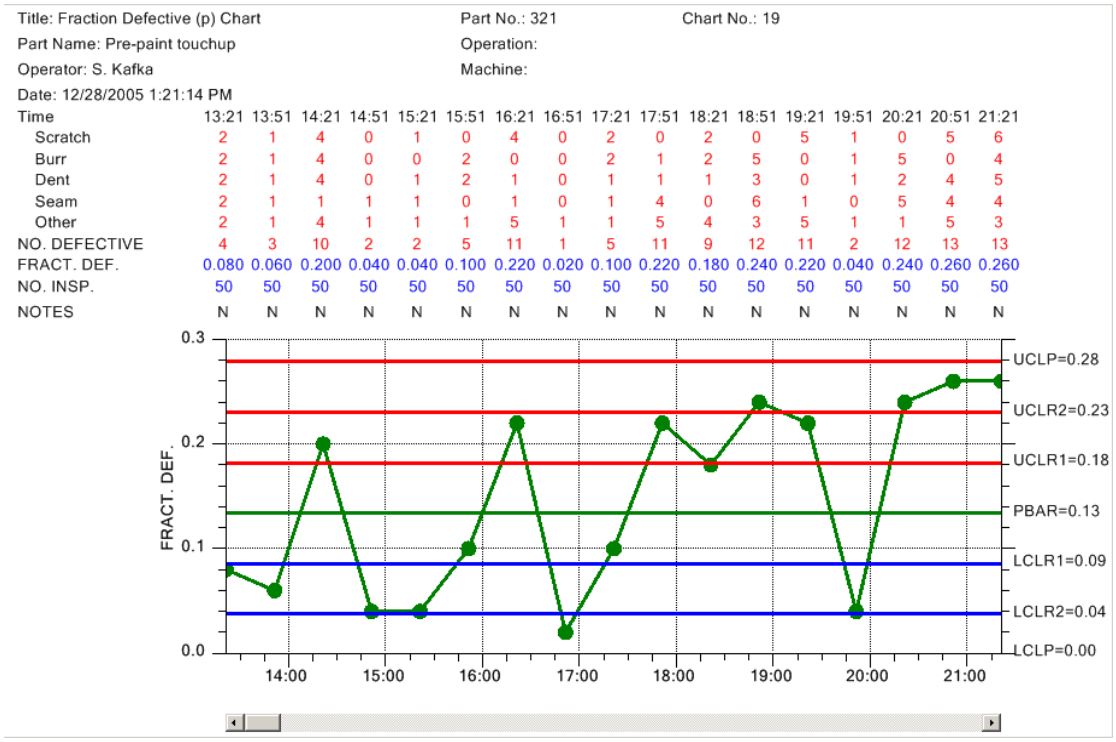
Multiple SPC Control Limits

The normal SPC control limit displays at the 3-sigma level, both high and low. A common standard is that if the process variable under observation falls outside of the +-3-sigma limits the process is out of control. The default setup of our variable control charts have a high limit at the +3-sigma level, a low limit at the -3-sigma level, and a target value. There are situations where the quality engineer also wants to display control limits at the 1-sigma and 2-sigma level. The operator might receive some sort of preliminary warning if the process variable exceeds a 2-sigma limit.

.

You are able to add additional control limit lines to an attribute control chart, as in the example MultLimitCharts.`BuildMultiLimitPChart`

.



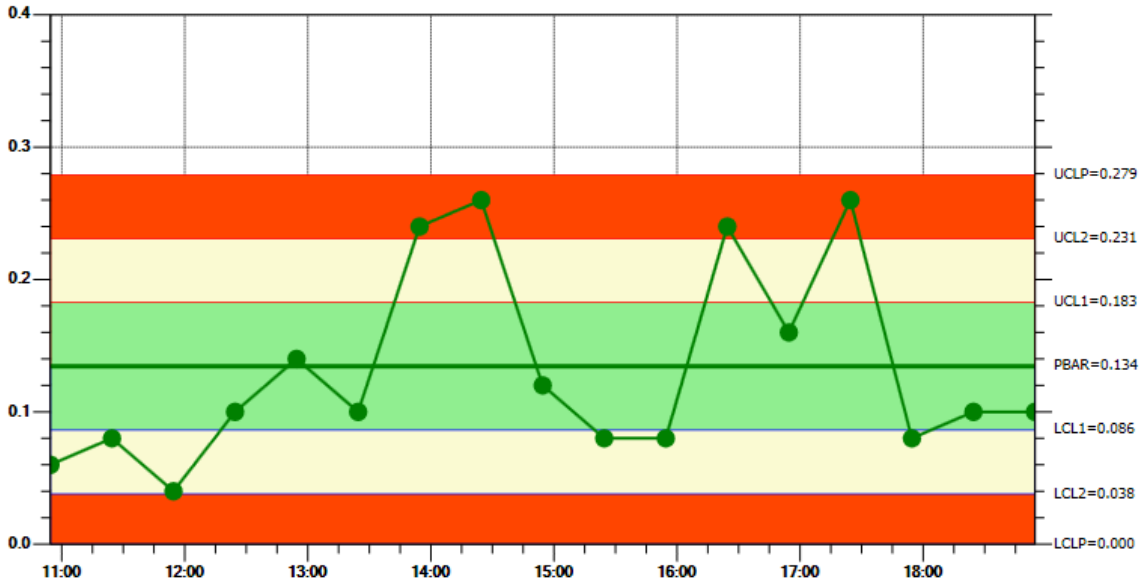
We added a method (**add3SigmaControlLimits**) which will generate multiple control limits, for +-1, 2, and 3- sigma levels, based on an initial specification of the target value, and the +-3 sigma control limits. This is most useful if you want to generate +-1, 2 and 3-sigma control limits in order to fill in between them with a zone fill color. If you call the **autoCalculateControlLimits** method, the initial +-1,2 and 3-sigma control limit values will be altered to the new, calculated values, but the control limit lines remain, with their new values. Since you do not normally want to be generating alarm messages for excursions into the +-1 and 2-sigma limit areas, the **add3SigmaControl** limits has the option of disabling alarm notification in the case of +-1 and +-2 alarm conditions.

[JavaScript]

```
var target = 0.14;
var lowlim  = 0;
var highlim  = 0.28;
var limitcheck = false;

primarychart.add3SigmaControlLimits(target, lowlim, highlim, limitcheck);
primarychart.setControlLimitLineFillMode (true);
```

[TypeScript]

```
let target: number = 0.14;
let lowlim: number  = 0;
let highlim: number  = 0.28;
let limitcheck: boolean = false;

primarychart.add3SigmaControlLimits(target, lowlim, highlim, limitcheck);
primarychart.setControlLimitLineFillMode (true);
```



Control Limit Fill Option used with +-1, 2 and 3-sigma control limits

Special Note - We view the technique described below, for adding additional control limits, much too complicated. We include it here only because it was included in the manuals for the other versions of QCSPCChart.

You can also add additional control limits one at a time. By default you get the +-3-sigma control limits. So additional control limits should be considered +-2-sigma and +-1-sigma control limits. Do not confuse control limits with specification limits, which must be added using the **addSpecLimit** method. There are two steps to adding additional control limits: creating a **SPCControlLimitRecord** object for the new control limit, and adding the control limit to the chart using the charts **addAdditionalControlLimit** method**.** It is critical that you add them in a specific order, that order being:

Primary Chart	SPC_LOWER_CONTROL_LIMIT_2	(2-sigma lower limit)
Primary Chart	SPC_UPPER_CONTROL_LIMIT_2	(2-sigma upper limit)
Primary Chart	SPC_LOWER_CONTROL_LIMIT_1	(1-sigma lower limit)
Primary Chart	SPC_UPPER_CONTROL_LIMIT_1	(1-sigma upper limit)

[JavaScript]

```
//target control limit primary chart
 var chartdata = attribchart.getChartData();

 var primarytarget =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_CONTR
OL_TARGET);
      primarytarget.setControlLimitValue ( 0.14);
      primarytarget.setControlLimitText("Target");

      //lower control limit primary chart
      var primarylowercontrollimit =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_LOWER
_CONTROL_LIMIT);
      primarylowercontrollimit.setControlLimitValue(0);
      primarylowercontrollimit.setControlLimitText("LCL");

      //upper control limit primary chart
      var primaryuppercontrollimit =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_UPPER
_CONTROL_LIMIT);
      primaryuppercontrollimit.setControlLimitValue( 0.28);
      primaryuppercontrollimit.setControlLimitText( "UCL");
```

[TypeScript]

```
//target control limit primary chart
 let chartdata : QCSPCChartTS.SPCControlChartData | null =
attribchart.getChartData();

 let primarytarget: QCSPCChartTS.SPCControlLimitRecord =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_CONTR
OL_TARGET);
      primarytarget.setControlLimitValue ( 0.14);
      primarytarget.setControlLimitText("Target");

      //lower control limit primary chart
let primarylowercontrollimit: QCSPCChartTS.SPCControlLimitRecord =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_LOWER
_CONTROL_LIMIT);
      primarylowercontrollimit.setControlLimitValue(0);
      primarylowercontrollimit.setControlLimitText("LCL");

      //upper control limit primary chart
let primaryuppercontrollimit: QCSPCChartTS.SPCControlLimitRecord =
chartdata.getControlLimitRecord(QCSPCChartTS.SPCControlChartData.SPC_PRIMARY_UPPER
_CONTROL_LIMIT);
      primaryuppercontrollimit.setControlLimitValue( 0.28);
      primaryuppercontrollimit.setControlLimitText( "UCL");
```

Special Note – When you create a **SPCControlLimitRecord** object, you can specify an actual limit level. If you do not call the charts **autoCalculateControlLimits** method, the control limit will be displayed at that value. If you do call **autoCalculateControlLimits**

method, the auto-calculated value overrides the initial value (0.0 in the examples above). When you call the charts **addAdditionalControlLimits** method, you specify the sigma level that is used by the **autoCalculateControlLimits** to calculate the control limit level.

If you want the control limits displayed as filled areas, set the charts **ControlLimitLineFillMode** property true.

```
primarychart.setControlLimitLineFillMode(true);
```

This will fill each control limit line from the limit line to the target value of the chart. In order for the fill to work properly, you must set this property after you define all additional control limits. Also, you must add the outer most control limits (SPC_UPPER_CONTROL_LIMIT_3 and SPC_LOWER_CONTROL_LIMIT_3) first, followed by the next outer most limits (SPC_UPPER_CONTROL_LIMIT_2 and SPC_LOWER_CONTROL_LIMIT_2), followed by the inner most control limits (SPC_UPPER_CONTROL_LIMIT_1 and SPC_LOWER_CONTROL_LIMIT_1). This way the fill of the inner limits will partially cover the fill of the outer limits, creating the familiar striped look you want to see.

Chart Y-Scale

You can set the minimum and maximum values of the two charts y-scales manually using the **PrimaryChart.MinY** and **PrimaryChart.MaxY** properties.

[JavaScript]

```
// Set initial scale of the y-axis of the mean chart
// If you are calling autoScalePrimaryChartYRange this isn't really needed
primarychart.setMinY( 0);
primarychart.setMaxY( 1.0);
```

[TypeScript]

```
// Set initial scale of the y-axis of the mean chart
// If you are calling autoScalePrimaryChartYRange this isn't really needed
primarychart.setMinY( 0);
primarychart.setMaxY( 1.0);
```

It is easiest to just call the auto-scale routines after the chart has been initialized with data, and any control limits calculated.

[JavaScript]

```
// Must have data loaded before any of the Auto.. methods are called
    SimulateData(attribchart, numssampleintervals, fractiondefective,
subgroupsize);
```

```
// Calculate the SPC control limits for both graphs of the current SPC chart
    attribchart.autoCalculateControlLimits();

// Scale the y-axis of the X-Bar chart to display all data and control limits
    attribchart.autoScalePrimaryChartYRange();
```

[TypeScript]

```
// Must have data loaded before any of the Auto.. methods are called
    this.SimulateData(attribchart, numssampleintervals, fractiondefective,
subgroupsize);

// Calculate the SPC control limits for both graphs of the current SPC chart
    attribchart.autoCalculateControlLimits();

// Scale the y-axis of the X-Bar chart to display all data and control limits
    attribchart.autoScalePrimaryChartYRange();
```

Once all of the graph parameters are set, call the method **rebuildChartUsingCurrentData**.

```
// Rebuild the chart using the current data and settings
attribchart.rebuildChartUsingCurrentData();
```

If, at any future time you change any of the chart properties, you will need to call **rebuildChartUsingCurrentData** to force a rebuild of the chart, taking into account the current properties. **rebuildChartUsingCurrentData** invalidates the chart and forces a redraw**.** Our examples that update dynamically demonstrate this technique. The chart is setup with some initial settings and data values. As data is added in real-time to the graph, the chart SPC limits, and y-scales are constantly recalculated to take into account new data values. The code below is Extracted from the **RealTimeSPCChartUpdates.BuildRealTimeAttribSPCChart** example.

[JavaScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';

    var timerId = 0;
    var spcrtvarchart = null;
    var spcrtattribchart = null;
    var ontopspcchart = null;
    var timercount = 0;
    var timercountlimit = 0;



    export async function BuildRealTimeAttribSPCChart(canvasid) {

        var htmlcanvas = document.getElementById(canvasid);
```

```
        var spccharttype =
QCSPCChartTS.SPCControlChartData.FRACTION_DEFECTIVE_PARTS_CHART;
        var subgroupsize = 100;
        var numberpointsinview = 12;
        var charttitle = " p-Chart (Fraction)";


        var attribchart =
QCSPCChartTS.SPCBatchAttributeControlChart.newSPCBatchAttributeControlChartChartTy
peSubgroupSize(htmlcanvas, spccharttype, 1, subgroupsize, numberpointsinview);
        if (!attribchart) return;

        ontopspcchart = spcrtattribchart = attribchart;

    .
    .


    }

    function TimerTickProc()
    {
        var numssampleintervals = 1;
        var chartmean = 30;
        var chartsigma = 5;
        var fractiondefective = 0.134;
        var subgroupsize = 100;

         if (spcrtattribchart)
         {
            SimulateAttribData(spcrtattribchart, numssampleintervals,
fractiondefective, subgroupsize);
            if (ontopspcchart == spcrtattribchart)
              spcrtattribchart.rebuildChartUsingCurrentData();
         }
         timercount++;
         if (timercount > timercountlimit )
            StopTimer();
    }

    export async function StartTimer (milliseconds, count)
    {
       if (timerId != 0)
         StopTimer();
       timercountlimit = count;
       timerId = setInterval(() =>
       TimerTickProc(),
       milliseconds);
    }

    export async function  StopTimer ()
    {
        clearInterval(timerId);
        timerId = 0;
    }



function SimulateAttribData(spcchart, count, fractiondefective,  subgroupsize) {
        // batch number for a given sample subgroup
        var batchCounter = 0;
        var i = 0;
        var timestamp = new Date();
        if (!spcchart) return;
        var chartdata= spcchart.getChartData();
        if (!chartdata) return;
        var currentcount = chartdata.getCurrentNumberRecords();
        var charttype = spcchart.getSPCChartType();
```

```
        if (currentcount > 0) // start date at the previous ending date plut time
increment
        {
            var ts  = chartdata.getTimeValue(currentcount-1);
            timestamp = ts;
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }
        for (i = 0; i < count; i++) {

            var samples = chartdata.simulateDefectRecordMeanType(subgroupsize *
fractiondefective, charttype);
            batchCounter = i + currentcount;
            // Add a new sample record
            chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, "");

            // Simulate passage of timeincrementminutes minutes
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }

    }
```

[TypeScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';

export class RealTimeSPCChartUpdate {

    timerId: number = 0;
    spcrtvarchart: QCSPCChartTS.SPCChartBase | null = null;
    spcrtattribchart: QCSPCChartTS.SPCChartBase | null = null;
    ontopspcchart: QCSPCChartTS.SPCChartBase | null = null;
    timercount: number = 0;
    timercountlimit: number = 0;

    public constructor() {

    }

  public async  BuildRealTimeAttribSPCChart(canvasid: string) {

        let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
        let spccharttype: number =
QCSPCChartTS.SPCControlChartData.FRACTION_DEFECTIVE_PARTS_CHART;
        let subgroupsize: number = 100;
        let numberpointsinview: number = 12;
        let charttitle: string = " p-Chart (Fraction)";


        let attribchart: QCSPCChartTS.SPCChartBase =
QCSPCChartTS.SPCBatchAttributeControlChart.newSPCBatchAttributeControlChartChartTy
peSubgroupSize(htmlcanvas, spccharttype, 1, subgroupsize, numberpointsinview);
        if (!attribchart) return;
        .
.
.

    }

  TimerTickProc()
    {
        let numssampleintervals: number = 1;
```

```
        let fractiondefective: number = 0.134;
        let subgroupsize: number = 100;
         if (this.spcrtattribchart)
         {
            this.SimulateAttribData(this.spcrtattribchart, numssampleintervals,
fractiondefective, subgroupsize);
            if (this.ontopspcchart == this.spcrtattribchart)
              this.spcrtattribchart.rebuildChartUsingCurrentData();
         }
         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;
    }

  SimulateAttribData(spcchart: QCSPCChartTS.SPCChartBase, count: number,
fractiondefective: number,  subgroupsize: number) {
        // batch number for a given sample subgroup
        let batchCounter: number = 0;
        let i: number = 0;
        let timestamp: Date = new Date();
        if (!spcchart) return;
        let chartdata: QCSPCChartTS.SPCControlChartData | null =
spcchart.getChartData();
        if (!chartdata) return;
        let currentcount: number = chartdata.getCurrentNumberRecords();
        let charttype: number = spcchart.getSPCChartType();
        if (currentcount > 0) // start date at the previous ending date plut time
increment
        {
            let ts : Date = chartdata.getTimeValue(currentcount-1);
            timestamp = ts;
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }
        for (i = 0; i < count; i++) {

            let samples: QCSPCChartTS.DoubleArray =
chartdata.simulateDefectRecordMeanType(subgroupsize * fractiondefective,
charttype);
            batchCounter = i + currentcount;
            // Add a new sample record
            chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, "");

            // Simulate passage of timeincrementminutes minutes
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }

    }

}
```

Updating Chart Data

The real-time example above demonstrates how the SPC chart data is updated, using the **chartdata.addNewSampleRecord** method. In this case, the chart data updates with each timer tick event, though it could just as easily be any other type of event. If you have already collected all of your data and just want to plot it all at once, use a simple loop like most of our examples do to update the data.

[JavaScript]

```
function SimulateData(spcchart, count, fractiondefective,  subgroupsize) {
    // batch number for a given sample subgroup
    var batchCounter = 0;
    var i;
    var timestamp = new Date();
    if (!spcchart) return;
    if (!spcchart.getChartData()) return;
    var charttype = spcchart.getSPCChartType();
    var currentcount = spcchart.getChartData().getCurrentNumberRecords();
    if (currentcount > 0) // start date at the previous ending date plut time
increment
    {
        var ts  = chartdata.getTimeValue(currentcount-1);
        timestamp = ts;
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }
    for (i = 0; i < count; i++) {

    var samples =
spcchart.getChartData().simulateDefectRecordMeanType(subgroupsize *
fractiondefective, charttype);
        batchCounter = currentcount + i;
 // Add a new sample record
       spcchart.getChartData().addNewSampleRecordBatchNumberDateSamplesNotes
(batchCounter, timestamp, samples,"");

        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }

}
```

[TypeScript]

```
  SimulateData(spcchart: QCSPCChartTS.SPCChartBase, count: number,
fractiondefective: number,  subgroupsize: number) {
        // batch number for a given sample subgroup
        let batchCounter: number = 0;
        let i: number = 0;
        let timestamp: Date = new Date();
        if (!spcchart) return;
        let chartdata: QCSPCChartTS.SPCControlChartData | null =
spcchart.getChartData();
        if (!chartdata) return;
        let currentcount: number = chartdata.getCurrentNumberRecords();
        let charttype: number = spcchart.getSPCChartType();
        if (currentcount > 0) // start date at the previous ending date plut time
increment
        {
```

```
            let ts : Date = chartdata.getTimeValue(currentcount-1);
            timestamp = ts;
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }
        for (i = 0; i < count; i++) {

            let samples: QCSPCChartTS.DoubleArray =
chartdata.simulateDefectRecordMeanType(subgroupsize * fractiondefective,
charttype);
            batchCounter = i + currentcount;
            // Add a new sample record
            chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, "");

            // Simulate passage of timeincrementminutes minutes
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }

    }
```

In this example the sample data and the time stamp for each sample record is simulated. In your application, you will probably be reading the sample record values from some sort of database or file, along with the actual time stamp for that data.

If you want to append a text note to a sample record, use one of the **chartdata.addNewSampleRecord...** methods that have a *notes* parameter. The code below is extracted from the AttributeControlCharts example.

[JavaScript]

```
function SimulateData(spcchart, count, fractiondefective,  subgroupsize) {
    // batch number for a given sample subgroup
    var batchCounter = 0;
    var i;
    var timestamp = new Date();
    var notesstring = "";
    if (!spcchart) return;
    if (!spcchart.getChartData()) return;
    var charttype = spcchart.getSPCChartType();
    var currentcount = spcchart.getChartData().getCurrentNumberRecords();
    if (currentcount > 0) // start date at the previous ending date plut time
increment
    {
        var ts  = chartdata.getTimeValue(currentcount-1);
        timestamp = ts;
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }
    for (i = 0; i < count; i++) {

    var samples =
spcchart.getChartData().simulateDefectRecordMeanType(subgroupsize *
fractiondefective, charttype);
        batchCounter = currentcount + i;
       var r = QCSPCChartTS.ChartSupport.getRandomDouble();
       if (r < 0.1) // make a note on every tenth item, on average
           notesstring = "Note for sample subgroup #" + batchCounter.toString() +
               ". Spray paint nozzel clogged. Replaced with new, Enois nozzle.";
       else
```

```
            notesstring = "";
 // Add a new sample record
       spcchart.getChartData().addNewSampleRecordBatchNumberDateSamplesNotes
(batchCounter, timestamp, samples,notesstring);

        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }

}
```

[TypeScript]

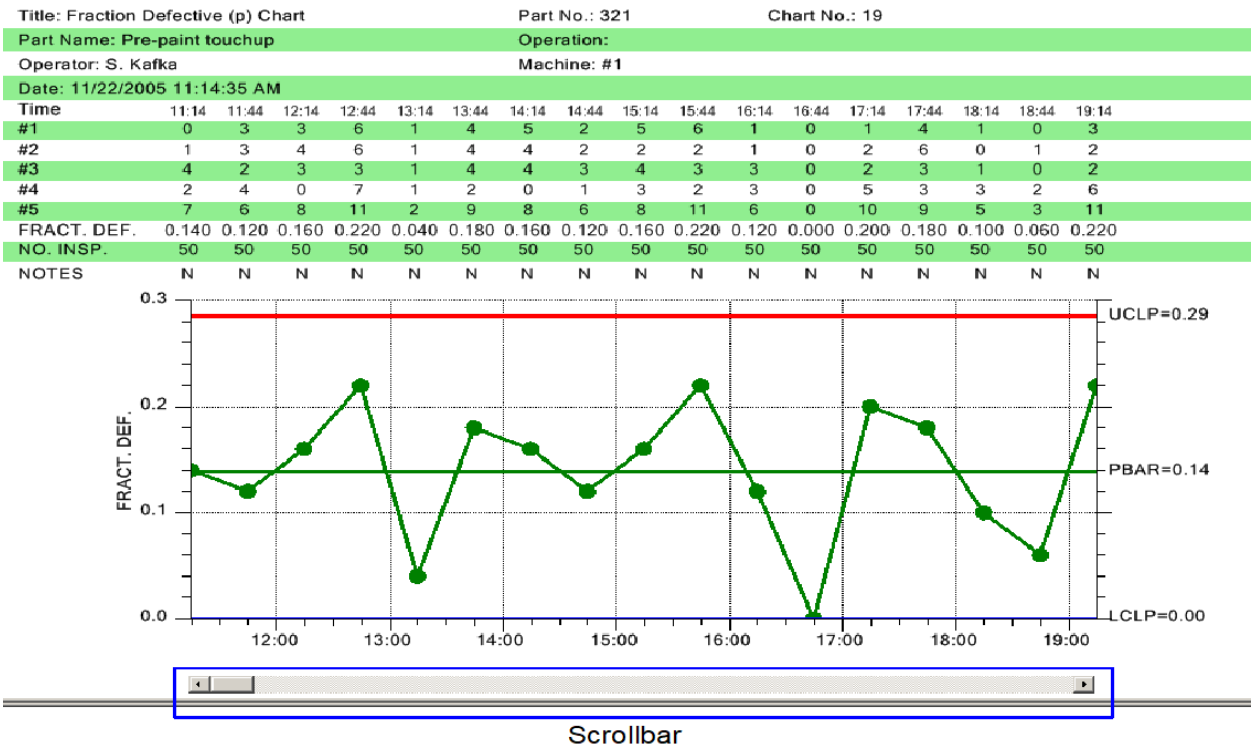
```
  SimulateData(spcchart: QCSPCChartTS.SPCChartBase, count: number,
fractiondefective: number,  subgroupsize: number) {
        // batch number for a given sample subgroup
        let batchCounter: number = 0;
        let i: number = 0;
        let timestamp: Date = new Date();
        let notesstring: string = "";
        if (!spcchart) return;
        let chartdata: QCSPCChartTS.SPCControlChartData | null =
spcchart.getChartData();
        if (!chartdata) return;
        let currentcount: number = chartdata.getCurrentNumberRecords();
        let charttype: number = spcchart.getSPCChartType();
        if (currentcount > 0) // start date at the previous ending date plut time
increment
        {
            let ts : Date = chartdata.getTimeValue(currentcount-1);
            timestamp = ts;
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }
        for (i = 0; i < count; i++) {

            let samples: QCSPCChartTS.DoubleArray =
chartdata.simulateDefectRecordMeanType(subgroupsize * fractiondefective,
charttype);
            batchCounter = i + currentcount;
       let r: number = QCSPCChartTS.ChartSupport.getRandomDouble();
       if (r < 0.1) // make a note on every tenth item, on average
           notesstring = "Note for sample subgroup #" + batchCounter.toString() +
               ". Spray paint nozzel clogged. Replaced with new, Enois nozzle.";
       else
           notesstring = "";
            // Add a new sample record
            chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, "");

            // Simulate passage of timeincrementminutes minutes
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }

    }
```

Scatter Plots of the Actual Sampled Data

- This option is not applicable for attribute control charts.

Enable Chart ScrollBar

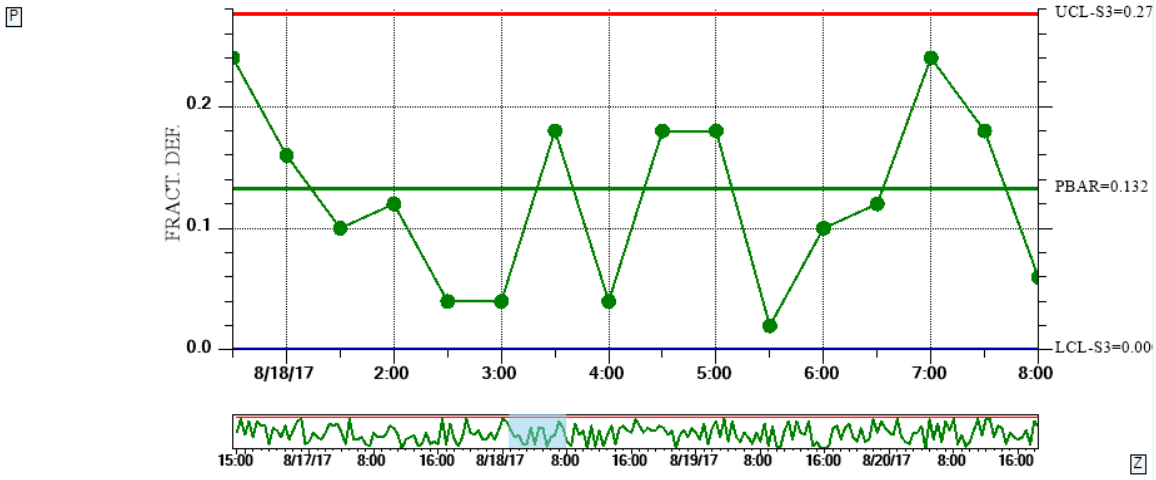
Set the **EnableScrollBar** property true to enable the chart scrollbar. You will then be able to window in on 8-20 sample subgroups at a time, from a much larger collection of measurement data representing hundreds or even thousands of subgroups, and use the scrollbar to move through the data, similar to paging through a spreadsheet.



Zooming as an option for the scrollbar

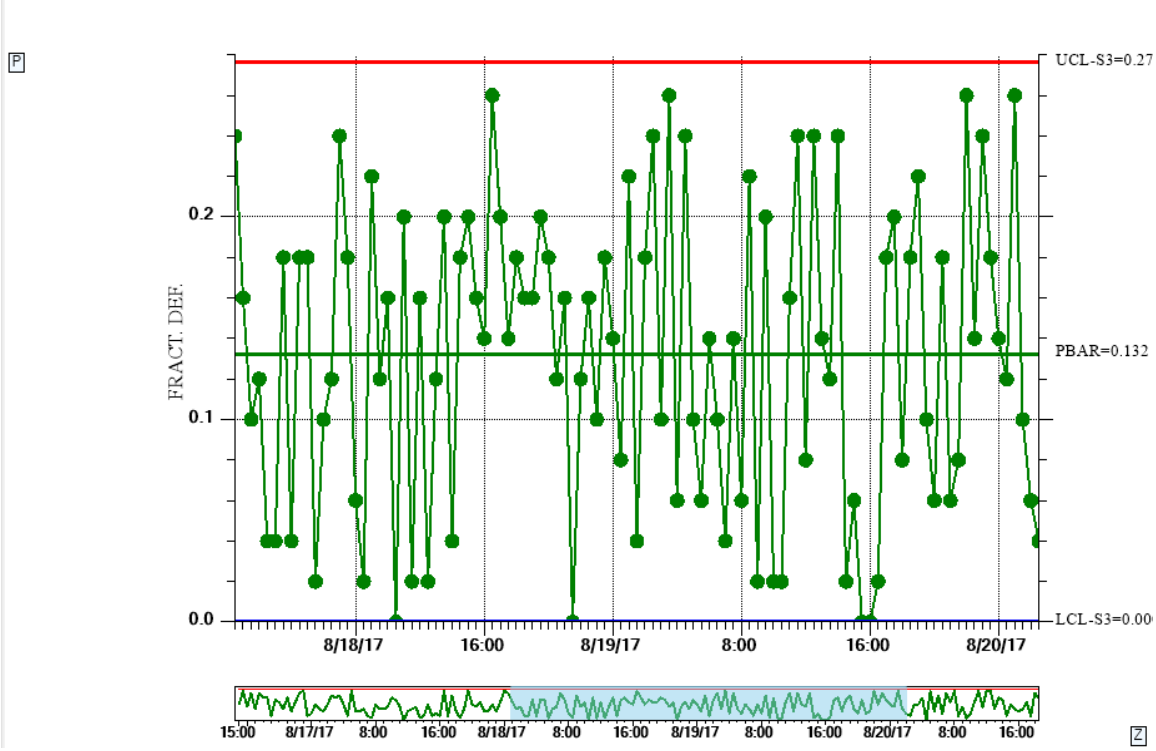
The horizontal scrollbar can be replaced (toggled on/off actually) using the UI, with a zoom window, by clicking on the z-button in the lower right corner. The user will be able to scroll the charts by dragging the transparent blue zoom window left or right, or expand the x-axis scale by dragging the left or right edge of the zoom window.

Title: Fraction Defective (p) Chart					Part No.: 321					Chart No.: 19						
Part Name: Pre-paint touchup					Operation:											
Operator:K. Peterson					Machine:											
Date: 8/16/2017 3:07:47 PM																
TIME	0:30	1:00	1:30	2:00	2:30	3:00	3:30	4:00	4:30	5:00	5:30	6:00	6:30	7:00	7:30	8:00
FRACT. DEF.	0.240	0.160	0.100	0.120	0.040	0.040	0.180	0.040	0.180	0.180	0.020	0.100	0.120	0.240	0.180	0.060
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
ALARM	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Notes	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N



Click on the Z button in the lower right corner and a zoom window will replace the scrollbar. Use the mouse to scroll the charts by dragging the transparent blue zoom window left or right, or expand the x-axis scale by dragging the left or right edge of the zoom window.

If the number of points in the display exceeds the initial setup value for the number of data points, the table is temporarily removed to prevent overlap of the table columns. If the number of data points is reduced to <= the initial number of points, the table will reappear.



If you use the zoom window to display a lot of points, the table at the top will disappear to prevent the table columns from overlapping.

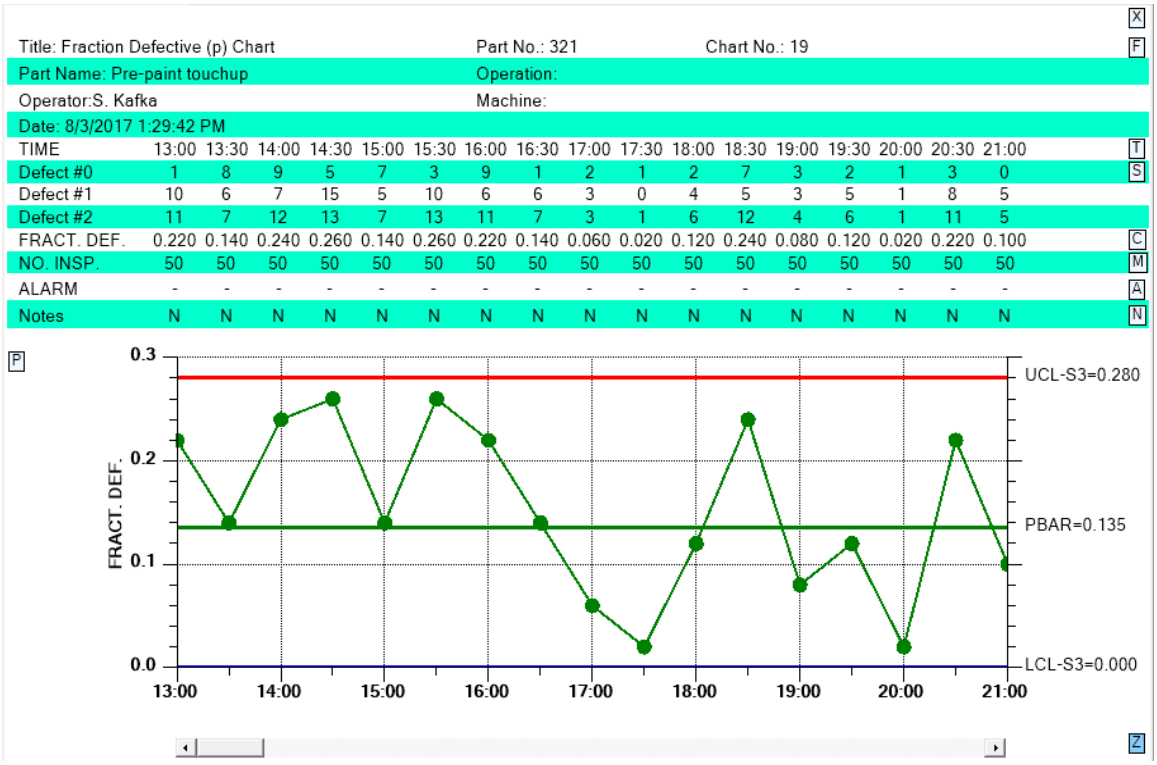
The default display display for all chart types uses the scrollbar. The zoom option is seen as a small button with the character 'Z' in the lower right corner of the display. You enter/exit the zoom mode by clicking on that button. If you do not want the button to show at all, effectively making the zoom option inaccessible to the end user, set EnableZoomToggles property false.

.

```
attribchart.setEnableZoomToggles(false);
```

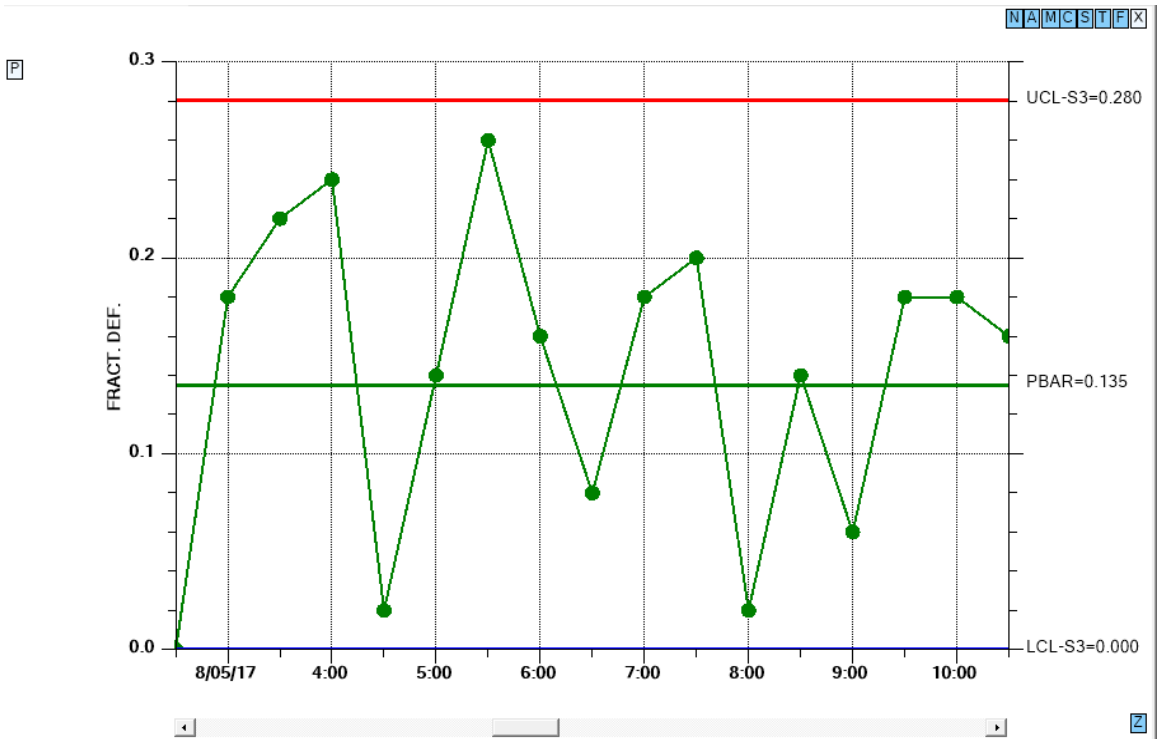
Collapsible Items

Like the Zoom option described in the previous section, there are also UI buttons which can toggle on and off rows in the table, and the Primary and Secondary charts. Like the Zoom button, these UI buttons can be hidden from the end user if you don't want them to show. See the end of this section for examples. On the top and right of the table buttons will selectively collapse/restore rows of the table, freeing up more space for charts. Buttons to the left and bottom of the Primary and Secondary charts also permit the user to collapse/restore either of those charts, allowing the remaining chart to display in all of the remaining space.



Buttons on the top, right and left of the display permit the user to selectively collapse portions of the chart.

Click on the N, A, M, P C and S buttons and shrink the table to following size.



In the example above, all of the chart options except for the Primary chart, have been toggled off using the buttons.

The buttons at the right of the table (and at the top right if toggled off) use the following ID's.

X	Turn on/off all table items at once
F	Turn on/off form data
T	Turn on/off sample interval time stamp data
S	Turn on/off sample value data
C	Turn on/off calculated value (mean, range, sum, etc.) data
P	Turn on/off process capability data
M	Turn on/off number of samples data
A	Turn on/off alarm data
N	Turn on/off notes data
Z	Bottom right - Turn on/off zoom control – the only button not affected by the X button above

The buttons at the left of the primary and secondary charts use the following ID's.

P	Turn on/off the Primary chart

S Turn on/off the Secondary chart

If you do not want the buttons to show at all, effectively making these UI options inaccessible to the end user, set these properties false.

.

Hide the chart buttons on the left.

```
attribchart.setEnableChartToggles(false);
```

Hide the table buttons on the right.

```
attribchart.setEnableTableRowToggles(false);
```

You can hide ALL of the UI buttons (zoom, chart, and table) using the property EnableDisplayOptionToggles. This is what you use if you do not want the end user to have any control over the display of the table and chart.

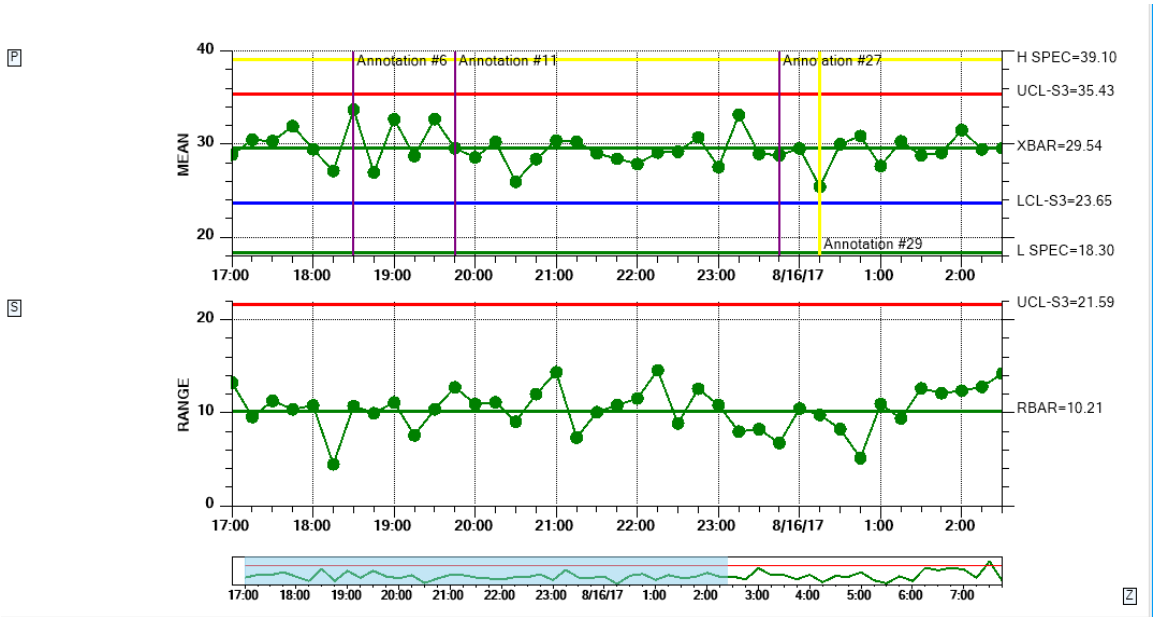
```
attribchart.setEnableDisplayOptionToggles(false);
```

If you want to selectively enable options, first set EnableDisplayOptionToggles true. The other button display enables won't work unless this option is true. Then disable the options you do not want to show. In the example below, only the zoom option is left visible.

```
attribchart.setEnableDisplayOptionToggles (true);
attribchart.setEnableTableRowToggles(false);
attribchart.setEnableChartToggles(false)
attribchart.setEnableZoomToggle(true);
```

Enhanced Annotations

The chart annotations have been enhanced with a vertical line with accompanying text using programmer specified justification.



The enhanced chart annotations include a vertical line to mark the data point, and many justification options (top, middle, bottom, left, right and center).

The new version of addAnnotation is just an override of the original, with added parameters to specify the vertical line attributes and the text justification.

addAnnotation

Add an annotation to a data point in the specified SPC chart.

```
public addAnnotationChartNumIndexString(chart: number, datapointindex: number,
text: string): number

public addAnnotationChartNumIndexChartText(chart: number, datapointindex: number,
textobj: ChartText): number

public addAnnotationChartNumXDateYNumChartText(chart: number, x: Date, y: number,
textobj: ChartText): number

public addAnnotationChartNumIndexStringJustAttrib(chart: number, datapointindex:
number, text: string, just: number, attrib: ChartAttribute): number

public addAnnotationChartNumStringJustAttrib(chart: number, text: string, just:
number, attrib: ChartAttribute): number
```

where:

chart Specifies whether the annotation is added to the primary, or secondary chart. Use one of the SPChartObjects constants: SPCChartObjects.PRIMARY_CHART or SPCChartObjects.SECONDARY_CHART.

datapointindex The index of the data point the annotation is for.

text A string string representing the annotation.

just The justification for the text to the x-position of the annotation. Use one of the annotation justification constants:
ANNOTATION_UPPER_RIGHT, ANNOTATION_UPPER_LEFT,
ANNOTATION_LOWER_RIGHT, ANNOTATION_LOWER_LEFT,
ANNOTATION_UPPER_CENTER,
ANNOTATION_LOWER_CENTER,
ANNOTATION_DATAPOINT_RIGHT,
ANNOTATION_DATAPOINT_LEFT

attrib A the attribute of the vertical line.

You call the addAnnotation method immediately after the addNewSampleRecord.. method call. For Example:

[JavaScript]

```
                // Add the new sample subgroup to the chart
      chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, notesstring);

      var index = chartdata.getCurrentNumberRecords() - 1;
      var annotstring = "Annotation #" + index.toString();
      var lineattrib =
QCSPCChartTS.ChartAttribute.newChartAttribute3(QCSPCChartTS.ChartColor.PURPLE, 2,
               QCSPCChartTS.ChartConstants.LS_SOLID);

      // Adjust justification to minimize overlap of adjacent annotations
      var annotjust = QCSPCChartTS.SPCAnnotation.ANNOTATION_UPPER_LEFT;

      attribchart.addAnnotation(QCSPCChartTS.SPCChartObjects.PRIMARY_CHART, index,
        annotstring, annotjust, lineattrib);
```

[TypeScript]

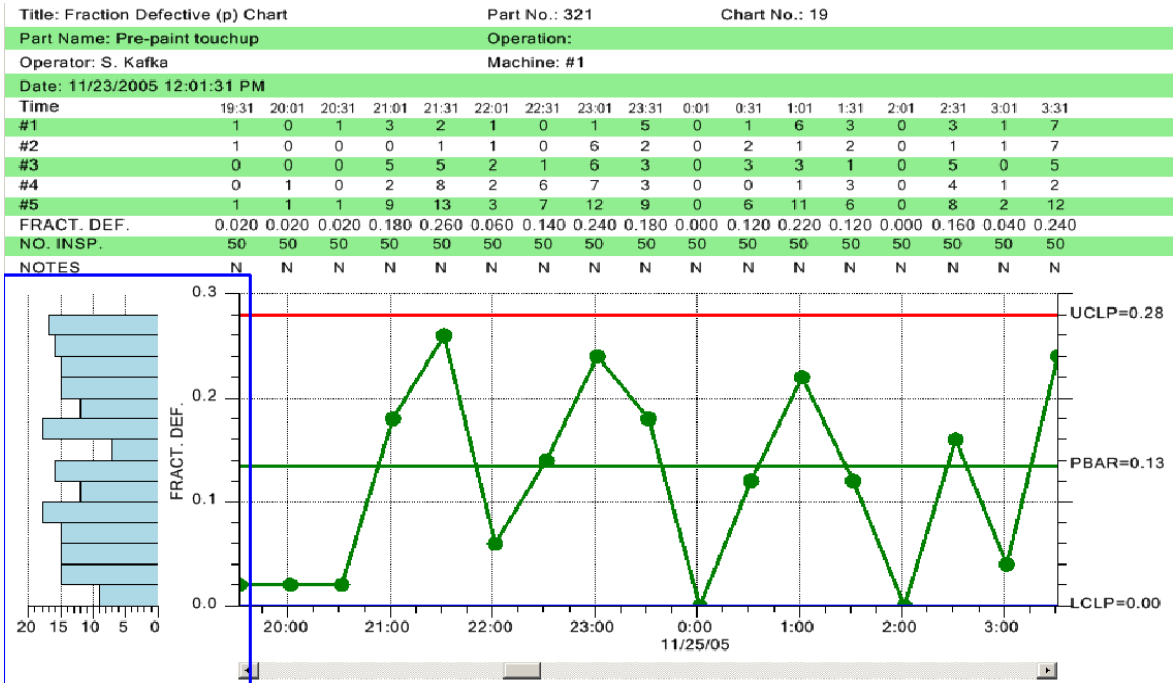
```
                // Add the new sample subgroup to the chart
      chartdata.addNewSampleRecordBatchNumberDateSamplesNotes (timestamp, samples,
notesstring);

      let index: number = chartdata.getCurrentNumberRecords() - 1;
      let annotstring: string = "Annotation #" + index.toString();
      let lineattrib: QCSPCChartTS.ChartAttribute =
QCSPCChartTS.ChartAttribute.newChartAttribute3(QCSPCChartTS.ChartColor.PURPLE, 2,
               QCSPCChartTS.ChartConstants.LS_SOLID);

      // Adjust justification to minimize overlap of adjacent annotations
      let annotjust: number = QCSPCChartTS.SPCAnnotation.ANNOTATION_UPPER_LEFT;

      attribchart.addAnnotation(QCSPCChartTS.SPCChartObjects.PRIMARY_CHART, index,
        annotstring, annotjust, lineattrib);
```

SPC Chart Histograms

Viewing frequency histograms of the variation in the primary variable side-by-side with the SPC control chart makes it even easier to find out whether the variations are the result of natural variations or the result of some systemic change in the process. You can turn on integrated frequency histograms for either chart using the **PrimaryChart.DisplayFrequencyHistogram** property of the chart.

[JavaScript / TypeScript]

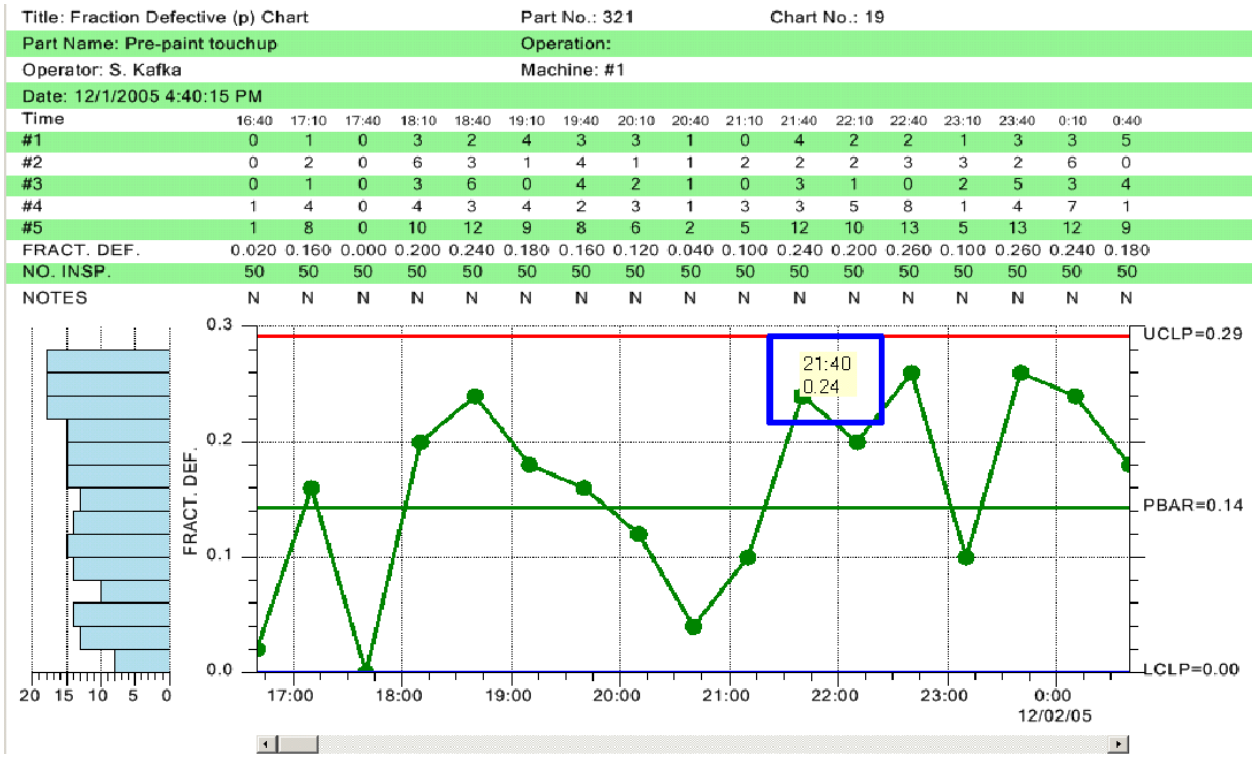
```
primarychart.setDisplayFrequencyHistogram( true);
```



SPC Chart Data and Notes Tooltips

You can invoke two types of tooltips using the mouse. The first is a data tooltip. When you hold the mouse button down over one of the data points in the primary chart, the x and y values for that data point display in a popup tooltip..

Data Tooltip



In the default mode, the data tooltip displays the x,y value of the data point nearest the mouse click. If the x-axis is a time axis then the x-value is displayed as a time stamp; otherwise, it is displayed as a simple numeric value, as is the y-value. You can optionally display subgroup information (sample values, calculated values, process capability values and notes) in the data tooltip window, under the x,y value, using enable flags in the primary charts tooltip property.

Extracted from the MultiDivCharts.BuildXBarRChart example.

[JavaScript]

```
if (primarychart)
{
    var datatooltip = primarychart.getDatatooltip();
    if (datatooltip)
    {
      datatooltip.setEnableCategoryValues( true);
      datatooltip.setEnableProcessCapabilityValues( true);
      datatooltip.setEnableCalculatedValues(true);
      datatooltip.setEnableNotesString ( true);
    }
}
```

[TypeScript]

```
let primarychart: QCSPCChartTS.SPCChartObjects | null =
xbarrchart.getPrimaryChart();
if (primarychart)
{
    let datatooltip: QCSPCChartTS.SPCDataToolTip | null =
primarychart.getDatatooltip();
    if (datatooltip)
    {
      datatooltip.setEnableCategoryValues( true);
      datatooltip.setEnableProcessCapabilityValues( true);
      datatooltip.setEnableCalculatedValues(true);
      datatooltip.setEnableNotesString ( true);
    }
}
```

where

setEnableCategoryValues
setEnableProcessCapabilityValues
setEnableCalculatedValues
setEnableNotesStrings

Display the category (subgroup sample values) in the data tooltip.

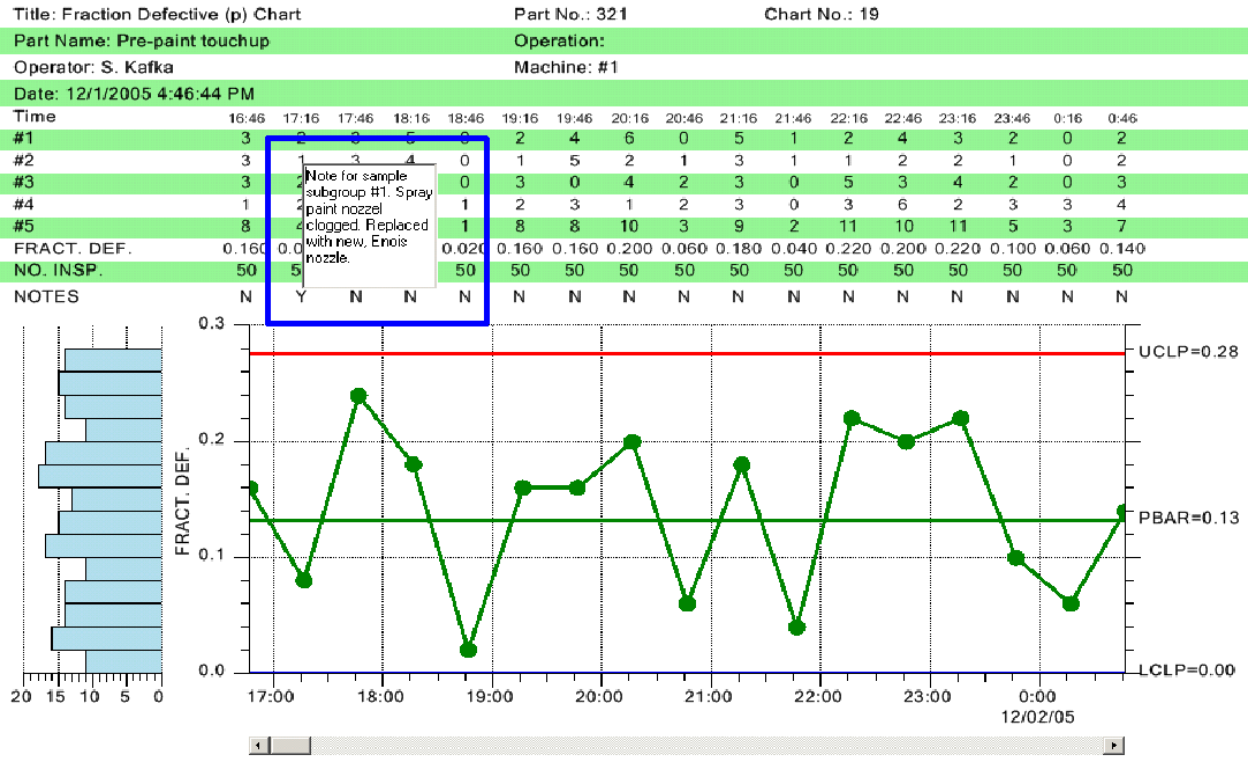
```
datatooltip.setEnableCategoryValues(true);
```

Display the current notes string for the sample subgroup.

```
datatooltip.setEnableCategoryValues(true);
```

If you are displaying the Notes line in the table portion of the chart, the Notes entry for a sample subgroup will display “Y” if a note was recorded for that sample subgroup, or “N” if no note was recorded. Notes are recorded using one of the **chartdata.addNewSampleRecord...** methods that include a notes parameter, or by using the **chartdata.setNotes**, or **chartdata.appendNotes** methods. See the section *Updating Chart Data*. If you click on a “Y” in the Notes row for a sample subgroup, the complete text of the note for that sample subgroup will display in a edit text box, immediately above the “Y”. You can actually edit the notes in the edit text box.

Notes Tooltip



[JavaScript]

```
function SimulateData(spcchart, count, mean, sigma) {
    // batch number for a given sample subgroup
    var batchCounter = 0;
    var i;
    var timestamp = new Date();
    if (!spcchart) return;
    if (!spcchart.getChartData()) return;
    var da = new QCSPCChartTS.DoubleArray();

    var currentcount = spcchart.getChartData().getCurrentNumberRecords();
    for (i = 0; i < count; i++) {
        // Simulate a sample subgroup record
        var samples =
spcchart.getChartData().simulateMeasurementRecordMeanRange(mean, sigma);
        // Update chart data using i as the batch number
        batchCounter = currentcount + i;
        var note = "";
        if ((i % 5) == 0) note = "This is a note";
        // Add a new sample record to the chart data

spcchart.getChartData().addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter
, timestamp, samples, note);
        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }
```

```
}
```

[TypeScript]

```
SimulateData(spcchart: QCSPCChartTS.SPCChartBase, count: number, mean: number,
sigma: number)  {
    // batch number for a given sample subgroup
    let batchCounter: number = 0;
        let i: number = 0;
        let timestamp: Date = new Date();
        let chartdata: QCSPCChartTS.SPCControlChartData | null =
spcchart.getChartData();
        if (!chartdata) return;
        let currentcount = chartdata.getCurrentNumberRecords();

    for (i = 0; i < count; i++) {
        // Simulate a sample subgroup record
        let samples: QCSPCChartTS.DoubleArray =
chartdata.simulateMeasurementRecordMeanRange(mean, sigma);
        // Update chart data using i as the batch number
        batchCounter = currentcount + i;
        let note: string = "";
        if ((i % 5) == 0) note = "This is a note";
        // Add a new sample record to the chart data
        chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, note);
        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }

}
```

Both kinds of tooltips are on by default. Turn the tooltips on or off in the program using the **EnableDataToolTip** and **EnableNotesToolTip** flags.

[JavaScript / TypeScript]

```
// Enable data and notes tooltips
attribchart.setEnableDataToolTip(true);
attribchart.setEnableNotesToolTip(true);
```

The notes tooltip has an additional option. In order to make the notes tooltip “editable”, the tooltip, which is a simple edit box, displays on the first click, and goes away on the second click. You can click inside the edit box and not worry the tooltip suddenly disappearing. The notes tooltip works this way by default. If you wish to explicitly set it, or change it so that the tooltip only displays while the mouse button is held down, as the data tooltips do, set the **chartdata.NotesToolTips.ToolTipMode** property to **NotesToolTip.MOUSEDOWN_TOOLTIP**, as in the example below.

[JavaScript]

```
// Enable data and notes tooltips
attribchart.setEnableDataToolTip(true);
attribchart.setEnableNotesToolTip(true);
```

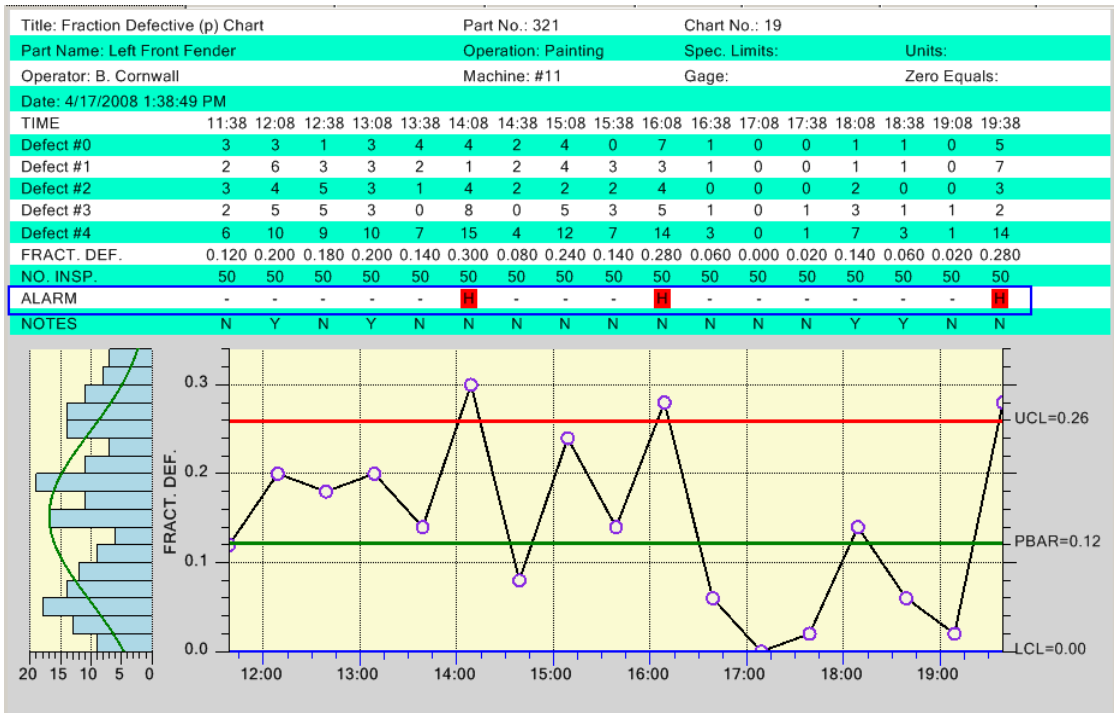
```
var chartdata = attribchart.getChartData();
if (chartdata) {
   var tooltip = chartdata.getNotesTooltips();
   tooltip.setButtonMask( QCSPCChartTS.ChartConstants.RIGHT_BUTTON);
  // default is MOUSETOGGLE_TOOLTIP
   tooltip.setToolTipMode(QCSPCChartTS.NotesToolTip.MOUSEDOWN_TOOLTIP);
}
```

[TypeScript]

```
// Enable data and notes tooltips
attribchart.setEnableDataToolTip(true);
attribchart.setEnableNotesToolTip(true);

let chartdata: QCSPCChartTS.SPCControlChartData | null =
attribchart.getChartData();
if (chartdata) {
   let tooltip: QCSPCChartTS.SPCDataTooltip | null = chartdata.getNotesTooltips();
   tooltip.setButtonMask( QCSPCChartTS.ChartConstants.RIGHT_BUTTON);
  // default is MOUSETOGGLE_TOOLTIP
  tooltip.setToolTipMode(QCSPCChartTS.NotesToolTip.MOUSEDOWN_TOOLTIP);
}
```

Enable Alarm Highlighting

EnableAlarmStatusValues



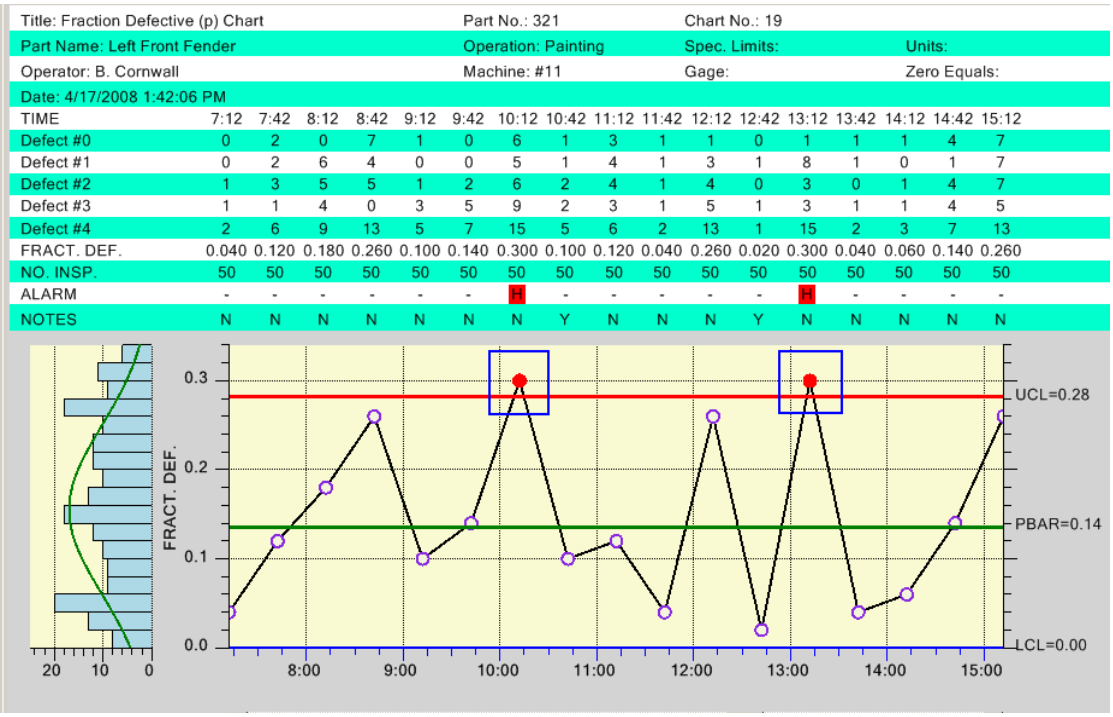
There are several alarm highlighting options you can turn on and off. The alarm status line above is turned on/off using the EnableAlarmStatusValues property. We have set it on by default, so you will have to turn it off if you don't want it. Each sample interval has

two small boxes that are labeled using one of three different characters. An “H” signifies a high alarm, a “L” signifies a low alarm, and a “-“ signifies that there is no alarm.

[JavaScript / TypeScript]

```
attribchart.setEnableAlarmStatusValues(false);
```

ChartAlarmEmphasisMode



[Javasc

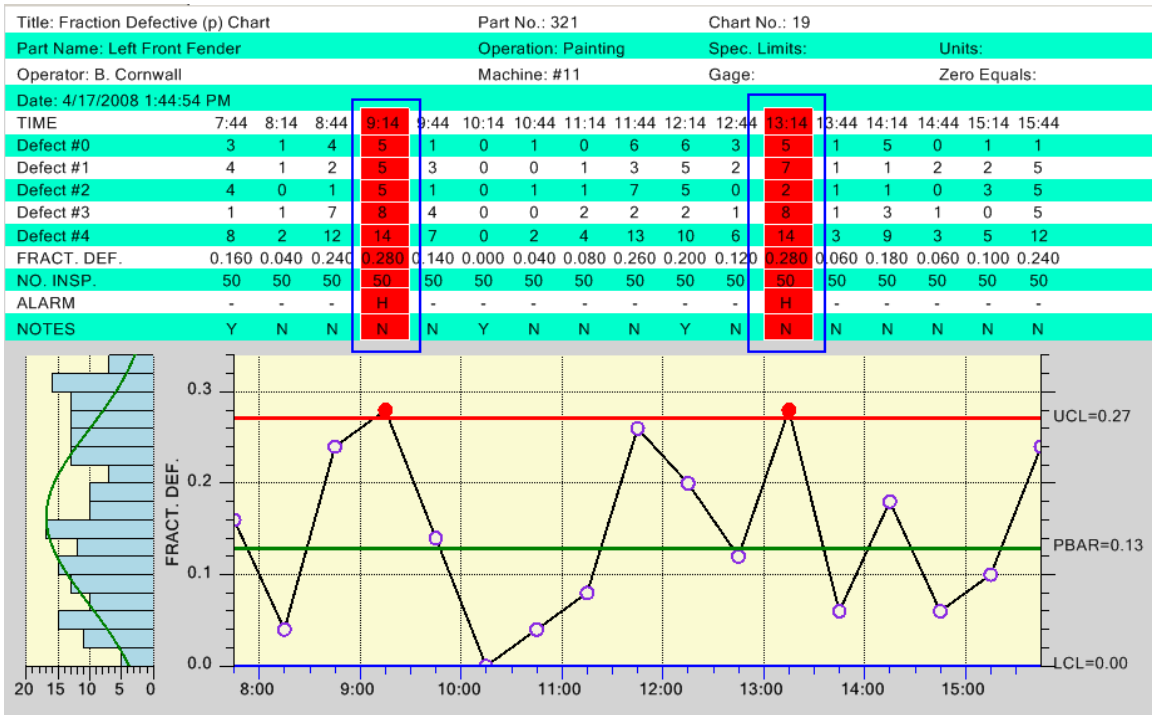
[JavaScript / TypeScript]

```
attribchart.setChartAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_SY
MBOL);
```

The scatter plot symbol used to plot a data point in the primary and secondary charts is normally a fixed color circle. If you turn on the alarm highlighting for chart symbols the symbol color for a sample interval that is in an alarm condition will change to reflect the color of the associated alarm line. In the example above, a low alarm (blue circle) occurs at the beginning of the chart and a high alarm (red circle) occurs at the end of the chart.

Alarm symbol highlighting is turned on by default. To turn it off use the SPCChartBase.ALARM_NO_HIGHLIGHT_SYMBOL constants.

TableAlarmEmphasisMode



[JavaScript / TypeScript]

```
// Table alarm emphasis mode
attribchart.setTableAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_BA
R);
```

The entire column of the data table can be highlighted when an alarm occurs. There are four modes associated with this property:

ALARM_HIGHLIGHT_NONE	No alarm highlight
ALARM_HIGHLIGHT_TEXT	Text alarm highlight
ALARM_HIGHLIGHT_OUTLINE	Outline alarm highlight
ALARM_HIGHLIGHT_BAR	Bar alarm highlight

The example above uses the ALARM_HIGHLIGHT_BAR mode.

Title: Variable Control Chart (X-Bar & R)							Part No.: 283501				Chart No.: 17						
Part Name: Transmission Casing Bolt							Operation: Threading				Spec. Limits:				Units: 0.0001 inch		
Operator: J. Fenamore							Machine: #11				Gage: #8645				Zero Equals: zero		
Date: 4/15/2008 4:08:49 PM																	
TIME	6:23	6:38	6:53	7:08	7:23	7:38	7:53	8:08	8:23	8:38	8:53	9:08	9:23	9:38	9:53	10:08	10:23
MEAN	32.0	28.2	32.5	23.2	26.5	30.2	26.6	28.4	36.5	28.7	27.7	28.8	29.3	30.0	35.0	27.3	30.0
RANGE	16.7	17.6	16.7	12.3	15.0	14.7	17.8	16.9	15.7	15.9	21.1	9.8	19.3	19.0	11.7	14.5	17.7
SUM	160.2	141.0	162.5	116.1	132.3	151.1	132.9	142.0	182.6	143.3	138.6	143.8	146.4	150.0	175.2	136.5	150.0
Cpk	0.173	0.172	0.173	0.170	0.169	0.169	0.167	0.167	0.169	0.168	0.167	0.167	0.166	0.166	0.168	0.167	0.166
Cpm	0.229	0.228	0.228	0.228	0.227	0.227	0.227	0.226	0.226	0.226	0.225	0.225	0.225	0.224	0.225	0.225	0.224
Ppk	0.168	0.167	0.168	0.165	0.164	0.164	0.162	0.162	0.163	0.163	0.162	0.162	0.161	0.161	0.163	0.162	0.161
ALARM	-	-	-	-	-	-	L	-	-	-	-	-	-	H	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

The example above uses the ALARM_HIGHLIGHT_TEXT mode

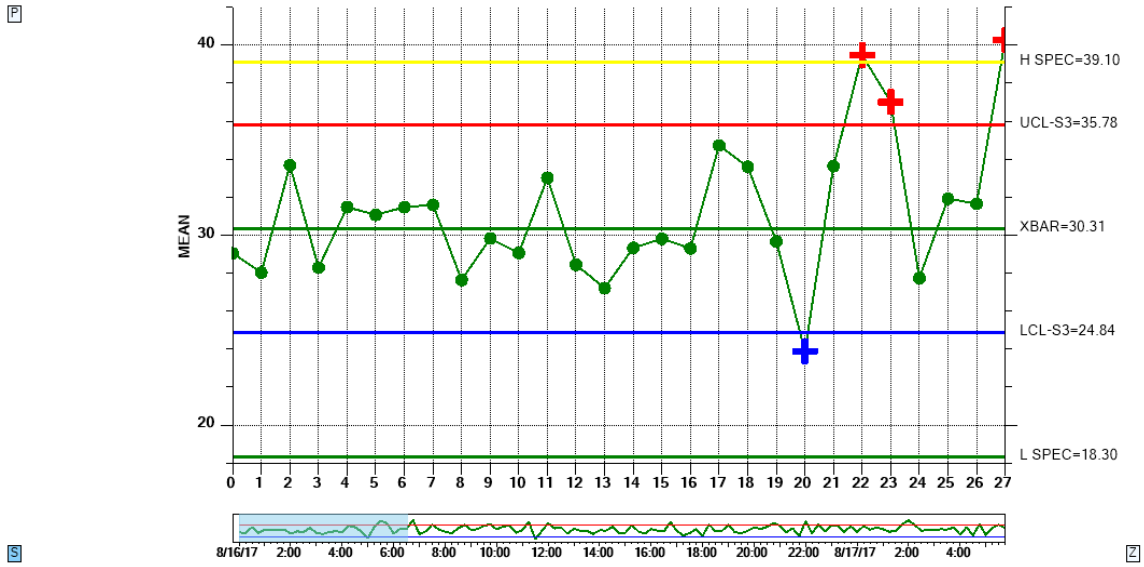
Title: Variable Control Chart (X-Bar & R)							Part No.: 283501				Chart No.: 17						
Part Name: Transmission Casing Bolt							Operation: Threading				Spec. Limits:				Units: 0.0001 inch		
Operator: J. Fenamore							Machine: #11				Gage: #8645				Zero Equals: zero		
Date: 4/15/2008 4:11:27 PM																	
TIME	12:41	12:56	13:11	13:26	13:41	13:56	14:11	14:26	14:41	14:56	15:11	15:26	15:41	15:56	16:11	16:26	16:41
MEAN	24.3	29.8	29.5	33.1	30.4	28.8	37.4	25.5	29.2	24.6	26.2	29.5	31.1	28.6	31.1	27.6	34.7
RANGE	9.2	19.1	17.4	12.7	12.6	12.0	10.5	20.0	16.7	16.4	16.4	13.2	16.9	16.2	12.1	19.3	8.1
SUM	121.6	149.1	147.5	165.6	152.1	143.8	187.1	127.6	145.8	123.2	131.1	147.5	155.3	142.9	155.5	138.1	173.4
Cpk	0.188	0.188	0.187	0.188	0.188	0.188	0.190	0.189	0.188	0.186	0.185	0.184	0.184	0.184	0.184	0.183	0.185
Cpm	0.226	0.226	0.225	0.225	0.225	0.226	0.226	0.225	0.225	0.225	0.224	0.224	0.224	0.224	0.224	0.223	0.224
Ppk	0.184	0.183	0.183	0.184	0.184	0.184	0.186	0.184	0.183	0.181	0.180	0.180	0.180	0.179	0.179	0.178	0.180
ALARM	L	-	-	-	-	-	-	-	-	H	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

The example above uses the ALARM_HIGHLIGHT_OUTLINE mode. In the table above, the column outlines in blue and red reflect what is actually displayed in the chart, whereas in the other TableAlarmEmphasisMode examples the outline just shows where the alarm highlighting occurs.

The default mode is ALARM_HIGHLIGHT_NONE mode.

Enhanced Out of Limit Symbol

One of the SPC chart options is to mark a data point which is outside of control limits by changing the color of the symbol. It is now possible to also also change the size and symbol type for out of limit symbols.



A data point which is outside of control limits can be automatically marked using color, symbol type, and color.

In the example above, the out of control symbol for the Primary chart is an extra large plus sign, while for the the Secondary chart it is an extra large square, both contrasting with the default circle symbol.

The default symbol for the out of control indication is the same as the in control indication, a filled circle. Only a color change signifies out of control. To change the notification symbol, and size, use the **OutOfLimitSymbolNumber** and **OutOfLimitSymbolSize** properties, which apply separately to the Primary and Secondary charts.

setOutOfLimitSymbolNumber

Set the symbol type for data points found to be in alarm. Use one of the symbol type constants found in the ChartConstants class: SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE. :

setOutOfLimitSymbolSize

Set the size in pixels of the out of limit symbol:

For example:

[JavaScript]

```
var primarychart = xbarsigmachart.getPrimaryChart();
if (primarychart)
{
// Set symbol emphasis type, and size, for primary chart
    primarychart.setOutOfLimitSymbolNumber( QCSPCChartTS.ChartConstants.PLUS);
    primarychart.setOutOfLimitSymbolSize(16);
}
```

[TypeScript]

```
let primarychart: QCSPCChartTS.SPCChartObjects | null =
xbarsigmachart.getPrimaryChart();

if (primarychart)
{
// Set symbol emphasis type, and size, for primary chart
    primarychart.setOutOfLimitSymbolNumber( QCSPCChartTS.ChartConstants.PLUS);
    primarychart.setOutOfLimitSymbolSize(16);
}
```

See VariableControlCharts.BuildXBarSigmaChart for an example.

AutoLogAlarmsAsNotes

When an alarm occurs, details of the alarm can be automatically logged as a Notes record. Just set the AutoLogAlarmsAsNotes property to true.

```
attribchart.setAutoLogAlarmsAsNotes( true);
```

Changing the Batch and Event Control Chart X-Axis Labeling Mode

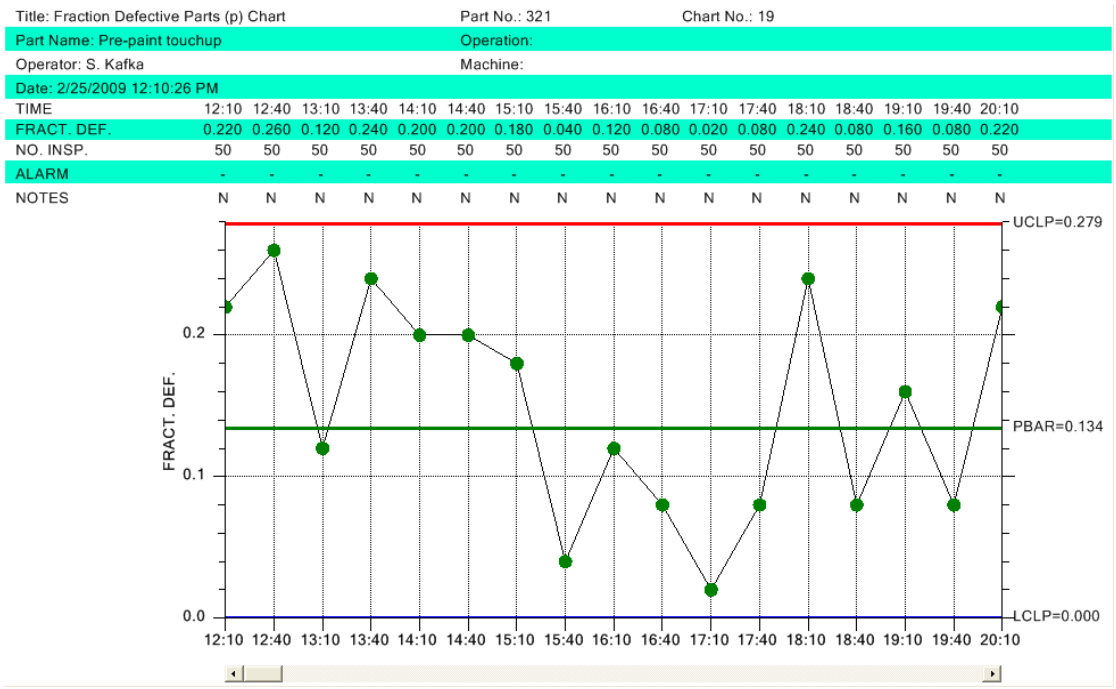
IYou may find that labeling every subgroup tick mark with a time stamp, or a user-defined string, causes the axis labels to stagger because there is not enough room to display the tick mark label without overlapping its neighbor. In these cases you may wish to reduce the number of sample subgroups you show on the page using the *numdatapointsinview* variable found in all of the example programs.

```
// Number of datapoints in the view
 numdatapointsinview = 13;
```

You can rotate the x-axis labels using the charts XAxisLabelRotation property.

```
attribchart.setXAxisLabelRotation(90);
```

Batch Control Chart X-Axis Time Stamp Labeling



Batch X-Bar R Chart using time stamp labeling of the x-axis

Set the x-axis labeling mode using the overall charts XAxisStringLabelMode property, setting it SPCChartObjects.AXIS_LABEL_MODE_TIME.

```
attribchart.setEnableScrollBar( true);
attribchart.setEnableCategoryValues(false);

// Label the tick mark with time stamp of sample group
attribchart.setXAxisStringLabelMode(QCSPCChartTS.SPCChartObjects.AXIS_LABEL_MODE_T
IME);
```

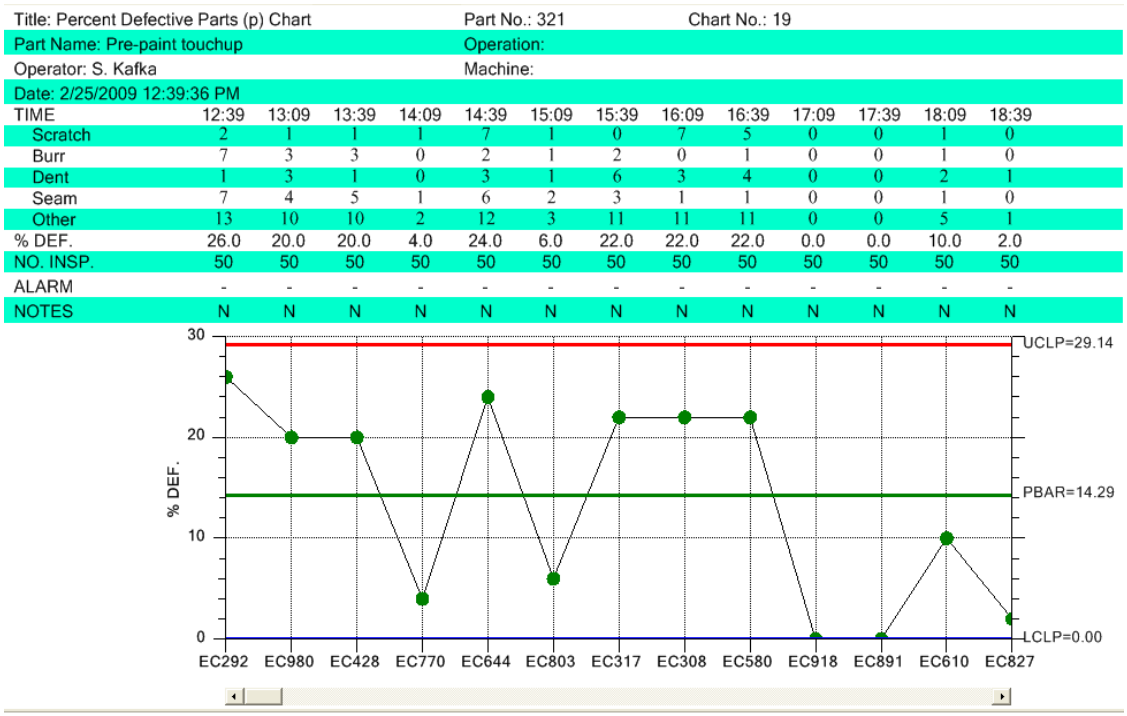
When updating the chart with sample data, use addNewSampleRecord overload that has batch number and a time stamp parameters.

```
chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter, timestamp,
samples, note);
```

See the example program AttributeControlCharts.BuildPChartF

for an example. Reset the axis labeling mode back to batch number labeling by assigning the XAxisStringLabelMode property to SPCChartObjects.AXIS_LABEL_MODE_DEFAULT.

Batch Control Chart X-Axis User-Defined String Labeling



Batch X-Bar R Chart user-defined string labeling of the x-axis

Set the x-axis labeling mode using the overall charts XAxisStringLabelMode property, setting it QCSPCChartTS.SPCChartObjects.AXIS_LABEL_MODE_STRING.

```
// Label the tick mark with user-defined strings
attribchart.setXAxisStringLabelMode(QCSPCChartTS.SPCChartObjects.AXIS_LABEL_MODE_S
TRING);
```

Use the addAxisUserDefinedString method to supply a new string for every new sample subgroup. It must be called every time the addNewSampleRecord method is called, or the user-defined strings will get out of sync with their respective sample subgroup. Reset the axis labeling mode back to batch number labeling by assigning the

XAxisStringLabelMode property to SPCChartObjects.AXIS_LABEL_MODE_DEFAULT.

[JavaScript]

```
spcchart.getChartData().addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter
, timestamp, samples, note);

var randomnum=  Math.round(100 * QCSPCChartTS.ChartSupport.getRandomDouble());
var batchidstring = "EC" + randomnum.toString();

spcchart.getChartData().addAxisUserDefinedString(batchidstring);
```

[TypeScript]

```
chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter, timestamp,
samples, note);

let randomnum: number=  Math.round(100 *
QCSPCChartTS.ChartSupport.getRandomDouble());
let batchidstring: string = "EC" + randomnum.toString();

chartdata.addAxisUserDefinedString(batchidstring);
```

See the example program AttributeControlCharts.BuildCChart for an example.

Changing Default Characteristics of the Chart

All *Attribute Control Charts* have one distinct graph with its own set of properties. This graph is the Primary Chart.

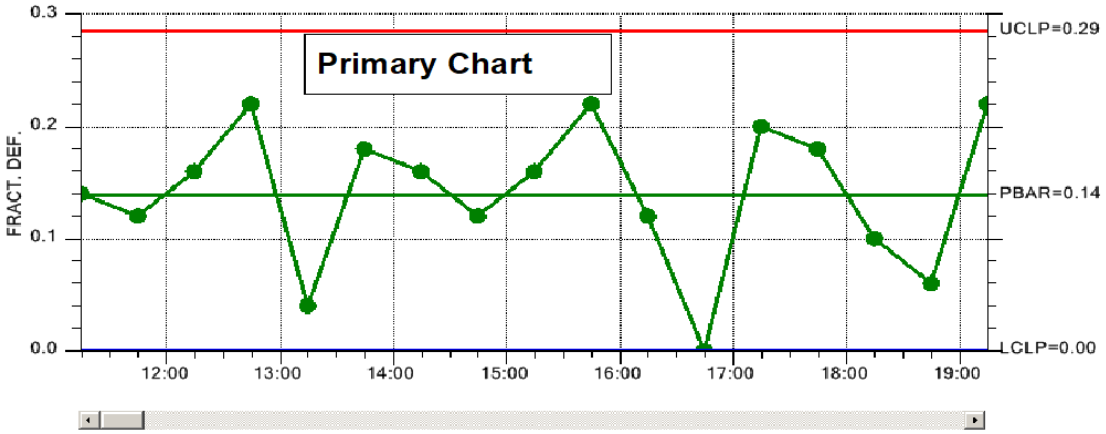
Title: Fraction Defective (p) Chart | Part No.: 321 | Chart No.: 19

Part Name: Pre-paint touchup | Operation:

Operator: S. Kafka | Machine: #1

Date: 11/22/2005 11:14:35 AM

Time	11:14	11:44	12:14	12:44	13:14	13:44	14:14	14:44	15:14	15:44	16:14	16:44	17:14	17:44	18:14	18:44	19:14
#1	0	3	3	6	1	4	5	2	5	6	1	0	1	4	1	0	3
#2	1	3	4	6	1	4	4	2	2	2	1	0	2	6	0	1	2
#3	4	2	3	3	1	4	4	3	4	3	3	0	2	3	1	0	2
#4	2	4	0	7	1	2	0	1	3	2	3	0	5	3	3	2	6
#5	7	6	8	11	2	9	8	6	8	11	6	0	10	9	5	3	11
FRACT. DEF.	0.140	0.120	0.160	0.220	0.040	0.180	0.160	0.120	0.160	0.220	0.120	0.000	0.200	0.180	0.100	0.060	0.220
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
NOTES	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N



Logically enough, the properties of the objects that make up each of these graphs are stored in a property named PrimaryChart. Once the graph is initialized, you can modify the default characteristics of each graph using these properties.

[JavaScript]

```
var attribchart =
QCSPCChartTS.SPCBatchAttributeControlChart.newSPCBatchAttributeControlChartChartTy
peSubgroupSize(htmlcanvas, spccharttype, 1, subgroupsize, numberpointsinview);

var primarychart = attribchart.getPrimaryChart();
if (primarychart)
{
    primarychart.setPlotMeasurementValues(true);
    primarychart.getXAxis().setLineColor( QCSPCChartTS.ChartColor.BLUE);
    primarychart.getXAxis().setLineWidth(3);
}
```

[TypeScript]

```
let attribchart: QCSPCChartTS.SPCChartBase | null =
QCSPCChartTS.SPCBatchAttributeControlChart.newSPCBatchAttributeControlChartChartTy
peSubgroupSize(htmlcanvas, spccharttype, 1, subgroupsize, numberpointsinview);

let primarychart: QCSPCChartTS.SPCChartObjects | null =
attribchart.getPrimaryChart();
if (primarychart)
```

```
{
    primarychart.setPlotMeasurementValues(true);
    primarychart.getXAxis().setLineColor( QCSPCChartTS.ChartColor.BLUE);
    primarychart.getXAxis().setLineWidth(3);
}
```

The **PrimaryChart** object is an instance of the **SPCChartObjects** class. The **SPCChartObjects** class contains the objects needed to display a single graph. Below you will find a summary of the class properties.

Note - Since JavaScript/TypeScript does not support properties in the same way that C# does, you set and get the values of the properties using the "set" and "get" in front of the property name, i.e. **getAnnotationFont** and **setAnnotationFont.**

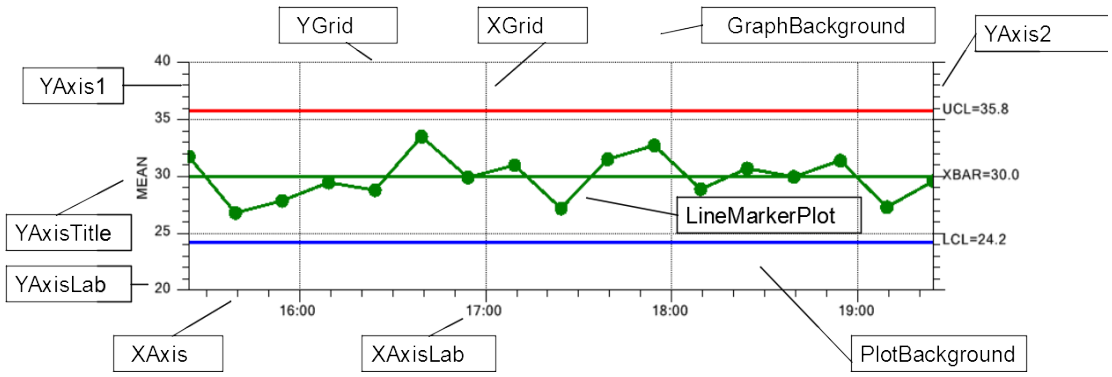
Public Instance Properties

AnnotationArray	set/get the array of TextObject objects, representing the annotations of the chart.
AnnotationFont	set/get annotation font.
AnnotationNudge	set/get the x and y-values use to offset a data points annotation with respect to the actual data point.
AxisLabelFont	set/get the font used to label the x- and y-axes.
AxisTitleFont	set/get the font used for the axes titles.
ControlLabelPosition	set/get that numeric label for a control limit is placed inside, or outside the plot area INSIDE_PLOTAREA.
ControlLimitData	set/get the array of the plot objects associated with control limits.
Datatooltip	set/get a reference to the charts tooltip.
DefaultChartBackgroundColor	set/get the default background color for the graph area.
DefaultNumberControlLimits	set/get the number of control limits in the chart.
DefaultPlotBackgroundColor	set/get the default background color for the plot area.
DisplayChart	set/get to true to enable the drawing of this chart.
DisplayFrequencyHistogram	set/get to true to enable the drawing of the frequency histogram attached to the chart.
FrequencyHistogramChart	set/get a reference to the optional frequency histogram attached to the chart.
GraphBackground	set/get a reference to the charts graph background object.

BatchIncrement	set/get increment between adjacent samples of Batch type charts that use a numeric x-scale.
BatchStartValue	set/get the starting numeric value of the x-scale for Batch type charts that use a numeric x-scale.
BatchStopValue	set/get the ending numeric value of the x-scale for Batch type charts that use a numeric x-scale.
Header	set/get a reference to the charts header.
HeaderFont	set/get the font used for the chart title.
HistogramStartPos	set/get the left edge, using normalized coordinates, of the frequency histogram plotting area.
HistogramOffset	set/get the offset of the histogram with respect to the GraphStartPosX position, using normalized coordinates, of the frequency histogram plotting area.
MaxY	set/get the maximum value used to scale the y-axis of the chart.
MinY	set/get the minimum value used to scale the y-axis of the chart.
ParentSPCChartBase	set/get that parent SPCChartBase object.
PlotBackground	set/get a reference to the charts plot background object.
PlotMeasurementValues	set/get to true to enable the plotting of all sampled values, as a scatter plot, in addition to the mean or median values.
PPhysTransform1	set/get a reference to the charts physical coordinate system.
ProcessVariableData	Holds a reference to an object encapsulating the plot object data associated with the main variable of the chart.
SampledDataData	set/get the array of the sample data.
SubHead	set/get a reference to the charts subhead.
SubheadFont	set/get the font used for the chart subhead.
TableFont	set/get the font used for the data table.
TextTemplate	set/get the text template for the data tooltip.
TimeIncrementMinutes	set/get the increment between adjacent samples of charts that use a numeric x-scale.
ToolTipFont	set/get tooltip font.
ToolTipSymbol	set/get a reference to the charts tooltip

	symbol.
XAxis	set/get a reference to the charts x-axis.
XAxisLab	set/get a reference to the charts x-axis labels.
XGrid	set/get a reference to the charts x-axis grid.
XValueTemplate	set/get the x-value template for the data tooltip.
YAxis1	set/get a reference to the charts left y-axis.
YAxis2	set/get a reference to the charts right y-axis.
YAxisLab	set/get a reference to the charts left y-axis labels.
YAxisTitle	set/get a reference to the charts left y-axis title.
YGrid	set/get a reference to the charts y-axis grid.
YValueTemplate	set/get the y-value template for the data tooltip.

The main objects of the graph are labeled in the graph below.



Chapter 8 - Named and Custom Control Rule Sets

Western Electric (WECO) Rules

The normal SPC control limit rules display at the 3-sigma level, both high and low. In this case, a simple threshold test determines if a process is in, or out of control. Once a process is brought under control using the simple 3-sigma level tests, quality engineers often want to increase the sensitivity of the control chart, detecting and correcting problems before the 3-sigma control limits are reached. Other, more complex tests rely on more complicated decision-making criteria. These rules utilize historical data and look for a non-random pattern that can signify that the process is out of control, before reaching the normal +-3 sigma limits. The most popular of these are the Western Electric Rules, also know as the WECO Rules, or WE Runtime Rules. First implemented by the Western Electric Co. in the 1920's, these quality control guidelines were codified in the 1950's and form the basis for all of the other rule sets. Different industries across the globe have have developed their own variants on the WECO Rules. Other sets of rules, common enough to have an identifying name, i.e. *named rules*, are listed below.

WECO Runtime and Supplemental Rules – Western Electric Co. - Western Electric Company (1956), *Statistical Quality Control handbook.* (1 ed.), Indianapolis, Indiana: Western Electric Co., p. v, OCLC 33858387. Sometimes the Supplemental Rules are referred to as the Montgomery Rules, after the statistical quality control expert Douglas Mongtomery. *Introduction to Statistical Quality Control* (5 ed.), Hoboken, New Jersey: John Wiley & Sons, ISBN 9780471656319

Nelson Rules – The Nelson rules were first published in the October 1984 issue of the Journal of Quality Technology in an article by Lloyd S Nelson.

AIAG Rules– The (AIAG) Automotive Industry Action Group control rules are published in the their industry group "Statistical Process Control Handbook".

Juran Rules - Joseph M. Juran was an international expert in quality control and defined these rules in his "Juran's Quality Handbook", McGraw-Hill Professional; 6 edition (May 19, 2010), **ISBN-10:** 0071629734

Hughes Rules – The only sources we could find for the Hughes rules were all second hand. If anyone can direct is to an original source for the Hughes Rules, please send an e-mail to support@quinn-curtis.com.

Duncan Rules – Acheson Johnston Duncan was an international expert in quality control and published his rules in the text book "Quality control and industrial statistics" (fifth edition). Irwin, 1986.

Gitlow Rules - Dr. Howard S. Gitlow is an international expert in Sigma Six, TQM and SPC. His rules are found in his book "Tools and Methods for the Improvement of Quality", 1989, **ISBN-10: 0256056803** .

Westgard Rules – The Westgard rules are based on the work of James Westgard, a leading expert in laboratory quality management . They are considered "Laboratory quality control rules". You can find more information about the Westgard Rules, and James Westgard at the web site: http://www.westgard.com

The rules sets have many individual rules in common. In particular, the WECO rules and the Nelson rules, have 7 out of 8 rules in common, and only differ in the fourth rule.

Western Electric (WECO) Rules

In the Western Electric Rules A process is considered out of control if any of the following criteria are met:

1. **The most recent point plots outside one of the 3-sigma control limits.** If a point lies outside either of these limits, there is only a 0.3% chance that this was caused by the normal process.

2. **Two of the three most recent points plot outside and on the same side as one of the 2-sigma control limits.** The probability that any point will fall outside the warning limit is only 5%. The chances that two out of three points in a row fall outside the warning limit is only about 1%.

3. **Four of the five most recent points plot outside and on the same side as one of the 1-sigma control limits.** In normal processing, 68% of points fall within one sigma of the mean, and 32% fall outside it. The probability that 4 of 5 points fall outside of one sigma is only about 3%.

4. **Eight out of the last eight points plot on the same side of the center line, or target value.** Sometimes you see this as 9 out of 9, or 7 out of 7. There is an equal chance that any given point will fall above or below the mean. The chances that a point falls on the same side of the mean as the one before it is one in two. The odds that the next point will also fall on the same side of the mean is one in four. The probability of getting eight points on the same side of the mean is only around 1%.

These rules apply to both sides of the center line at a time. Therefore, there are eight actual alarm conditions: four for the above center line sigma levels and four for the below center line sigma levels.

There are also additional WE Rules for trending. These are often referred to as **WE Supplemental Rules**. Don't rely on the rule number, often these are listed in a different order.

5. **Six points in a row increasing or decreasing.** The same logic is used here as for rule 4 above. Sometimes this rule is changed to seven points rising or falling.

6. **Fifteen points in a row within one sigma.** In normal operation, 68% of points will fall within one sigma of the mean. The probability that 15 points in a row will do so, is less than 1%.

7. **Fourteen points in a row alternating direction.** The chances that the second point is always higher than (or always lower than) the preceding point, for all seven pairs is only about 1%.

8. **Eight points in a row outside one sigma.** Since 68% of points lie within one sigma of the mean, the probability that eight points in a row fall outside of the one-sigma line is less than 1%.

The rules are described as they appear in the literature. In many cases, a given rule actually specifies two test conditions; the first being a value N out of M above a plus sigma control limit, and the second being a value N out of M below a minus sigma control limit. Examples of this are rules #1, #2 and #3 for WECO and Nelson rules. In other cases, similar rules only contain one test case; N out of M above (or below) a given sigma control limit. Example of this are the Juran rules #2..#5, Hughes Rules #2..#9, Gitlow Rules #2..#5, and Duncan Rules #2..#5.

While the list of named rules below follow what is presented in the literature, the actual rule numbering should be ignored. That is because in the software we implement all rules as simple single condition rules. The first rule in all of the named rule sets is implemented as two rules; a single point greater than 3-sigma; and a single point less than -3-sigma. And WECO and Nelson rules #2 and #3 are implemented as four rules; two N out of M greater than x-sigma condition limits, and two N out of M less than x-sigma condition limits. A complete cross reference to the named rules listed below, and our own rule number system is found in Table 1. This is important, because when you try to access a particular named rule within the software, you must use our rule number system.

Basic Rules

The Basic Rules are the default rules for all of the SPC charts. They correspond to the +-3-sigma rules used by almost every industry standard SPC chart implementation.

1. One of one point is outside of +-3-sigma control limits

Nelson Rules

The Nelson rules are almost identical to the combination of the WECO Runtime and Supplemental Rules. The only difference is in Rule #4.

4. Nine out of the last nine points plot on the same side of the center line, or target value.

AIAG Rules

1. One of one point is outside of +-3-sigma control limits
2. Seven out of seven are above or below center line
3. Seven points in a row increasing
4. Seven points in a row decreasing

Juran Rules

1. One of one point is outside of +- 3-sigma control limits
2. Two of three points above 2-sigma control limits
3. Two of three points below -2-sigma control limits
4. Four of five points is above 1-sigma control limits
5. Four of five points is below -1-sigma control limit s
6. Six points in a row increasing
7. Six points in a row decreasing
8. Nine out of nine are above or below center line
9. Eight points in a row on both sides of center line, none in zone C

Hughes Rules

1. One of one point is outside of +- 3-sigma control limits
2. Two of three points above 2-sigma control limits
3. Two of three points below -2-sigma control limit s
4. Three of seven points above 2-sigma control limits
5. Three of seven points below -2-sigma control limit s
6. Four of ten points above 2-sigma control limits
7. Four of ten points below -2-sigma control limits
8. Four of five points is above 1-sigma control limits

9. Four of five points is below -1-sigma control limits
10. Seven points in a row increasing
11. Seven points in a row decreasing
12. Ten of eleven are above center line
13. Ten of eleven are below center line
14. Twelve of fourteen are above center line
15. Twelve of fourteen are below center line

Gitlow Rules

1. One of one point is outside of +- 3-sigma control limits
2. Two of three points above 2-sigma control limits
3. Two of three points below -2-sigma control limits
4. Four of five points is above 1-sigma control limits
5. Four of five points is below -1-sigma control limits
6. Eight points in a row increasing
7. Eight points in a row decreasing
8. Eight out of Eight are above center line
9. Eight out of Eight are below center line

Duncan Rules

1. One of one point is outside of +- 3-sigma control limits
2. Two of three points above 2-sigma control limits
3. Two of three points below -2-sigma control limits
4. Four of five points is above 1-sigma control limits
5. Four of five points is below -1-sigma control limits
6. Seven points in a row increasing
7. Seven points in a row decreasing

Westgard Rules

1. One of one point is outside of +- 3-sigma control limits - 13s
2. Two of two points outside +-2-sigma control limits - 22s
3. Four of four points outside +-1-sigma control limits - 41s
4. Ten of ten points on one side of center line - 10x
5. Two adjacent points on opposite sides of +-2-sigma - R4s
6. Seven of seven points in a trend increasing or decreasing - 7T
7. One of one point is outside of +- 2-sigma control limits – 12s

8. Two of three points outside +-2-sigma control limits - 2of32s
9. Three of three points outside +-1-sigma control limits - 31s
10. Six of six points on one side of center line - 6x
11. Eight of eight points on one side of center line - 8x
12. Nine of nine points on one side of center line - 9x
13. Twelve of twelve points on one side of center line – 12x

By default, only the first six Westgard rules described above are enabled. The others can be turned on using the **useNamedRuleSet** method and setting *ruleflags* array elements true for the additional rules. Make sure you use our rule numbers and not the rule numbering above.

The Levey-Jennings chart uses its own subset of these rules:

- One of one point is outside of +- 3-sigma control limits – 13s
- Two of two points outside +-2-sigma control limits – 22s
- Four of four points outside +-1-sigma control limits – 41s
- Ten of ten points on one side of center line - 10x
- Two adjacent points on opposite sides of +-2-sigma – R4s
- One of one point is outside of +- 2-sigma control limits – 12s

with the others being optional.

- Seven of seven points in a trend increasing or decreasing - 7T
- Two of three points outside +-2-sigma control limits - 2of32s
- Three of three points outside +-1-sigma control limits - 31s
- Six of six points on one side of center line - 6x
- Eight of eight points on one side of center line - 8x
- Nine of nine points on one side of center line - 9x
- Twelve of twelve points on one side of center line – 12x

Control Rule Templates

All of the named rules fall into one of our standard rule categories. Each rule category is a flexible template which can be used to evaluate a test condition across a wide range of parameters. A list of the template categories appears below.

Standardized Templates for Control Rule Evaluation

Template #	**Standard Control Limit tests**
1	N of M above X sigma (from center line), used for UCL tests

2	N of M below X sigma (from center line), used for LCL tests
3	Reserved
4	N of M beyond X sigma (from center line, either side) or control limits – points beyond the +- limit values – don't have to all be on one side
	Trending
5	N of M trending up (increasing)
6	N of M trending down (decreasing)
7	N of M trending up (increasing) or down (decreasing)
	Hugging (lack of variance)
8	N of M within X sigma (from center line, either side)
9	N ofr M within X sigma of each other (no reference to center line)
	Oscillation
10	N of M alternating about X sigma (from center line)
11	N of M alternating (no reference to center line)

For example, rule #1 for all of the named rules (a single point plots outside of +- 3 sigma) is implemented as one instance of template #1 (N of M above X sigma, where N=1, M=1 and X = 3) and one instance of template #2 (N of M below X sigma) where N=1, M=1 and X = -3).

Rule #2 for WECO and Nelson (two of three point plots outside of +- 2 sigma) is implemented as one instance of template #1 (N of M above X sigma, where N=2, M=3 and X = 2) and one instance of template #2 (N of M below X sigma) where N=2, M=3 and X = -2).

Rule #4 and #5 for Hughes (three of seven points above/below 2-sigma control limit) is implemented as one instance of template #1 (N of M above X sigma, where N=3, M=7 and X = 2) and one instance of template #2 (N of M below X sigma) where N=3, M=7 and X = -2).

Rule #6 for Gitlow (eight points in a row increasing) is implemented as one instance of template #5 (N of M trending up) where N=8 and M=8.

The templates are important because using them you can modify any existing named rule, changing the M, N or X parameter. Or, you can create completely new rules.

Taking these factors into account, we have redefined and renumbered the rules, identifying each with the template and parameters used by each rule, .

Standardized Template Parameters and Rule # Cross Reference for Named Rules

Basic Rules			**New**			
Rule #	**Description**	**Rule #**	**Template #**	**N of**	**M**	**X**
#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3

WECO		**New**				
Rule #	**Description**	**Rule #**	**Template #**	**N of**	**M**	**X**
#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3
#2	2 of 3 < 2 sigma	3	1	2	3	-2
	2 of 3 > 2 sigma	4	2	2	3	2
#3	4 of 5 < sigma	5	1	4	5	-1
	4 of 5 > sigma	6	2	4	5	1
#4	8 of 8 < center line	7	1	8	8	0
	8 of 8 > center line	8	2	8	8	0

WECO+Supplemental		**New**				
Rule #	**Description**	**Rule #**	**Template #**	**N of**	**M**	**X**
#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3
#2	2 of 3 < 2 sigma	3	1	2	3	-2
	2 of 3 > 2 sigma	4	2	2	3	2
#3	4 of 5 < sigma	5	1	4	5	-1
	4 of 5 > sigma	6	2	4	5	1
#4	8 of 8 < center line	7	1	8	8	0
	8 of 8 > center line	8	2	8	8	0
#5	6 of 6 incr. or dec.	9	7	6	6	0
#6	15 of 15 within 1 sigma	10	8	15	15	1
#7	14 of 14 alternating	11	11	14	14	0
#8	8 of 8 outside zone C	12	4	8	8	1

Nelson Rule #	Description	New Rule #	Template #	N of	M	X
#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3
#2	2 of 3 < 2 sigma	3	1	2	3	-2
	2 of 3 > 2 sigma	4	2	2	3	2
#3	4 of 5 < sigma	5	1	4	5	-1
	4 of 5 > sigma	6	2	4	5	1
#4	9 of 9 < center line	7	1	9	9	0
	9 of 9 > center line	8	2	9	9	0
#5	6 of 6 incr. or dec.	9	7	6	6	0
#6	15 of 15 within 1 sigma	10	8	15	15	1
#7	14 of 14 alternating	11	11	14	14	0
#8	8 points outside zone C	12	4	8	8	1

AIAG Rule #	Description	New Rule #	Template #	N of	M	X
#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3
#2	7 of 7 < center line	3	1	7	7	0
#3	7 of 7 > center line	4	2	7	7	0
#4	7 of 7 increasing	5	5	7	7	0
#5	7 of 7 decreasing	6	6	7	7	0

Juran Rule #	Description	New Rule #	Template #	N of	M	X
#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3
#2	2 of 3 < 2 sigma	3	1	2	3	-2
#3	2 of 3 > 2 sigma	4	2	2	3	2
#4	4 of 5 < sigma	5	1	4	5	-1
#5	4 of 5 > sigma	6	2	4	5	1
#6	6 of 6 increasing	7	5	6	6	0
#7	6 of 6 decreasing	8	6	6	6	0
#8	9 of 9 > center line	9	1	9	9	0
#9	9 of 9 < center line	10	2	9	9	0
#10	8 of 8 outside zone C	11	4	8	8	1

Hughes Rule #	Description	New Rule #	Template #	N of	M	X
#1	1 of 1 < 3 sigma	1	1	1	1	-3

	1 of 1 > 3 sigma	2	2	1	1	3
#2	2 of 3 < 2 sigma	3	1	2	3	-2
#3	2 of 3 > 2 sigma	4	2	2	3	2
#4	3 of 7 < 2 sigma	5	1	3	7	-2
#5	3 of 7 > 2 sigma	6	2	3	7	2
#6	4 of 10 < 2 sigma	7	1	4	10	-2
#7	4 of 10 > 2 sigma	8	2	4	10	2
#8	4 of 5 < sigma	9	1	4	5	-1
#9	4 of 5 > sigma	10	2	4	5	1
#10	7 of 7 increasing	11	5	7	7	0
#11	7 of 7 decreasing	12	6	7	7	0
#12	10 of 11 < center line	13	1	10	11	0
#13	10 of 11 > center line	14	2	10	11	0
#14	12 of 14 < center line	15	1	12	14	0
#15	12 of 14 > center line	16	2	12	14	0

Gitlow Rule #	**Description**	**New Rule #**	**Template #**	**N of**	**M**	**X**
#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3
#2	2 of 3 < 2 sigma	3	1	2	3	-2
#3	2 of 3 > 2 sigma	4	2	2	3	2
#4	4 of 5 < sigma	5	1	4	5	-1
#5	4 of 5 > sigma	6	2	4	5	1
#6	8 of 8 increasing	7	5	8	8	0
#7	8 of 8 decreasing	8	6	8	8	0
#8	8 of 8 < center line	9	1	8	8	0
#9	8 of 8 > center line	10	2	8	8	0

Duncan Rule #	**Description**	**New Rule #**	**Template #**	**N of**	**M**	**X**
#1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3
#2	2 of 3 < 2 sigma	3	1	2	3	-2
#3	2 of 3 > 2 sigma	4	2	2	3	2
#4	4 of 5 < sigma	5	1	4	5	-1
#5	4 of 5 > sigma	6	2	4	5	1
#6	7 of 7 increasing	7	5	7	7	0
#7	7 of 7 decreasing	8	6	7	7	0

Westgard Rule #	Description	New Rule #	Template #	N of	M	X
1	1 of 1 < 3 sigma	1	1	1	1	-3
	1 of 1 > 3 sigma	2	2	1	1	3
2	2 of 2 < 2 sigma	3	1	2	2	-2
	2 of 2 > 2 sigma	4	2	2	2	2
3	4 of 4 < 1 sigma	5	1	4	4	-1
	4 of 4 > 1 sigma	6	2	4	4	1
4	10 of 10 < centerline	7	1	10	10	0
	10 of 10 > centerline	8	2	10	10	0
5	R2s – 2-sigma limits	9	10	1	1	2
6	7 of 7 trending	10	7	7	7	0
7	1 of 1 > 2 sigma	11	1	1	1	-2
	1 of 1 < 2 sigma	12	2	1	1	2
8	2 of 3 > 3 sigma	13	1	2	3	-2
	2 of 3 < 3 sigma	14	2	2	3	2
9	3 of 3 > 1 sigma	15	1	3	3	-1
	3 of 3 < 1 sigma	16	2	3	3	1
10	6 of 6 < centerline	17	1	6	6	0
	6 of 6 > centerline	18	2	6	6	0
11	8 of 8 < centerline	19	1	8	8	0
	8 of 8 > centerline	20	2	8	8	0
12	9 of 9 < centerline	21	1	9	9	0
	9 of 9 > centerline	22	2	9	9	0
13	12 of 12 < centerline	23	1	12	12	0
	12 of 12 > centerline	24	2	12	12	0

For the Levey-Jennings chart Westgard rules (1-7) are in effect:

Westgard Rule #	Description	New Rule #	Template #	N of	M	X
1	1 of 1 < 3 sigma 1	1	1	1	-3	
	1 of 1 > 3 sigma 2	2	1	1	3	
2	2 of 2 < 2 sigma 3	1	2	2	-2	
	2 of 2 > 2 sigma 4	2	2	2	2	
3	4 of 4 < 1 sigma 5	1	4	4	-1	
	4 of 4 > 1 sigma 6	2	4	4	1	
4	10 of 10 < centerline	7	1	10	10	0
	10 of 10 > centerline	8	2	10	10	0
5	R2s – 2-sigma limits	9	10	1	1	2
6	7 of 7 trending 10	7	7	7	0	
7	1 of 1 > 2 sigma 11	1	1	1	-2	

	1 of 1 < 2 sigma	12	2	1	1	2	
8	2 of 3 > 3 sigma	13	1	2	3	-2	
	2 of 3 < 3 sigma	14	2	2	3	2	
9	3 of 3 > 1 sigma	15	1	3	3	-1	
	3 of 3 < 1 sigma	16	2	3	3	1	
10	6 of 6 < centerline		17	1	6	6	0
	6 of 6 > centerline		18	2	6	6	0
11	8 of 8 < centerline		19	1	8	8	0
	8 of 8 > centerline		20	2	8	8	0
12	9 of 9 < centerline		21	1	9	9	0
	9 of 9 > centerline		22	2	9	9	0
13	12 of 12 < centerline		23	1	12	12	0
	12 of 12 > centerline		24	2	12	12	0

Implementing a Named Rule Set

You are able to add a named rule set to an SPC application using a single call. Call the **useNamedRuleSet** method, passing in the appropriate rule ID.

SPCChartObjects.useNamedRuleSet Method

```
public useNamedRuleSet(ruleset: number) {

public useNamedRuleSetRuleFlags(ruleset: number, ruleflags: BoolArray)
```

Parameters

ruleset

One of the SPCControlLimitRecord named rule indentifiers: BASIC_RULES, WECO_RULES,WECOANDSUPP_RULES, NELSON_RULES,AIAG_RULES, JURAN_RULES, HUGHES_RULES,GITLOW_RULES, WESTGARD_RULES, and DUNCAN_RULES.

ruleflags

An array of bool, one for each named rule in the rule set. ///

Special Note for Levey-Jennings chart users:
You do not need to specify Westgard Rules for the Levey-Jennings chart. They are added automatically when the chart is created.

[JavaScript / TypeScript]

```
// Use the standard +-3-sigma control limits. Equivalent to WECO control rules #1
and #2.
primarychart.useNamedRuleSet(QCSPCChartTS.SPCControlLimitRecord.BASIC_RULES);

// Use the WECO rules.  This corresponds WECO rules #1... #4, which in our
organization of the rules is #1 to #8
primarychart.useNamedRuleSet(QCSPCChartTS.SPCControlLimitRecord.WECO_RULES);

// Use the WECO rules.  This corresponds WECO rules #1... #8 , which in our
organization of the rules is #1 to #12
primarychart.useNamedRuleSet(QCSPCChartTS.SPCControlLimitRecord.WECOANDSUPP_RULES)
;

// Use the complete set of Nelson rules.
primarychart.useNamedRuleSet(QCSPCChartTS.SPCControlLimitRecord.NELSON_RULES);

// Use the complete set of AIAG rules.
primarychart.useNamedRuleSet(QCSPCChartTS.SPCControlLimitRecord.AIAG_RULES);

// Use the complete set of Juran rules.
primarychart.useNamedRuleSet(QCSPCChartTS.SPCControlLimitRecord.JURAN_RULES);

// Use the complete set of Hughes rules.
primarychart.useNamedRuleSet(QCSPCChartTS.SPCControlLimitRecord.HUGHES_RULES);


// Use the complete set of Gitlow rules.
primarychart.useNamedRuleSet(QCSPCChartTS.SPCControlLimitRecord.GITLOW_RULES);

// Use the complete set of Westgard rules.
primarychart.useNamedRuleSet(QCSPCChartTS.SPCControlLimitRecord.WESTGARD_RULES);

// Use the complete set of Duncan rules.
primarychart.useNamedRuleSet(QCSPCChartTS.SPCControlLimitRecord.DUNCAN_RULES);
```

The named rule setup methods also come in variants which allow you to selectively enable/disable one or more rules from the named rule set. You do that by building a BoolArray object, containing true or false for each named rule in the rule set, and calling **useNamedRuleSet** method. The **getInitializedRuleBoolArray** is just a simple utility method which creates and fills out an appropriately sized BoolArray array with a default true, or false value.

SPCChartObjects.getInitializedRuleBoolArray Method

```
public getInitializedRuleBoolArray(ruleset: number,
    initialvalue: boolean): BoolArray
```

Parameters

ruleset

One of the SPCControlLimitRecord named rule identifiers: BASIC_RULES, WECO_RULES,WECOANDSUPP_RULES, NELSON_RULES,AIAG_RULES, JURAN_RULES, HUGHES_RULES,GITLOW_RULES, WESTGARD_RULES, and DUNCAN_RULES.

initialvalue

True or False, all values of the returned BoolArray are initialized to this value.

*All rule numbering is based on our rule numbering, which separates greater than and less than tests into separate rules, as detailed in the previous tables.

[JavaScript]

```
var ruleflags =

primarychart.getInitializedRuleBoolArray(QCSPCChartTS.SPCControlLimitRecord.NELSON
_RULES, true);
  ruleflags.setElement(9,false);
  ruleflags.setElement(10, false);
primarychart.useNamedRuleSet(QCSPCChartTS.SPCControlLimitRecord.NELSON_RULES,
false);
```

[TypeScript]

```
let ruleflags: BoolArray =

primarychart.getInitializedRuleBoolArray(QCSPCChartTS.SPCControlLimitRecord.NELSON
_RULES, true);
  ruleflags.setElement(9,false);
  ruleflags.setElement (10, false);
primarychart.useNamedRuleSetFlags(QCSPCChartTS.SPCControlLimitRecord.NELSON_RULES,
false);
```

The example above disables rule #9 (6 of 6 increasing or decreasing) and rule #10 (15 of 15 within 1 sigma) from the Nelson Rules. You use our rule numbering system for specifying which rule.

See the example program NamedControlRules for examples of how to setup an SPC chart for a given set of control rules. Once you add a set of named control rules to your SPC chart, the next thing you will want to do is set the control limits. You can either set the limits using known values, or you can have our software calculate the limits using previously acquired sample data.

If you want to explicitly set the limits you must know the historical process mean (also called the center line) and the historical process sigma. You may already know your process sigma, or you may need to calculate it as 1/3 * (UCL – process mean), where UCL is your historical +3-sigma upper control limit. Once you have those two values,

everything else is automatic. Just call the u**pdateControlLimitUsingMeanAndSigma** method. It will run through all of the control limit records and fill out the appropriate limit values and other critical parameters.

chardata.updateControlLimitsUsingMeanAndSigma Method

```
 public updateControlLimitsUsingMeanAndSigma(chartpos: number, mean: number,
sigma: number)
```

Parameters

chartpos

specifiy either SPC_PRIMARY_CONTROL_TARGET or SPC_SECONDARY_CONTROL_TARGET

mean

specify the process mean.

sigma

specify the process sigma.

[JavaScript]

```
var processMean = your process mean
var processSigma = your process sigma
chartdata.updateControlLimitsUsingMeanAndSigma(QCSPCChartTS.SPCChartObjects.PRIMAR
Y_CHART,
    processMean, processSigma);
```

[TypeScript]

```
let processMean: number = your process mean
let processSigma: number = your process sigma
chartdata.updateControlLimitsUsingMeanAndSigma(QCSPCChartTS.SPCChartObjects.PRIMAR
Y_CHART,
    processMean, processSigma);
```

The center line value and sigma have different meanings for the Primary and Secondary charts. So the **updateControlLimitsUsingMeanAndSigma** applies to only one at a time. If you use it for the secondary chart control limits, use your historical center line value for the secondary chart type you are using. Calculate the sigma value as 1/3 * (UCL – center line), where UCL is your historical +3-sigma upper control limit for your secondary chart.

You can also auto-calculate the control limits by adding test data to your application (fed into the chart using the **addNewSampleRecord** method), and calling **autoCalculateControlLimits.** This fills out the **SPCControlLimit** record for each control rule of the named rule set and makes control limit checking possible. You will

find the **autoCalculateControlLimits** method used in all of SPC charts of the NamedControlRules example program.

[JavaScript]

```
primarychart.useNamedRuleSet(QCSPCChartTS.SPCControlLimitRecord.WECO_RULES);

// Must have data loaded before any of the Auto.. methods are called
 SimulateData(100, 20);

// Calculate the SPC control limits for both graphs of the current SPC chart
attribchart.autoCalculateControlLimits();
```

[TypeScript]

```
primarychart.useNamedRuleSet(QCSPCChartTS.SPCControlLimitRecord.WECO_RULES);

// Must have data loaded before any of the Auto.. methods are called
 this.SimulateData(100, 20);

// Calculate the SPC control limits for both graphs of the current SPC chart
attribchart.autoCalculateControlLimits();
```

Modifying Existing Named Rules

Perhaps you like everything about a named rule set, except for one or more rules. For example, you want to use the Hughes rules, but want to change the N of M parameters of rules #15 and #16. You do that using the **SPCControlLimitRecord** method **setCustomRuleParameters**.

SPCControlLimitRecord.setCustomRuleParameters Method

```
public setCustomRuleParameters(valuesincalc: number, numvaluesforruleviolation:
number)
```

Parameters

valuesincalc

Specifies the number of values to use in calculation.

numvaluesforruleviolation

Specifies the number of values that must be outside alarm limit for rule violation.

The example below changes the N of M parameters of Hughes rules #15 and #16 from their default N of M value (12 of 14), to the values (15 of 18).

[JavaScript]

```
primarychart.useNamedRuleSet(QCSPCChartTS.SPCControlLimitRecord.HUGHES_RULES);

var M = 18;
var N = 15;
if (chartdata)
{
  var clr  =

chartdata.getNamedControlRuleRecord(QCSPCChartTS.SPCControlLimitRecord.HUGHES_RULE
S, 15);
  clr.setCustomRuleParameters(M, N);
  clr =
chartdata.getNamedControlRuleRecord(QCSPCChartTS.SPCControlLimitRecord.HUGHES_RULE
S, 16);
  clr.setCustomRuleParameters(M, N);
}
```

[TypeScript]

```
primarychart.useNamedRuleSet(QCSPCChartTS.SPCControlLimitRecord.HUGHES_RULES);

let M: number = 18;
let N: number = 15;
if (chartdata)
{
  let clr: SPCControlLimitRecord  =

chartdata.getNamedControlRuleRecord(QCSPCChartTS.SPCControlLimitRecord.HUGHES_RULE
S, 15);
  clr.setCustomRuleParameters(M, N);
  clr =
chartdata.getNamedControlRuleRecord(QCSPCChartTS.SPCControlLimitRecord.HUGHES_RULE
S, 16);
  clr.setCustomRuleParameters(M, N);
}
```

When trying to access the **SPCControlLimitRecord** for a given control rule, it is very important that you use the **getNamedControlRuleRecord** method, not the **getControlLimitRecord** method. This is because the array indexing of the **ControlLimitRecords** does not match the named control rule numbering, i.e., **getNamedControlRuleRecord**(15) and **getControlLimitRecord**(15) return different SPCControlLimitRecords.

If you want to enable/disable a specific rule, after they have all been created,

[JavaScript]

```
var clr  =

chartdata.getNamedControlRuleRecord(QCSPCChartTS.SPCControlLimitRecord.AIAG_RULES,
3);
clr.setAlarmEnable( false);
```

[TypeScript]

```
let clr: SPCControlLimitRecord  =
```

```
chartdata.getNamedControlRuleRecord(QCSPCChartTS.SPCControlLimitRecord.AIAG_RULES,
3);
clr.setAlarmEnable( false);
```

Creating Custom Rules Sets Based on Named Rules

You can create your own custom set of rules, mixing and matching rules from the standard, named rule sets, using the the **addControlRule** method. Also, you can invent your own rules, based on one of our standard templates, and add those rules to your custom rule set.

SPCChartObjects.addControlRule Method

```
public addControlRule(ruleset: number, rulenum: number): number
```

Parameters

ruleset

One of the SPCControlLimitRecord named rule indentifiers: BASIC_RULES, WECO_RULES,WECOANDSUPP_RULES, NELSON_RULES,AIAG_RULES, JURAN_RULES, HUGHES_RULES,GITLOW_RULES, WESTGARD_RULES, and DUNCAN_RULES.

rulenum

The rule number (our rule number)

Note: Even if you do not call one of the **use..Rules** methods, you still end up with four control limits. These correspond to the +-3-sigma control limits, for both the Primary and Secondary (were applicable) chart. So, you do not need to add those to your custom set of rules. Start with the rules you want to add after the standard +-3-sigma rules. If, for some reason you cannot live with the default +-3-sigma rules, you can disable them with a call to **EnableDefaultLimits**(false, false).

Say you want to create custom rule set for the Primary chart, combining rules from Nelson (#3, #4), Juran (#5, #5), AIAG (#3,#4), Hughes (#12) and Duncan (#8). Make the following calls in the setup portion of your program.

[JavaScript / TypeScript]

```
primarychart.setEnableDefaultLimits(false, false);
primarychart.addControlRule(QCSPCChartTS.SPCControlLimitRecord.WECO_RULES, 1)
primarychart.addControlRule(QCSPCChartTS.SPCControlLimitRecord.WECO_RULES, 2)
primarychart.addControlRule(QCSPCChartTS.SPCControlLimitRecord.NELSON_RULES, 3);
```

```
primarychart.addControlRule(QCSPCChartTS.SPCControlLimitRecord.NELSON_RULES, 4);
primarychart.addControlRule(QCSPCChartTS.SPCControlLimitRecord.JURAN_RULES, 5);
primarychart.addControlRule(QCSPCChartTS.SPCControlLimitRecord.JURAN_RULES, 6);
primarychart.addControlRule(QCSPCChartTS.SPCControlLimitRecord.AIAG_RULES, 3);
primarychart.addControlRule(QCSPCChartTS.SPCControlLimitRecord.AIAG_RULES, 4);
primarychart.addControlRule(QCSPCChartTS.SPCControlLimitRecord.HUGHES_RULES, 12);
primarychart.addControlRule(QCSPCChartTS.SPCControlLimitRecord.DUNCAN_RULES, 8);
```

Normally there will be no reason to set custom rules for the secondary chart, since all of the named rules apply to the Primary chart. Nothing stops you from doing it though. We can't say whether or not it makes sense statistically.

Creating Custom Rules Sets Based on a Template

Add your own custom rule to the rule set using another version of the **addControlRuleRecord** method. This one specifies a template, N out of M values, a sigma level to attach the control rule to, and a flag on whether or not to display a limit line for the control rule. If you have multiple control rules attached to a given sigma level, you should only display a control line for one of them.

SPCChartObjects.addControlRule Method

```
public addControlRuleTemplateDisplayLimitLine(template: number, n: number, m:
number, sigmavalue: number,  displaylimitline: boolean): number
```

Parameters

template

Specifies the standardized template number.

n

The n value for the n of m condition ///

m

The m value for the n of m condition ///

sigmavalue

The sigma value associated with the limit ///

displaylimitline

True to display a limit line for the control limit ///

In your code it would something like:

[JavaScript]

```
var template = 2; // N of M  Greater than limit value
var N = 10;
var M = 13;
var sigmavalue = 2; // control limit value is the +2-sigma value
var displaylimitline = false; // no limit line.
primarychart.addControlRule(template, N, M,
    sigmavalue, displaylimitline);
```

[TypeScript]

```
let template: number = 2; // N of M  Greater than limit value
let N: number = 10;
let M: number = 13;
let sigmavalue: number = 2; // control limit value is the +2-sigma value
let displaylimitline: boolean = false; // no limit line.
primarychart.addControlRule(template, N, M,
    sigmavalue, displaylimitline);
```

Creating Custom Rules Not Associated With Sigma Levels

Most of the preceding control rules are based on the mean and sigma of the current control chart. The trending rules (N of M increasing/ decreasing) are an exception, because they don't use the mean or sigma value anywhere in their evaluation. Regardless, since many of the named rules include trending rules, they are included with the previous section. There are a couple of other control rules not directly related to the mean and sigma value of the chart. The first is a simple numeric limit. The limit is meant to be independent of the mean and sigma level of the chart. It can also be considered a specification limit. We have three routines you can use to add a numeric control limit to a chart. The first, **addSpecLimit**, creates a specification limit which monitors the main variable of the chart, and compares its value to the specified numeric threshold. It will display in the chart as a line with a limit string to the right of the plot area.

SPCChartObjects.addSpecLimit Method

```
public addSpecLimit(speclimittype: number,
 value: number, displaystring: string, attrib: ChartAttribute):
SPCControlPlotObjectData | null
```

Parameters

speclimittype

Specifiy either SPCChartObjects.SPC_LOWER_SPEC_LIMIT or SPCChartObjects.SPC_UPPER_SPEC_LIMIT

value
Specifies the value of the specification limit.

displaystring
The optional display string displayed to the right of the spec limit line.

attrib
The line attributes of the spec limit line.

Return Value
The SPCControlPlotObjectData object of the specification limit.

[JavaScript / TypeScript]

```
primarychart.addSpecLimit(QCSPCChartTS.SPCChartObjects.SPC_LOWER_SPEC_LIMIT,
    18.3, "L
SPEC", new QCSPCChartTS.ChartAttribute(QCSPCChartTS.ChartColor.GREEN, 3.0));
primarychart.addSpecLimit(QCSPCChartTS.SPCChartObjects.SPC_UPPER_SPEC_LIMIT,
    39.1, "H
SPEC", new QCSPCChartTS.ChartAttribute(QCSPCChartTS.ChartColor.YELLOW, 3.0));
```

The second, a**ddNumericControlLimit**, creates a limit which monitors the value of one of the charts SPCCalculatedValueRecord object values. This is what you would use if you wanted to monitor a subgroup mean, range, sigma against some arbitrary control limit, since in the appropriate charts, the subgroup mean, range and sigma are SPCCalculatedValueRecord objects.

The type, and ordering of the SPCCalculatedValueRecord records is unique to each chart type. Below is a list of the chart types and the calculated values for each.

MEAN_RANGE_CHART
0. SPCCalculatedValueRecord.SPC_MEAN_CALC,
1. SPCCalculatedValueRecord.SPC_RANGE_CALC
2. SPCCalculatedValueRecord.SPC_SUM_CALC

MEDIAN_RANGE_CHART
0. SPCCalculatedValueRecord.SPC_MEDIAN_CALC
1. SPCCalculatedValueRecord.SPC_RANGE_CALC;

MEAN_SIGMA_CHART
0. SPCCalculatedValueRecord.SPC_MEAN_CALC
1. SPCCalculatedValueRecord.SPC_STD_DEVIATION_CALC
2. SPCCalculatedValueRecord.SPC_SUM_CALC

MEAN_SIGMA_CHART_VSS
0. SPCCalculatedValueRecord.SPC_MEAN_VSS_CALC
1. SPCCalculatedValueRecord.SPC_STD_DEVIATION_VSS_CALC
2. SPCCalculatedValueRecord.SPC_SUM_CALC
MEAN_VARIANCE_CHART
0. SPCCalculatedValueRecord.SPC_MEAN_CALC
1. SPCCalculatedValueRecord.SPC_VARIANCE_CALC
2. SPCCalculatedValueRecord.SPC_SUM_CALC
INDIVIDUAL_RANGE_CHART
0. SPCCalculatedValueRecord.SPC_INDIVIDUAL_COPY_VALUE
1. SPCCalculatedValueRecord.SPC_INDIVIDUAL_ABS_RANGE_CAL
MAMR_CHART
0. SPCCalculatedValueRecord.SPC_MA_CALC
1. SPCCalculatedValueRecord.SPC_MR_CALC
MAMS_CHART
0. SPCCalculatedValueRecord.SPC_MA_CALC
1. SPCCalculatedValueRecord.SPC_MS_CALC
EWMA_CHART
0. SPCCalculatedValueRecord.SPC_EWMA_CALC
1. SPCCalculatedValueRecord.SPC_MEAN_CALC
LEVEY_JENNINGS_CHART
0. SPCCalculatedValueRecord.SPC_INDIVIDUAL_COPY_VALUE
MA_CHART
0. SPCCalculatedValueRecord.SPC_MA_CALC
TABCUSUM_CHAR
0. SPCCalculatedValueRecord.SPC_CUSUM_CPLUS_CALC
1. SPCCalculatedValueRecord.SPC_CUSUM_CMINUS_CALC
2. SPCCalculatedValueRecord.SPC_MEAN_CALC, "MEAN"

PERCENT_DEFECTIVE_PARTS_CHART)
0. SPCCalculatedValueRecord.SPC_PERCENT_DEFECTIVE_PARTS_CALC
FRACTION_DEFECTIVE_PARTS_CHART
0. SPCCalculatedValueRecord.SPC_FRACTION_DEFECTIVE_PARTS_CALC
FRACTION_DEFECTIVE_PARTS_CHART_VSS
0.
SPCCalculatedValueRecord.SPC_FRACTION_DEFECTIVE_PARTS_VSS_CALC
NUMBER_DEFECTIVE_PARTS_CHART
0. SPCCalculatedValueRecord.SPC_TOTAL_DEFECTIVE_PARTS_CALC
NUMBER_DEFECTS_CHART
0. SPCCalculatedValueRecord.SPC_TOTAL_DEFECTS_CALC
NUMBER_DEFECTS_PERUNIT_CHART
0. SPCCalculatedValueRecord.SPC_FRACTION_DEFECTS_CALC
NUMBER_DEFECTS_PERUNIT_CHART_VSS

0. SPCCalculatedValueRecord.SPC_FRACTION_DEFECTS_VSS_CALC
PERCENT_DEFECTIVE_PARTS_CHART_VSS
0.
SPCCalculatedValueRecord.SPC_PERCENT_DEFECTIVE_PARTS_VSS_CALC
NUMBER_DEFECTS_PER_MILLION_CHART
0. SPCCalculatedValueRecord.SPC_TOTAL_DEFECTS_CALC
1.
SPCCalculatedValueRecord.SPC_NUMBER_DEFECTS_PER_MILLION_CALC

SPCChartObjects.addNumericControlLimit Method

Parameters

sourcevar

The SPCCalculatedValue source of the item to test.

limtype

Specifiy either SPCChartObjects.SPC_LOWERTHAN_LIMIT or SPCChartObjects.SPC_GREATERTHAN_LIMIT

value

Specifies the value of the control limit.

displaystring

The optional display string displayed to the right of the limit line.

addline

True and the limit is displayed as a line in the chart.

attrib

The line attributes of the spec limit line.

Return Value

The SPCControlPlotObjectData object of the numeric limit.

[JavaScript]

```
// chartdata.getCalculatedValueRecord(0) returns the calcuated value which
calculates the subgroup mean

var cvr  = chartdata.getCalculatedValueRecord(0);
primarychart.addNumericControlLimit(cvr,
QCSPCChartTS.SPCChartObjects.SPC_LOWER_SPEC_LIMIT,
```

```
        22.3, "L CV(0)", true,
QCSPCChartTS.ChartAttribute.newChartAttributeColorWidth(QCSPCChartTS.ChartColor.GR
EEN, 3.0));
primarychart.addNumericControlLimit(cvr,
QCSPCChartTS.SPCChartObjects.SPC_UPPER_SPEC_LIMIT,
        32.1, "H  CV(0)", true,
QCSPCChartTS.ChartAttribute .newChartAttributeColorWidth(QCSPCChartTS.ChartColor.Y
ELLOW, 3.0));
```

[TypeScript]

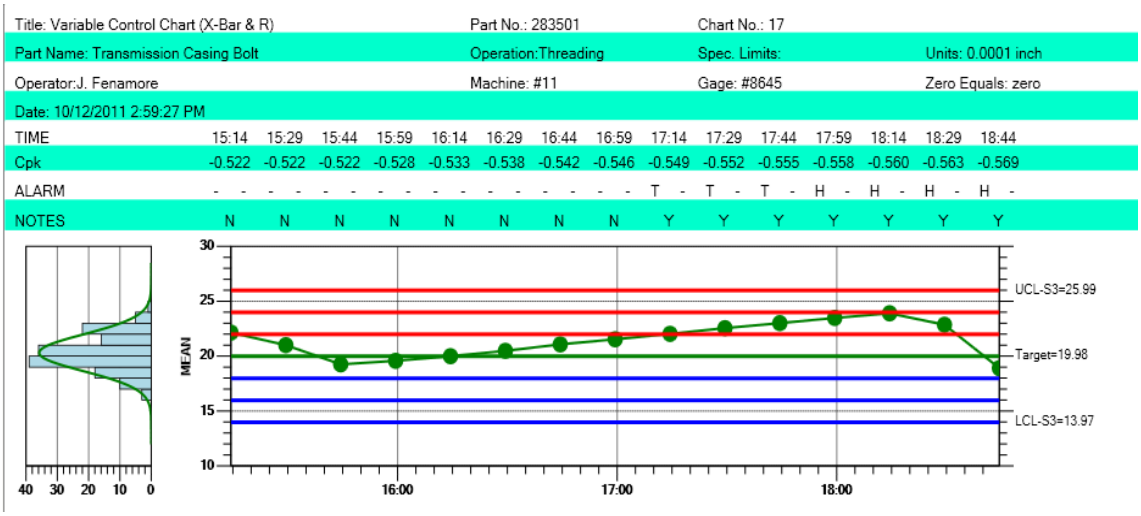
```
// chartdata.getCalculatedValueRecord(0) returns the calcuated value which
calculates the subgroup mean

let cvr: SPCCalculatedValueRecord  = chartdata.getCalculatedValueRecord(0);
primarychart.addNumericControlLimit(cvr,
QCSPCChartTS.SPCChartObjects.SPC_LOWER_SPEC_LIMIT,
        22.3, "L CV(0)", true,
QCSPCChartTS.ChartAttribute.newChartAttributeColorWidth(QCSPCChartTS.ChartColor.GR
EEN, 3.0));
primarychart.addNumericControlLimit(cvr,
QCSPCChartTS.SPCChartObjects.SPC_UPPER_SPEC_LIMIT,
        32.1, "H  CV(0)", true,
QCSPCChartTS.ChartAttribute .newChartAttributeColorWidth(QCSPCChartTS.ChartColor.Y
ELLOW, 3.0));
```

Enable Alarm Highlighting

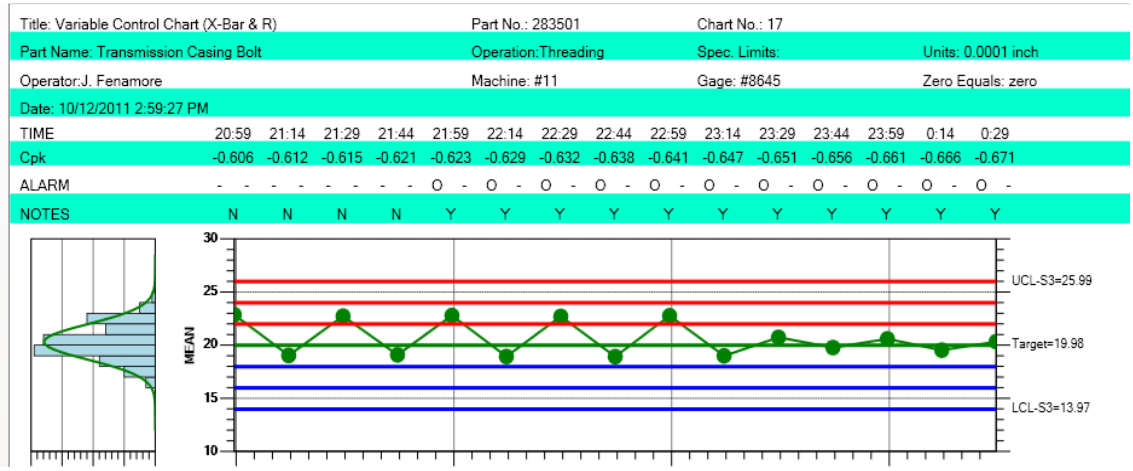
The alarm status line above is turned on/off using the **EnableAlarmStatusValues** property. We have set it on by default, so you will have to turn it off if you don't want it. Each sample interval has two small boxes that are labeled using one of several different characters, listed below. The most common are an "H" signifying a high alarm, a "L" signifying a low alarm, and a "-" signifying that there is no alarm. When specialized control rules are implemented, using the named rules, or custom rules involving trending, oscillation, or stratification, a "T", "O" or "S" may also appear.

"-"	No alarm condition
"H"	High - Measured value is above a high limit
"L"	Low - Measured value falls below a low limit
"T"	Trending - Measured value is trending up (or down).
"O"	Oscillation - Measured value is oscillating (alternating) up and down.
"S"	Stratification - Measured value is stuck in a narrow band.

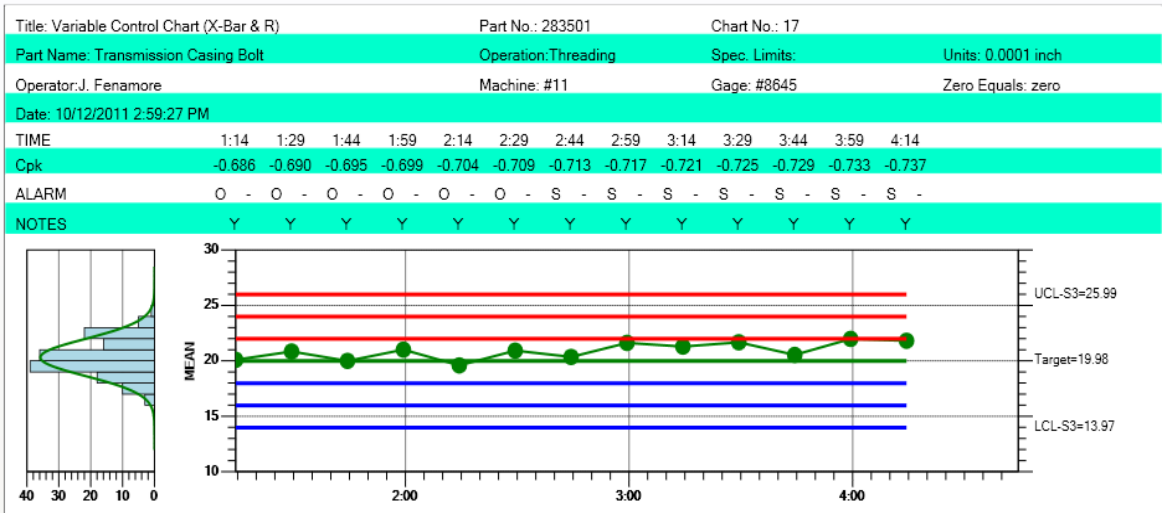
Trending



Oscillation



Stratification




```
spcchart.setEnableAlarmStatusValues(false);
```

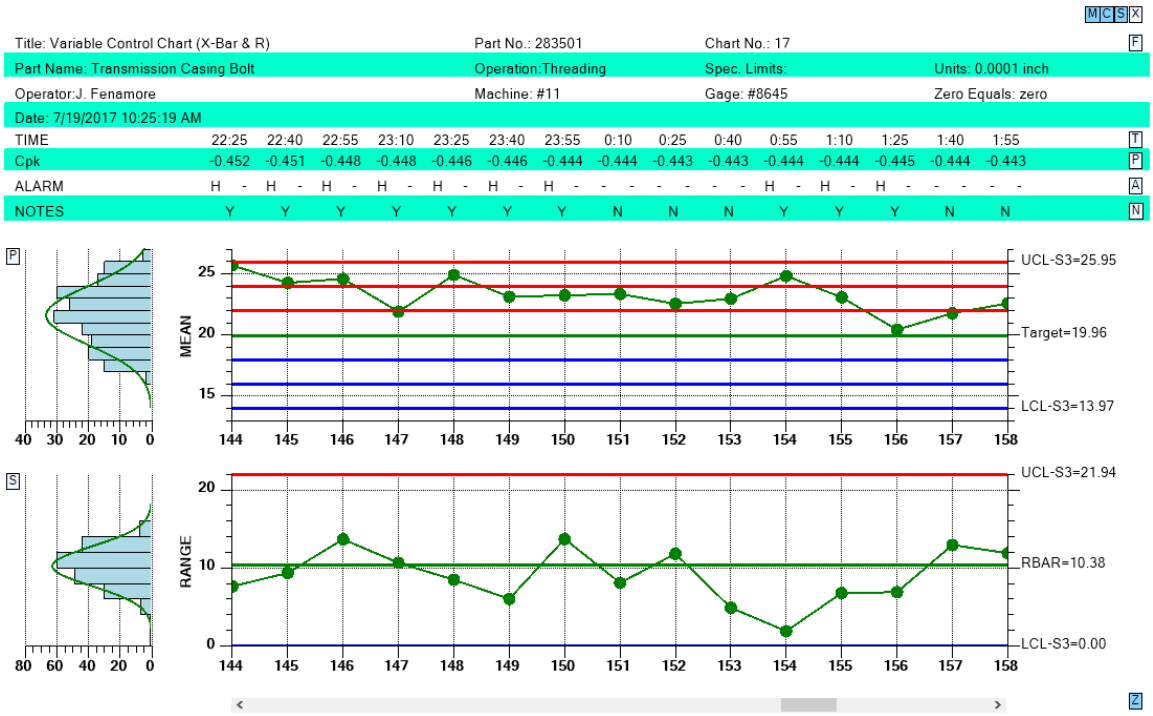
Reset N of M counters for control moves

There are cases when a user wants to keeps the same chart going, adding new sample intervals with new data, after the issue which caused the process to go out of control has been remedied. But many of the control limit tests found in the named control rules (WECO, Nelson, etc.) use N of M tests, where N out of M sample intervals must violate a specific rule. For example, a WECO rule says that if 4 out of 5 samples are outside of 1-sigma, it is an alarm condition. But if the process is now in control, the user may want to reset all N or M counts back to 0, exactly as if the plotting of the chart had started at the first sample interval. So we have added a reset mechanism for the N or M rules to a zero count for all rules.

Use the ChartData method reCenterControlLimits to reset the N of M counters back to 0.

[JavaScript / TypeScript]

```
chartdata.reCenterControlLimits();
```



The N of M counters are reset after sample interval 150 update

The example above uses the Nelson Rules. The samples intervals from 144 to 150 are shown to be in alarm, either 2 of 3 greater than 2-sigma, or 4 of 5 greater than 1-sigma. But sample interval 151 is greater than 1-sigma and should show a 4 of 5 greater than 1-sigma alarm. Yet it is not shown to be in alarm. This is because after the update for sample interval 150, the **reCenterControlLimits** method was called, resetting the counters for all N or M test back to zero. So it takes four additional 1-sigma violations to re-trigger the 4 of 5 greater than 1-sigma alarm.

Remove Negative LCL control limits for Variable Control Secondary charts, or Attribute Control Primary charts

We added a routine which when called will disable (both from display and alarm checking) any chart LCL control limit if the limit value is <= a specified value, 0.0 in most cases. Use the ChartData **setAutoDeleteControlLimits** method for this. We made the method more general than a simple routine to just delete negative control limits. It will remove any control limit that is (<, <=, >, >=) the specified value.

If a control limit meets the test criteria, which compares the control limit value, to the specified value, using the specified criteria, it is removed.

setAutoDeleteControlLimits

```
  public setAutoDeleteControlLimitsZero(compop: number, value: number)
  public setAutoDeleteControlLimitsZeroChartPos(chartpos: number, compop: number,
value: number)
```

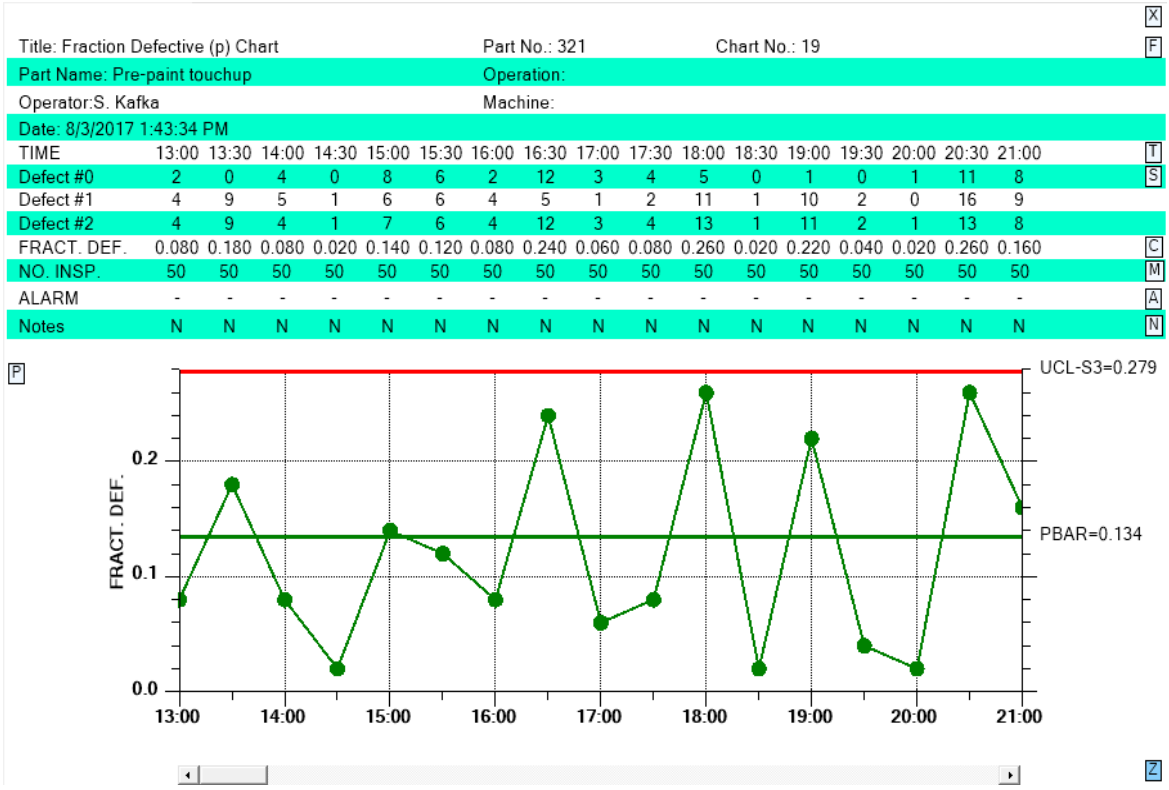
chartpos	Restrict the operation to Primary or Secondary chart. Use the constant SPCChartObjects.PRIMARY_CHART or SPCChartObjects.SECONDARY_CHART
compop	Use this comparison operator. Use one of the comparison operator constants: SPCControlChartData.COMPARISON_OP_LESSTHAN, SPCControlChartData.COMPARISON_OP_LESSTHAN_OR_EQ, SPCControlChartData.COMPARISON_OP_GREATERTHAN, SPCControlChartData.COMPARISON_OP_GREATERTHAN_OR_EQ,
value	Value used in comparison.

For a Variable Control chart, If you want to remove the LCL limit, equal to 0.0, from the Range (Secondary chart), call **setAutoDeleteControlLimits** with the following parameters.

```
chartdata.setAutoDeleteControlLimits(QCSPCChartTS.SPCChartObjects.SECONDARY_CHART,
QCSPCChartTS.SPCControlChartData.COMPARISON_OP_LESSTHAN_OR_EQ, 0.0);
```

For an Attributes Control chart, If you want to remove the LCL limit, equal to 0.0, from the Primary Chart, call **setAutoDeleteControlLimits** with the following parameters.

```
chartdata.setAutoDeleteControlLimits(QCSPCChartTS.SPCChartObjects.PRIMARY_CHART,
QCSPCChartTS.SPCControlChartData.COMPARISON_OP_LESSTHAN_OR_EQ, 0.0);
```



Title: Fraction Defective (p) Chart						Part No.: 321						Chart No.: 19					
Part Name: Pre-paint touchup						Operation:											
Operator:S. Kafka						Machine:											
Date: 8/3/2017 1:43:34 PM																	
TIME	13:00	13:30	14:00	14:30	15:00	15:30	16:00	16:30	17:00	17:30	18:00	18:30	19:00	19:30	20:00	20:30	21:00
Defect #0	2	0	4	0	8	6	2	12	3	4	5	0	1	0	1	11	8
Defect #1	4	9	5	1	6	6	4	5	1	2	11	1	10	2	0	16	9
Defect #2	4	9	4	1	7	6	4	12	3	4	13	1	11	2	1	13	8
FRACT. DEF.	0.080	0.180	0.080	0.020	0.140	0.120	0.080	0.240	0.060	0.080	0.260	0.020	0.220	0.040	0.020	0.260	0.160
NO. INSP.	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
ALARM	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Notes	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Note that the lower control limit for the Primary chart is not present. It was calculated be 0.0, and was removed by the software.

If you plan to fill the area between the control limit lines and the center line (zone colors) you must leave the 0.0 lower control limit in place so that the software can fill between the limit and the center line.

N of M testing when the most recent point entering the test is within limits

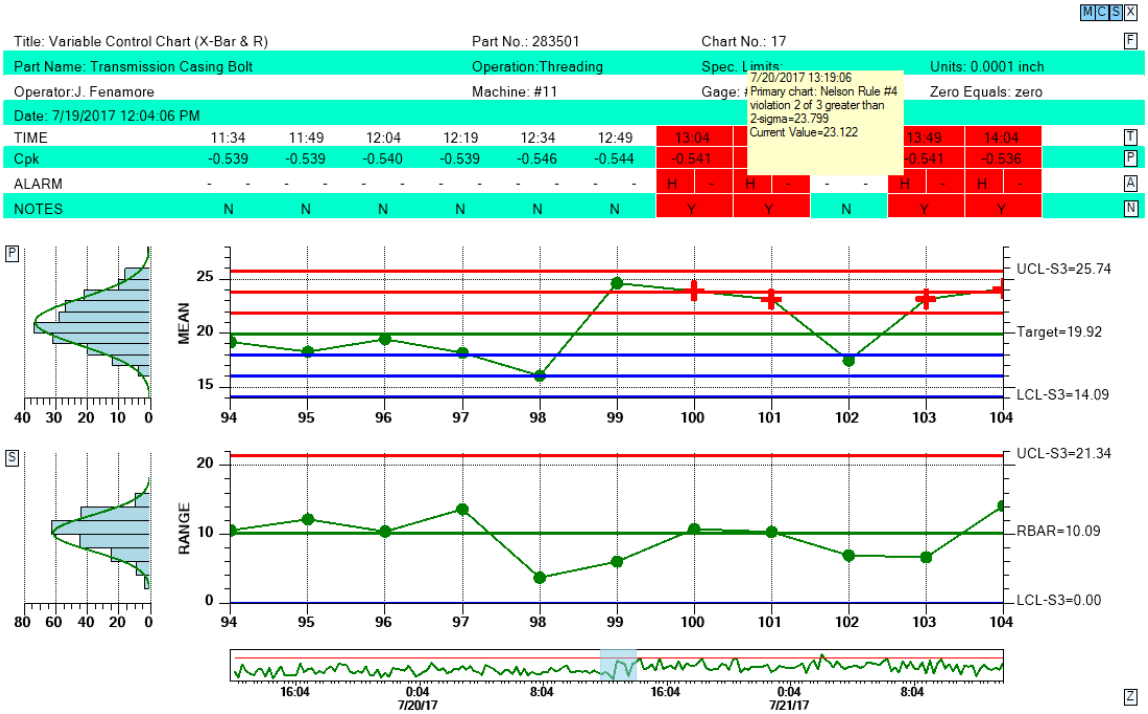
Our default mode takes a strict approach to N of M testing. Regardless of the value of the most recent point to enter the calculation (even if it is within the test limits), if N of M values are outside of limits we consider the sample interval to be in alarm. But some customers challenged this interpretation and presented us with published examples which show that if the most recent point is within limits and the previous N points out of limits, the sample interval should NOT be considered in alarm. The logic for those using the alternative evaluation scheme is that after the the first N values were found to fail the N of M test, a correction was made to the process. And therefore the next sample interval is within limits and should not be in alarm, even though it fails the strict N or M test, since it still picks up the N out of limit values before the process was corrected. We researched

the issue and found no agreement. It is implemented in the published literature both ways, going back 50 years. So a simple global flag has been added you can use to choose one evaluation method or the other, with the default being our original method.
If you want to change the N of M evaluation scheme from the default (strict N of M testing), to the alternative method, use the SPCControlLimitRecord

```
public static setDefaultAltNofMRule(value: boolean)
```

This is a static property and once set, will affect all charts that are created for the duration of the program. So you can set it at the start of your program and not have to worry about setting in subsequent chart setups.

In the picture below, the default method of N of M valuation produces an alarm at sample interval 101. While sample 101 is within 2-sigma, the previous two samples were greater than 2-sigma, so the 2 out of 3 > 2-sigma test fails for sample interval 101.



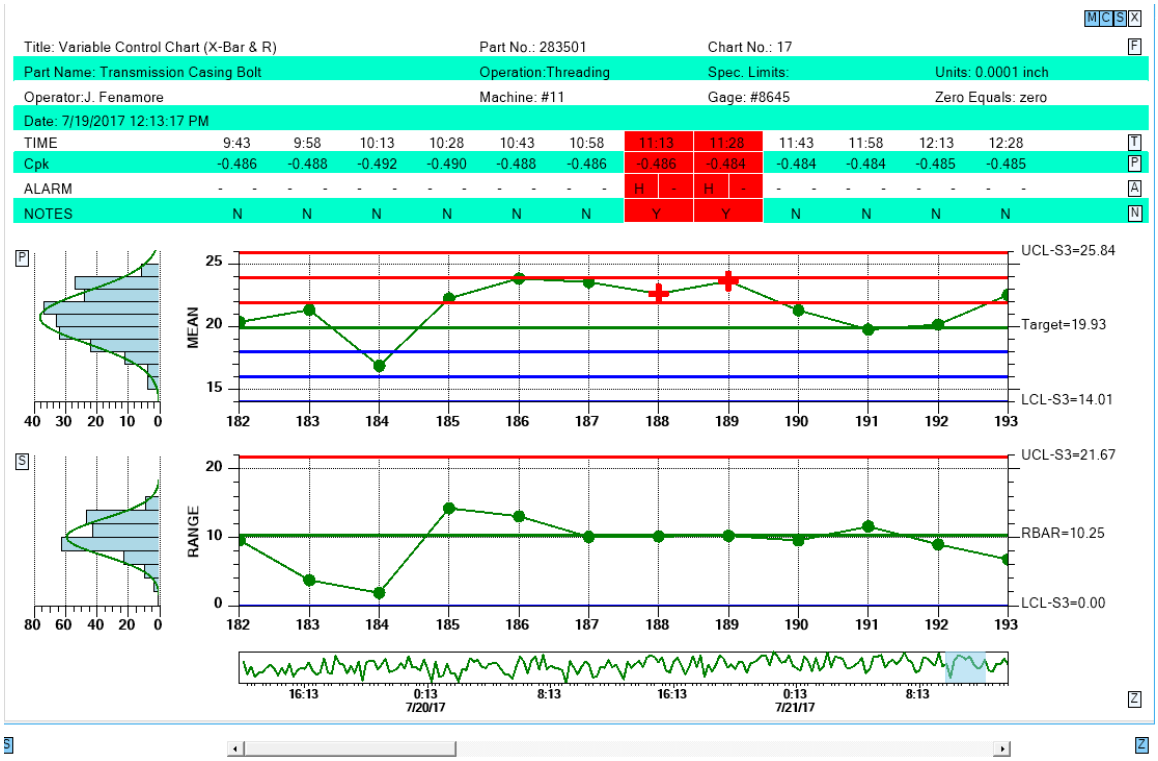
Using the default N of M testing, the sample interval 101 fails the 2 out of 3 > 2-sigma test of the WECO and Nelson rules.

If the following code was added to the chart setup

```
QCSPCChartTS.SPCControlLimitRecord.setDefaultAltNofMRule(true);
```

sample 101 would NOT be in alarm, even though the previous two samples were > 2-sigma.

In the picture below, DefaultAltNofMRule property has been set true. Using the regular, default rules, the sample interval at 190 would be considered in alarm because 4 out of 5 samples intervals were greater than 1-sigma. But since the alternative evaluation method for N of M rules is enabled, it is shown as NOT being in alarm, because it is within 1-sigma.



Using the Alternative N of M evaluation method, sample interval 190 does not show an alarm for the 4 out of 5 > 1-sigma test, because it is within 1-sigma.

Control Limit Alarm Event Handling

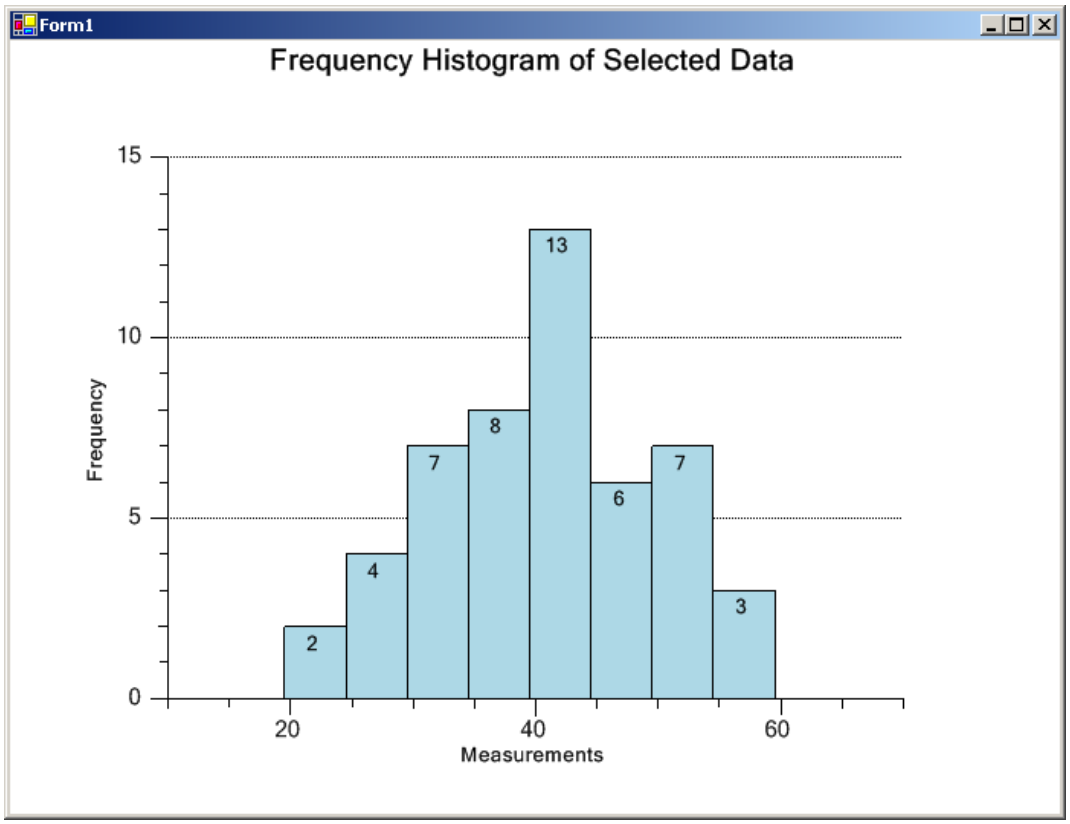
The **SPCControlChartData** class can throw an alarm event based on either the current alarm state, or an alarm transition from one alarm state to another. The **SPCControlLimitAlarmArgs** passes alarm data to the event handler. See *Chapter 5 - SPC Control Data and Alarm Classes,* for examples of using an alarm event handler.

Chapter 9 - Frequency Histogram, Normal Probability and Pareto Diagram Charts

FrequencyHistogramChart
ProbabilityChart
ParetoChart

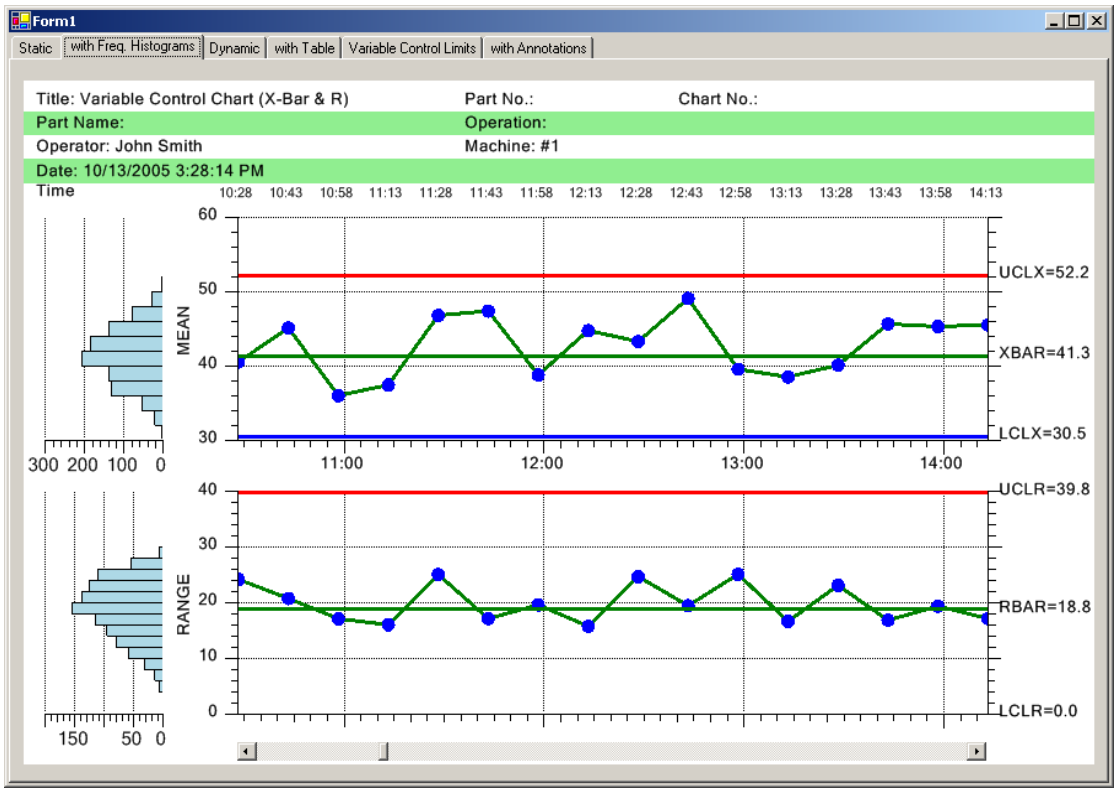
Frequency Histogram Chart

An SPC control chart will allow you to track the trend of critical variables in a production environment. It is important that the production engineer understand whether or not changes or variation in the critical variables are natural variations due to the tolerances inherent to the production machinery, or whether or not the variations are due to some systemic, assignable cause that needs to be addressed. If the changes in critical variables are due to the natural variations, then a frequency histogram of the variations will usually follow one of the common continuous (normal, exponential, gamma, Weibull) or discrete (binomial, Poisson, hypergeometric) distributions. It is the job of the SPC engineer to know what distribution best models his process. Periodically plotting of the variation of critical variables will give SPC engineer important information about the current state of the process. A typical frequency histogram looks like:



Frequency Histogram Chart

Viewing frequency histograms of both the variation in the primary variable (Mean, Median, count, or actual value), and the secondary variable (Range, Sigma or Moving Range) side-by-side with the SPC control chart makes it even easier to find out whether the variations are the result of natural variations or the result of some systemic change in the process.



XBar-R Chart with Integral Frequency Histograms

Creating an Independent (not part of a SPC chart) Frequency Histogram

The **FrequencyHistogramChart** class creates a standalone frequency histogram. It is a simple template where you need only supply data and set a few properties to create a proper frequency histogram. The example below extracted from the **FrequencyHistogram.FrequencyHistogramUserControl1** example program.

[JavaScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';


export async function BuildFrequencyHistogramChart(canvasid) {
        var htmlcanvas = document.getElementById(canvasid);
        var charttitle = "Frequency Histogram Example";


        var frequencyHistogramchart =
QCSPCChartTS.FrequencyHistogramChart.newFrequencyHistogramChart(htmlcanvas);
        if (!frequencyHistogramchart) return;

       // Frequency bins
        var freqLimits = [19.5, 24.5, 29.5, 34.5, 39.5, 44.5, 49.5, 54.5, 59.5];
        // data to be sorted into frequency bins
```

```
        var freqValues = [32,44,44,42,57,
                                  26,51,23,33,27,
                                  42,46,43,45,44,
                                  53,37,25,38,44,
                                  36,40,36,48,56,
                                  47,40,58,45,38,
                                  32,39,43,31,45,
                                  41,37,31,39,33,
                                  20,50,33,50,51,
                                  28,51,40,52,43];


        frequencyHistogramchart.initFrequencyHistogram(freqLimits, freqValues);

        frequencyHistogramchart.setPreferredSize(800, 600);

              // Set bar orientation
            frequencyHistogramchart.getMainTitle().setTextString("Frequency
Histogram of Selected Data");
           // Build chart


frequencyHistogramchart.setChartOrientation( QCSPCChartTS.ChartConstants.VERT_DIR)
;

    frequencyHistogramchart.setBarFillColor(QCSPCChartTS.ChartColor.LIGHTCORAL);


frequencyHistogramchart.getFrequencyHistogramPlot().setSegmentFillColor(4,QCSPCCha
rtTS.ChartColor.BLUE);
           frequencyHistogramchart.addFrequencyHistogramControlLine(20.0,
QCSPCChartTS.ChartAttribute.newChartAttributeColorWidth(QCSPCChartTS.ChartQCSPCCha
rtTS.ChartColor.LIGHTGREEN, 2));
           frequencyHistogramchart.addFrequencyHistogramControlLine(60.0,
QCSPCChartTS.ChartAttribute.newChartAttributeColorWidth(QCSPCChartTS.ChartQCSPCCha
rtTS.ChartColor.LIGHTGREEN, 2));
           frequencyHistogramchart.setAutoNormalCurve(true);

            frequencyHistogramchart.buildChart();
            frequencyHistogramchart.updateDraw();

    }
```

[TypeScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';


export class FrequencyHistogramChart {

    public constructor() {

    }

    public async  BuildFrequencyHistogramChart(canvasid: string) {

        let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
        let charttitle: string = "Frequency Histogram Example";


        let frequencyHistogramchart: QCSPCChartTS.FrequencyHistogramChart  =
QCSPCChartTS.FrequencyHistogramChart.newFrequencyHistogramChart(htmlcanvas);
        if (!frequencyHistogramchart) return;

       // Frequency bins
        let freqLimits: number[] = [19.5, 24.5, 29.5, 34.5, 39.5, 44.5, 49.5,
54.5, 59.5];
        // data to be sorted into frequency bins
        let freqValues: number[] = [32,44,44,42,57,
```

```
                                    26,51,23,33,27,
                                    42,46,43,45,44,
                                    53,37,25,38,44,
                                    36,40,36,48,56,
                                    47,40,58,45,38,
                                    32,39,43,31,45,
                                    41,37,31,39,33,
                                    20,50,33,50,51,
                                    28,51,40,52,43];


        frequencyHistogramchart.initFrequencyHistogram(freqLimits, freqValues);

        frequencyHistogramchart.setPreferredSize(800, 600);

               // Set bar orientation
            frequencyHistogramchart.getMainTitle().setTextString("Frequency
Histogram of Selected Data");
           // Build chart


frequencyHistogramchart.setChartOrientation( QCSPCChartTS.ChartConstants.VERT_DIR)
;

    frequencyHistogramchart.setBarFillColor(QCSPCChartTS.ChartColor.LIGHTCORAL);


frequencyHistogramchart.getFrequencyHistogramPlot().setSegmentFillColor(4,QCSPCCha
rtTS.ChartColor.BLUE);
           frequencyHistogramchart.addFrequencyHistogramControlLine(20.0,
QCSPCChartTS.ChartAttribute.newChartAttributeColorWidth(QCSPCChartTS.ChartQCSPCCha
rtTS.ChartColor.LIGHTGREEN, 2));
           frequencyHistogramchart.addFrequencyHistogramControlLine(60.0,
QCSPCChartTS.ChartAttribute.newChartAttributeColorWidth(QCSPCChartTS.ChartQCSPCCha
rtTS.ChartColor.LIGHTGREEN, 2));
           frequencyHistogramchart.setAutoNormalCurve(true);

            frequencyHistogramchart.buildChart();
            frequencyHistogramchart.updateDraw();

    }

}
```

All you have to do is supply the raw data, and the values of the frequency bins for which you want to accumulate values. The **FrequencyHistogramChart** class auto-scale a coordinate system, creates the proper x- and y-axes, and draws the resulting frequency histogram as a bar plot.

FrequencyHistogramChart.initFrequencyHistogram Method

Initializes the histogram frequency bin limits, and the data values for the histogram.

```
public static newFrequencyHistogramChart(context: Canvas): FrequencyHistogramChart

public static newFrequencyHistogramChartwValues(context: Canvas, frequencylimits:
number[], frequencyvalues: number[]): FrequencyHistogramChart

public initFrequencyHistogram(frequencylimits: number[], frequencyvalues:
number[])
```

Parameters

context

The canvas the chart is placed in

frequencylimits
The frequency limits of the histogram bins.
frequencyvalues
An array the values that are counted with respect to the frequency bins.

The image below uses the following data:

[JavaScript]

```
// Frequency bins
        var freqLimits = [19.5, 24.5, 29.5, 34.5, 39.5, 44.5, 49.5, 54.5, 59.5];
        // data to be sorted into frequency bins
        var freqValues = [32,44,44,42,57,
                                26,51,23,33,27,
                                42,46,43,45,44,
                                53,37,25,38,44,
                                36,40,36,48,56,
                                47,40,58,45,38,
                                32,39,43,31,45,
                                41,37,31,39,33,
                                20,50,33,50,51,
                                28,51,40,52,43];


        frequencyHistogramchart.initFrequencyHistogram(freqLimits, freqValues);
```

[TypeScript]

```
        let freqLimits: number[] = [19.5, 24.5, 29.5, 34.5, 39.5, 44.5, 49.5,
54.5, 59.5];
        // data to be sorted into frequency bins
        let freqValues: number[] = [32,44,44,42,57,
                                26,51,23,33,27,
                                42,46,43,45,44,
                                53,37,25,38,44,
                                36,40,36,48,56,
                                47,40,58,45,38,
                                32,39,43,31,45,
                                41,37,31,39,33,
                                20,50,33,50,51,
                                28,51,40,52,43];


        frequencyHistogramchart.initFrequencyHistogram(freqLimits, freqValues);
```

Once the init routine is called, the chart can be further customized using the properties and methods below.

Note - Since JavaScript/TypeScript does not support properties in the same way that C# does, you set and get the values of the properties using the “set” and “get” in front of the property name, i.e. **getAutoNormalCurve** and **setAutoNormalCurve.**

Public Static Properties

DefaultChartFontString	set/get the default font used in the chart. This is a string specifying the name of the font.

Public Instance Constructors

FrequencyHistogramChart	Initializes a new instance of the FrequencyHistogramChart class.

Public Instance Properties

AutoNormalCurve	set/get to true and a normal curve with the same area as the histogram is plotted in the chart
AutoScaleFlag	set/get to true to enable auto-scaling of the chart.
AutoScaleXDataMode	set/get a value controlling what items are included in the x-axis auto-scaling: : 0 = histogram data only, 1=normal curve only, 2 = control limit lines only, 3 = all three
AutoScaleYDataMode	set/get a value controlling what items are included in the x-axis auto-scaling: 0 = histogram data only, 1=normal curve only, 2 = both
BarAttrib	set/get the primary bar attribute object for the bars of the histogram.
BarDataValue	set/get the numeric label template object used to place numeric values on the bars.
BarFillColor	set/get the fill color for the chart object.
BarLineColor	set/get the line color for the chart object.
BarLineWidth	set/get the line width for the chart object.
ChartOrientation	set/get the orientation of the histogram bars in the chart.
CoordinateSystem	set/get the coordinate system object for the histogram.
DefaultAxisLabelsFont	set/get the default font used for the axes labels and axes titles.
DefaultDataValueFont	set/get the default font used for the numeric values labeling the bars.
DefaultFooterFont	set/get the font used for the chart footer.
DefaultMainTitleFont	set/get the font used for the main title.
DefaultSubHeadFont	set/get the font used for the main title.
DefaultToolTipFont	set/get the default font object used for the tooltip.

Footer	set/get the charts footer object
GraphBorder	set/get the default graph border for the chart.
HistogramDataset	set/get the GroupDatset object that holds the data used to plot the histogram.
MainTitle	set/get the main title object for the chart.
NormalCurveX	Returns the array of x-values of the normal curve array
NormalCurveY	Returns the array of y-values of the normal curve array
PlotBackground	set/get the plot background object.
SubHead	set/get the subhead title object for the chart.
XAxis	set/get the x-axis object.
XAxisLab	set/get the x-axis labels object.
XAxisTitle	set/get the x-axis title object.
XGrid	set/get the x-axis grid object.
YAxis	set/get the y-axis object.
YAxisLab	set/get the y-axis labels object. Accessible only after BuildGraph
YAxisTitle	set/get the y-axis title object.
YGrid	set/get the y-axis grid object.

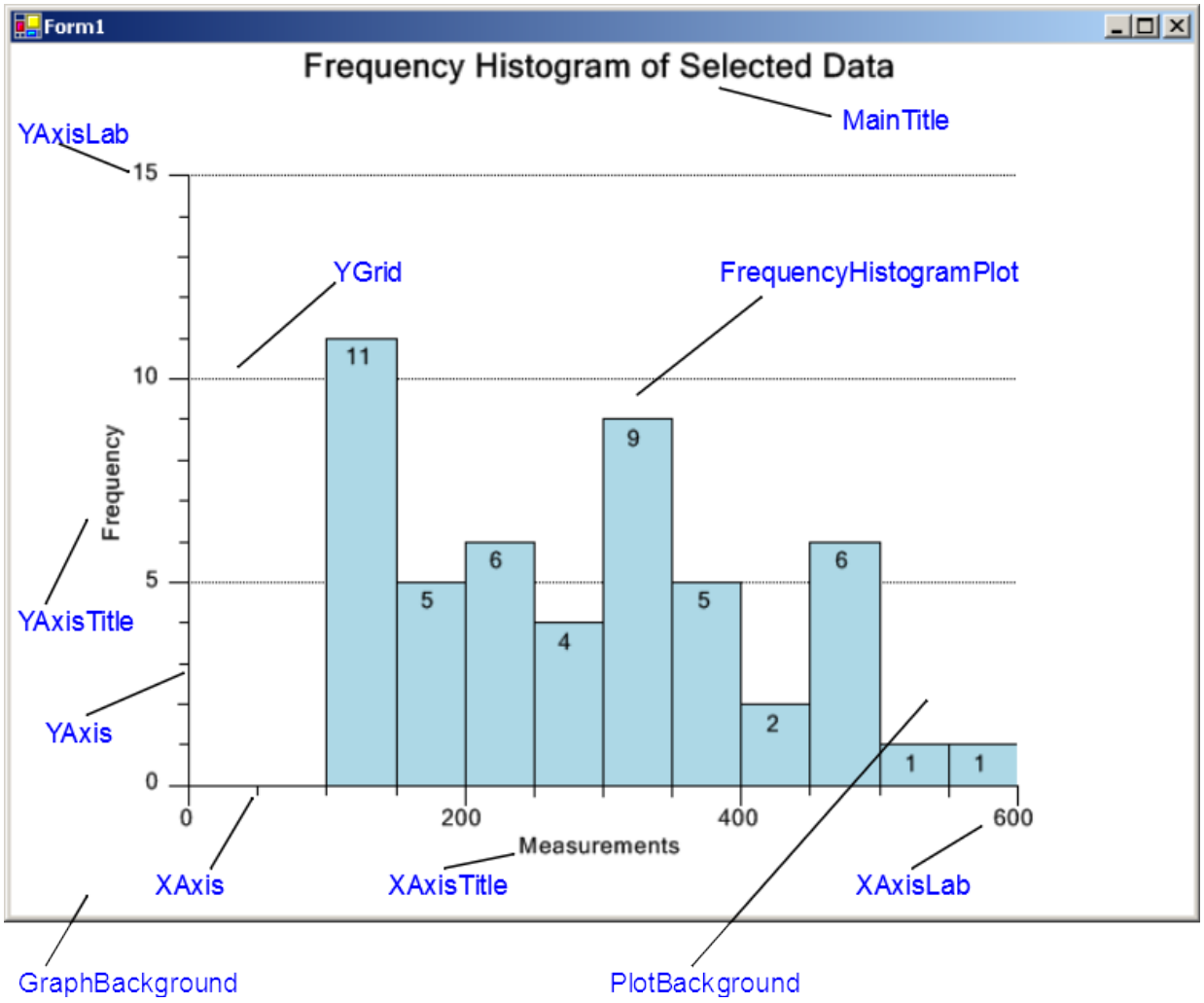
Public Instance Functions

addFrequencyHistogramControlLine	Add a control limit line to the frequency histogram
initFrequencyHistogram	Initializes the histogram frequency bin limits, and the data values to be analyzed for the histogram.

The FrequencyHistogramChart properties are documented in the QCSPCChartJSTSClassesIndex.html documentation file, located in the docs/docs/ subdirectory.

Changing Default Characteristics of the Chart

A **FrequencyHistogramChart** object has one distinct graph with its own set of properties. Once the graph is initialized (using the **initFrequencyHistogram**, or one of the **FrequencyHistogramChart** constructors**)**, you can modify the default characteristics of each graph using these properties. For example, you can change the color of y-axis, and the y-axis labels using the LineColor property of those objects.



[JavaScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';


export async function BuildFrequencyHistogramChart(canvasid) {
        var htmlcanvas = document.getElementById(canvasid);
        var charttitle = "Frequency Histogram Example";


        var frequencyHistogramchart =
QCSPCChartTS.FrequencyHistogramChart.newFrequencyHistogramChart(htmlcanvas);
        if (!frequencyHistogramchart) return;

       // Frequency bins
        var freqLimits = [19.5, 24.5, 29.5, 34.5, 39.5, 44.5, 49.5, 54.5, 59.5];
        // data to be sorted into frequency bins
        var freqValues = [32,44,44,42,57,
                                26,51,23,33,27,
                                42,46,43,45,44,
                                53,37,25,38,44,
                                36,40,36,48,56,
                                47,40,58,45,38,
                                32,39,43,31,45,
```

```
                                41,37,31,39,33,
                                20,50,33,50,51,
                                28,51,40,52,43];


        frequencyHistogramchart.initFrequencyHistogram(freqLimits, freqValues);

        frequencyHistogramchart.setPreferredSize(800, 600);


frequencyHistogramchart.getYAxis().setLineColor(QCSPCChartTS.ChartColor.GREEN);
frequencyHistogramchart.getYAxis().setLineWidth(3);

frequencyHistogramchart.getYAxisLab().setLineColor(QCSPCChartTS.ChartColor.DARKMAG
ENTA);

            frequencyHistogramchart.buildChart();
            frequencyHistogramchart.updateDraw();
}
```

[TypeScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';


export class FrequencyHistogramChart {

    public constructor() {

    }

    public async  BuildFrequencyHistogramChart(canvasid: string) {

        let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
        let charttitle: string = "Frequency Histogram Example";


        let frequencyHistogramchart: QCSPCChartTS.FrequencyHistogramChart  =
QCSPCChartTS.FrequencyHistogramChart.newFrequencyHistogramChart(htmlcanvas);
        if (!frequencyHistogramchart) return;

       // Frequency bins
        let freqLimits: number[] = [19.5, 24.5, 29.5, 34.5, 39.5, 44.5, 49.5,
54.5, 59.5];
        // data to be sorted into frequency bins
        let freqValues: number[] = [32,44,44,42,57,
                                26,51,23,33,27,
                                42,46,43,45,44,
                                53,37,25,38,44,
                                36,40,36,48,56,
                                47,40,58,45,38,
                                32,39,43,31,45,
                                41,37,31,39,33,
                                20,50,33,50,51,
                                28,51,40,52,43];


        frequencyHistogramchart.initFrequencyHistogram(freqLimits, freqValues);

        frequencyHistogramchart.setPreferredSize(800, 600);

frequencyHistogramchart.getYAxis().setLineColor(QCSPCChartTS.ChartColor.GREEN);
frequencyHistogramchart.getYAxis().setLineWidth(3);

frequencyHistogramchart.getYAxisLab().setLineColor(QCSPCChartTS.ChartColor.DARKMAG
ENTA);

            frequencyHistogramchart.buildChart();
            frequencyHistogramchart.updateDraw();
```

```
}
```

Special Considerations

1. The **FrequencyHistogramChart** class uses the **QCChart2D HistogramPlot** plot object class to draw the histogram bars. That class uses individually assignable colors for each bar of the bar plot. The standard **setLineColor** and **setFillColor** properties do not work to change the color of the histogram bars in this case. Instead, you can change the histogram bar colors by calling **setSegmentLineColor** and **setSegmentFillColor**.

[JavaScript]

```
var i = 0;
for ( i=0; i < 9; i++)
{

frequencyHistogramchart.getFrequencyHistogramPlot().setSegmentFillColor(i,QCSPCCha
rtTS.ChartColor.BLUE);

frequencyHistogramchart.getFrequencyHistogramPlot().setSegmentLineColor(i,QCSPCCha
rtTS.ChartColor.BLACK);
}
```

[TypeScript]

```
let i: number = 0;
for ( i=0; i < 9; i++)
{

frequencyHistogramchart.getFrequencyHistogramPlot().setSegmentFillColor(i,QCSPCCha
rtTS.ChartColor.BLUE);

frequencyHistogramchart.getFrequencyHistogramPlot().setSegmentLineColor(i,QCSPCCha
rtTS.ChartColor.BLACK);
}
```

You can also use the utility properties, **BarFillColor**, **BarLineColor** and **BarLineWidth**, we added to the **FrequencyHistogramPlot** that will set all of the bars of the histogram plot at once.

[JavaScript / TypeScript]

```
  frequencyHistogramchart.setBarFillColor(QCSPCChartTS.ChartColor.BLUE);
  frequencyHistogramchart.setBarLineColor(QCSPCChartTS.ChartColor.BLACK);
```

Adding Control Lines and Normal Curve to Histogram Plot

You can add control limit alarm lines to the histogram plot. The control limit lines will be parallel to the frequency axis. Second, a normal distribution curve can be overlaid on top of the histogram data. The parameters are selected to give the normal distribution curve

the same mean, standard deviation and area as the underlying histogram data. If the underlying data is normal, then there should be a relatively close fit between the normal curve and the underlying frequency data.

Histogram Control Limit Lines and Normal Curve fit

[JavaScript / TypeScript]

```
frequencyHistogramchart.addFrequencyHistogramControlLine(20.0,QCSPCChartTS.ChartAt
tribute.newChartAttributeColorWidth (QCSPCChartTS.ChartColor.LIGHTGREEN, 2));

frequencyHistogramchart.addFrequencyHistogramControlLine(60.0,QCSPCChartTS.ChartAt
tribute.newChartAttributeColorWidth (QCSPCChartTS.ChartColor.LIGHTGREEN, 2));

frequencyHistogramchart.setAutoNormalCurve( true);
```

Auto-Scaling

There are three elements which you might want to consider for auto-scaling the frequency histogram chart: The actual histogram bars, the calculated normal curve, and any added limit lines. First you must have AutoScaleFlag set to true. The default mode will auto-scale to the histogram bars only. If you want to auto-scale to the fitted normal curve, set the AutoScaleXDataMode, and AutoScaleYDataMode to 1. If you want to scale the x-axis to the added limit lines, use AutoScaleXDataMode = 2. If you want to take all into account, set AutoScaleXDataMode = 3, and AutoScaleYDataMode=2.

AutoScaleYDataMode	set/get a value controlling what items are included in the x-axis auto-scaling: 0 = histogram data only, 1=normal curve only, 2 = both
AutoScaleXDataMode	set/get a value controlling what items are included in the x-axis auto-scaling: : 0 = histogram data only, 1=normal curve only, 2 = control limit lines only, 3 = all three

```
frequencyHistogramchart.AutoScaleYDataMode = 2;
frequencyHistogramchart.AutoScaleXDataMode = 3;
```

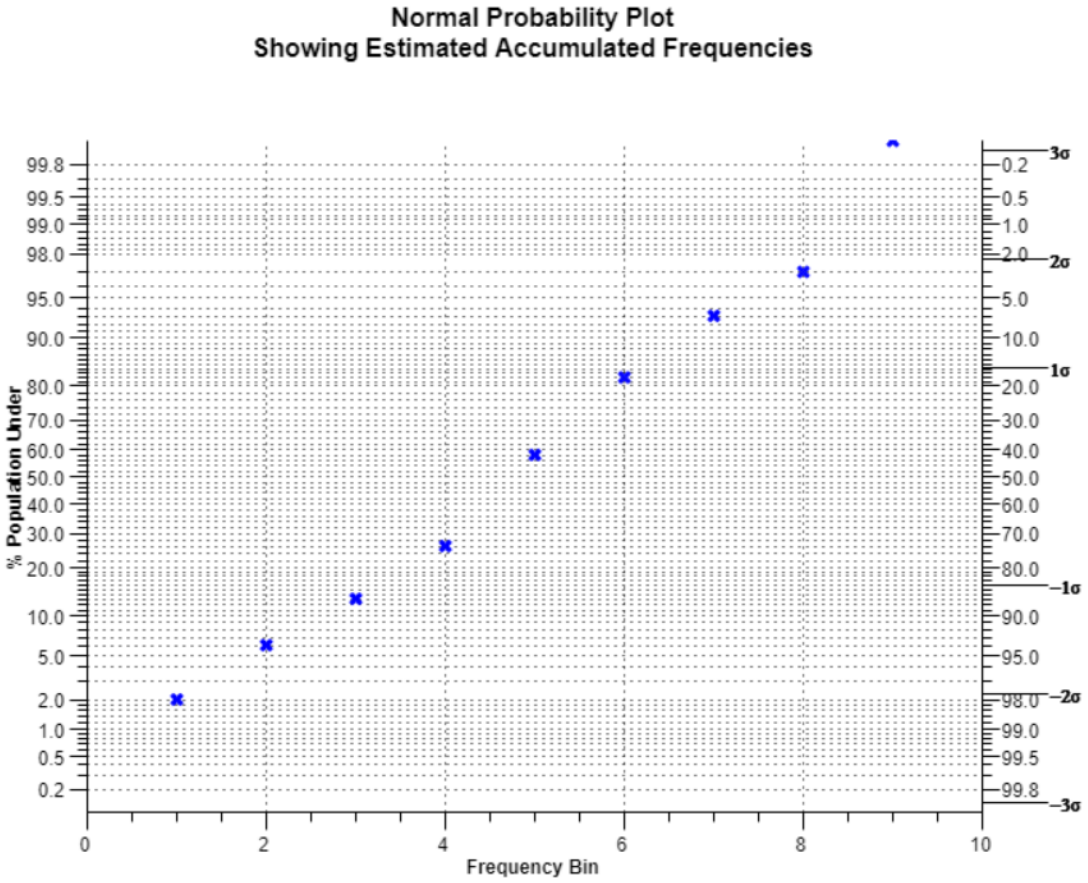
Normal Probability Plots

Another important tool the SPC engineer uses to model the process variation is the probability plot. The probability plot is a simple way to test whether control chart measurements fit a normal distribution. Usually probability plot graphs are plotted by

hand using special probability plot graph paper. We have added probability scale and axis classes that enable you to plot probability plots directly on the computer. Control chart measurements that follow a normal distribution curve will plot as a straight line when plotted in a normal probability plot.

Creating a Probability Plot

Cumulative Normal Probability Chart



The **ProbabilityChart** class creates a standalone probability plot. It is a simple template where you need only supply data and set a few properties to create a proper frequency histogram. The example below is extracted from the **ProbabilityChart.ProbabilityChart** example program.

[JavaScript]

```
export class ProbabilityChart {
```

```
    constructor() {
        // Frequency bins
        this.freqLimits = [19.5, 24.5, 29.5, 34.5, 39.5, 44.5, 49.5, 54.5, 59.5];
        // data to be sorted into frequency bins
        this.freqValues = [32, 44, 44, 42, 57,
            26, 51, 23, 33, 27,
            42, 46, 43, 45, 44,
            53, 37, 25, 38, 44,
            36, 40, 36, 48, 56,
            47, 40, 58, 45, 38,
            32, 39, 43, 31, 45,
            41, 37, 31, 39, 33,
            20, 50, 33, 50, 51,
            28, 51, 40, 52, 43];
    }
    async BuildProbabilityChart(canvasid) {
        let htmlcanvas = document.getElementById(canvasid);
        let charttitle = "Probability Example";
        let probchart =
QCSPCChartTS.ProbabilityChart.newProbabilityChart(htmlcanvas);
        if (!probchart)
            return;
        probchart.setPreferredSize(800, 600);
        //    init the Probability chart with the raw  frequencies of the dataset
        probchart.initProbabilityChartRaw(this.freqLimits, this.freqValues);
        probchart.MainTitle.TextString = "Normal Probability Plot";
        probchart.MainTitle.addNewLineTextString("Showing Estimated Accumulated
Frequencies");

        probchart.buildChart();
        probchart.updateDraw();
    }
}
```

[TypeScript]

```
export class ProbabilityChart {
          // Frequency bins
     freqLimits: number[] = [19.5, 24.5, 29.5, 34.5, 39.5, 44.5, 49.5, 54.5,
59.5];
     // data to be sorted into frequency bins
     freqValues: number[] = [32,44,44,42,57,
                             26,51,23,33,27,
                             42,46,43,45,44,
                             53,37,25,38,44,
                             36,40,36,48,56,
                             47,40,58,45,38,
                             32,39,43,31,45,
                             41,37,31,39,33,
                             20,50,33,50,51,
                             28,51,40,52,43];


     public constructor() {

     }

      public async BuildProbabilityChart(canvasid: string) {

          let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
          let charttitle: string = "Probability Example";

          let probchart: QCSPCChartTS.ProbabilityChart =
QCSPCChartTS.ProbabilityChart.newProbabilityChart(htmlcanvas);
          if (!probchart) return;

          probchart.setPreferredSize(800, 600);

     //    init the Probability chart with the raw  frequencies of the dataset
```

```
          probchart.initProbabilityChartRaw(this.freqLimits, this.freqValues);

          probchart.MainTitle.TextString = "Normal Probability Plot";
          probchart.MainTitle.addNewLineTextString("Showing Estimated Accumulated
Frequencies");

          probchart.ProbabilityPlot.setColor(QCSPCChartTS.ChartColor.RED); // sets
both line and fill color
          probchart.ProbabilityPlot.ChartObjAttributes.SymbolSize = 12;
          probchart.YAxis1.LineColor = QCSPCChartTS.ChartColor.GREEN;
          probchart.YAxis1.LineWidth = 3;
          probchart.ProbabilityPlot.LineColor = QCSPCChartTS.ChartColor.RED;
          probchart.ProbabilityPlot.ChartObjAttributes.SymbolSize = 12;

          probchart.YAxis2.LineColor = QCSPCChartTS.ChartColor.BLUE;
          probchart.YAxis2.LineWidth = 3;
          probchart.YAxisLab1.LineColor = QCSPCChartTS.ChartColor.DARKMAGENTA;

          probchart.buildChart();
          probchart.updateDraw();
     }
}
```

All you have to do is supply the raw data, and the limit values of the frequency bins for which you want to accumulate values. The **ProbabilityChart** class auto-scale a coordinate system, creates the proper x- and y-axes, and draws the resulting probability plot as a scatter plot.

The x-values represent the x-values of the frequency bins. The y-values can take several different forms. Internally, the software process the cumulative, normalized, value of the frequency bins. But, you can provide the source data in many different forms.

1. **Raw sample data format**– use **initProbabilityChartRaw**(xvalues: number[], yvalues: number[]) - The raw y-values of the raw data, same as if you were calling the initializing the FrequencyHistogramChart class. The software will calculate the frequency histogram of the raw data. Using the number of values in each frequency bin, an array of the cumulative sums of the frequency bins is created. And last, that cumulative frequency bin array is normalized to the range 0.0 to 1.0.

This method uses data in exactly the same format as the FrequencyHistogramChart uses.

```
freqLimits: number[] = [19.5, 24.5, 29.5, 34.5, 39.5, 44.5, 49.5, 54.5, 59.5];
// data to be sorted into frequency bins
freqValues: number[] = [32,44,44,42,57,
                        26,51,23,33,27,
                        42,46,43,45,44,
                        53,37,25,38,44,
                        36,40,36,48,56,
                        47,40,58,45,38,
                        32,39,43,31,45,
                        41,37,31,39,33,
                        20,50,33,50,51,
                        28,51,40,52,43];

probchart.initProbabilityChartRaw(freqLimits, freqValues)
```

2. **Histogram data format** - use **initProbabilityChart**(xvalues: number[], histogramvalues: number[]) - This method assumes that the data is supplied already converted to frequency histogram data. But it is not cumulative, and not normalized.

```
freqLimits: number[] = [19.5, 24.5, 29.5, 34.5, 39.5, 44.5, 49.5, 54.5, 59.5];
// data to be sorted into frequency bins
freqValues: number[] = [32,44,44,42,57,
                        26,51,23,33,27,
                        42,46,43,45,44,
                        53,37,25,38,44,
                        36,40,36,48,56,
                        47,40,58,45,38,
                        32,39,43,31,45,
                        41,37,31,39,33,
                        20,50,33,50,51,
                        28,51,40,52,43];

let frequencyHistogramchart: QCSPCChartTS.FrequencyHistogramChart = new
QCSPCChartTS.FrequencyHistogramChart ();
frequencyHistogramchart.initFrequencyHistogram(this.freqLimits, this.freqValues);

// get the values of the frequency bins (not cumulative, not normalized) of the frequency
histogram
let datavalues : QCSPCChartTS.SimpleDataset =
frequencyHistogramchart.getFrequencyHistogramDataset();
let x1: number[] = datavalues.getXData();
let y1: number[] = datavalues.getYData();
probchart.initProbabilityChart(this.freqLimits, y1);
```

The example above uses the FrequencyHistogramChart to transform the raw data into an array of histogram values (*y1*). You can also specify the histogram data directly, using a simple number array.

```
let y1: number[] = [2,4,7,8,13,6,7,3]
```

3. **Cumlative histogram data format** - use **initProbabilityChartCumulative**(xvalues: number[], cumyvalues: number[])- This method assumes that the data is supplied already converted to frequency histogram data. It also needs to be converted to a cumulative format. But it is not normalized.

Now, if you were to convert the original histogram values to a cumulative histogram array, the first array element would be the same, 2. The second element would be the sum of element 0 and 1. The third element would be the sum of element 0, 1, and 2. And so on to the last element, which would be the sum of all elements in the original histogram value array. So, starting with the histogram data from above, [2,4,7,8,13,6,7,3], the resulting cumulative histogram array would look like:

cumulative histogram values (cumyvalues) = [2,6,13,21,34,40,47,50];

Note that the last value (element 7, is 50, which is the same as the total number of data values in the original freqValues source array.

```
let freqLimits: number[] = [19.5, 24.5, 29.5, 34.5, 39.5, 44.5, 49.5, 54.5, 59.5];
```

```
let y1: number[] = [2,6,13,21,34,40,47,50];
probchart.initProbabilityChartCumulative(freqLimits, y1);
```

4. **Histogram data cumulative normalized format** - use **initProbabilityChartCumulativeNormalized**(xvalues: number[], cumnormyvalues: number[])- This method assumes that the data is supplied already converted to a cumlative, normalized, frequency histogram format.

To normalize the data to the range 0.0 to 1.0, you divide each value of the cumulative histogram array by the last value in the cumulative histogram array, which is 50 in this case. The resulting normalized cumulative histogram array looks like:

normalized cumulative histogram values (cumnormyvalues) = [0.04, 0.12, 0.26, 0.42, 0.68, 0.80,0.94,1.0];

```
let freqLimits: number[] = [19.5, 24.5, 29.5, 34.5, 39.5, 44.5, 49.5, 54.5, 59.5];
```

```
let y1: number[] = [0.04, 0.12, 0.26, 0.42, 0.68, 0.80,0.94,1.0];
probchart.initProbabilityChartCumulativeNormalized(freqLimits, y1);
```

Parameters

context
: *The canvas the chart is placed in*

frequencylimits
: The frequency limits of the histogram bins.

frequencyvalues
: An array the raw data values that are counted with respect to the frequency bins.

histogramvalues
: An array the data values that represent the histogram data values of the frequency bins..

cumyvalues
: An array the data values that represent the cumulative version of the histogram values.

cumnormyvalues
: An array the data values that represent the cumulative, normalized version of the histogram values.

Public Static (Shared) Properties

DefaultAxisLabelsFont	Set/Get default font object used for the axes labels.
DefaultAxisTitleFont	Set/Get the default font object used for the

	axes titles. Set this properties BuildGraph.
DefaultChartFontString	Set/Get the default font used in the chart. This is a string specifying the name of the font.
DefaultFooterFont	Set/Get the default font object used for the chart footer.
DefaultMainTitleFont	Set/Get the default font object used for the main chart title.
DefaultSigmaFont	Set/Get the default font object used for the axis labels sigma character.
DefaultSubHeadFont	Set/Get the default font object used for the subhead title.
DefaultToolTipFont	Set/Get the default font object used for the tooltip.

Public Instance Constructors

ProbabilityChart	Overloaded. Initializes a new instance of the ProbabilityChart class.

Public Instance Properties

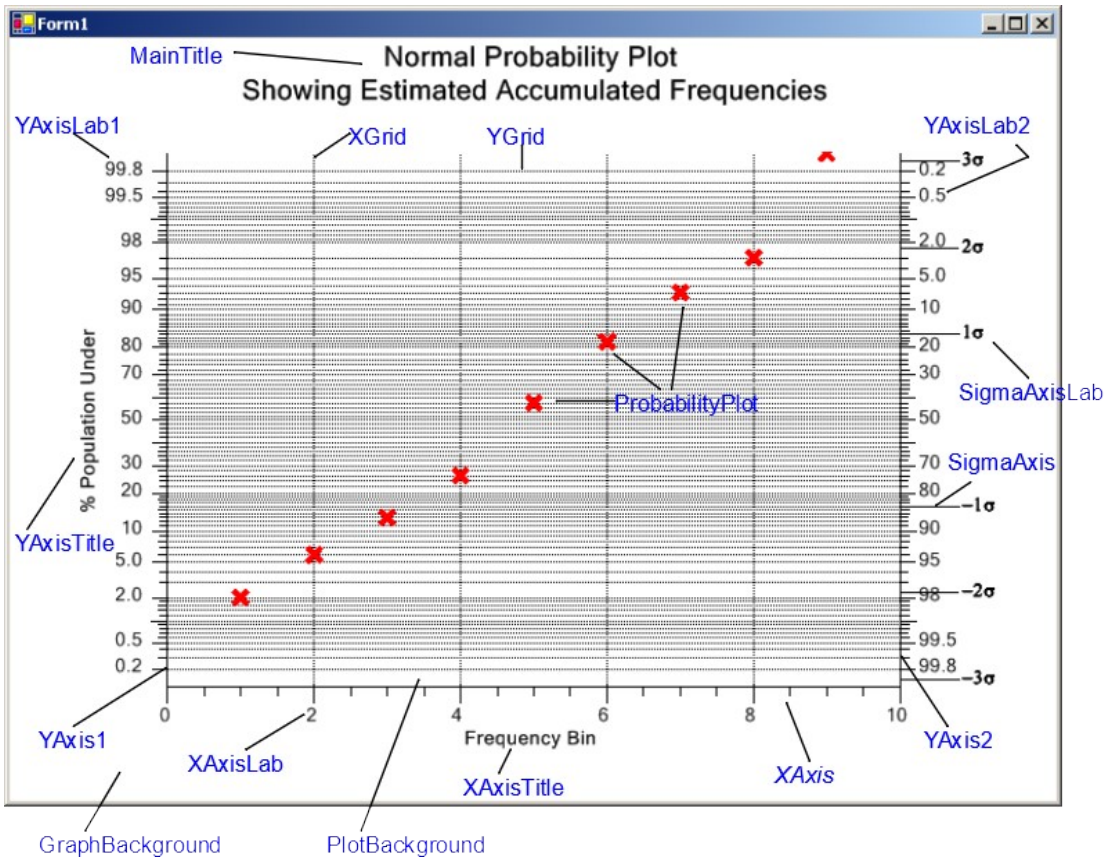
CoordinateSystem	Get the probability coordinate system for the chart.
Datatooltip	Get the chart data tooltip.
DefaultGraphBorder	Get/Set the default graph border object for the chart.
Footer	Get the chart footer object.
GraphBackground	Get the graph background object.
MainTitle	Get the chart title object.
PlotAttrib	Get the default primary plot attribute object for the plot of the chart. Set attributes before BuildChart.
PlotBackground	Get the plot background object.
ProbabilityDataset	Get the dataset holding the data values of the plot.
ProbabilityPlot	Get probability plot scatter plot object.
ResetOnDraw	Set/Get True the ChartView object list is cleared with each redraw
SigmaAxis	Get the sigma y-axis object of the chart.
SigmaAxisLab	Get the sigma y-axis labels object of the chart.
SubHead	Get the chart subhead object.
SymbolSize	Get/Set the default symbol size. Set attributes before BuildChart.
TextTemplate	Get the default text object template for the

	data tooltip.
ToolTipSymbol	Get the tooltip symbol object for the data tooltip.
XAxis	Get the x-axis object of the chart.
XAxisLab	Get the x-axis labels object of the chart.
XAxisTitle	Get the x-axis title object of the of the chart.
XGrid	Get the x-axis grid object of the of the chart.
XValues	Get the DoubleArray of the x-values of the data points plotted in the probability plot.
XValueTemplate	Get the default x-value object template for the data tooltip.
YAxis1	Get the left probability y-axis object of the chart.
YAxis2	Get the right probability y-axis object of the chart.
YAxisLab1	Get the left probability y-axis labels object of the chart.
YAxisLab2	Get the right probability y-axis labels object of the chart.
YAxisTitle	Get the y-axis title object of the of the chart.
YGrid	Get the y-axis title object of the of the chart.
YValues	Get the DoubleArray of the y-values of the data points plotted in the probability plot.
YValueTemplate	Get the default y-value object template for the data tooltip.

Public Instance Methods

BuildChart	Overloaded. Builds the probability chart using the base objects ChartView.
Copy	Overloaded. Copies the source ProbabilityChart object.
InitProbabilityChart	Initializes the x- and y-values of the data points plotted in the probability plot.
InitProbabilityDatasets	Builds the histogram dataset, histogramDataset, using the values in frequencyValues and frequencyLimits.

Changing Default Characteristics of the Chart



Once the graph is initialized (using the **initProbabilityChart** , or one of the **ProbabilityChart** constructors**)**, you can modify the default characteristics of each graph using these properties. For example, you can change the color of y-axis, and the y-axis labels using the **LineColor** property of those objects.

[JavaScript]

```
export class ProbabilityChart {
    constructor() {
        // Frequency bins
        this.freqLimits = [19.5, 24.5, 29.5, 34.5, 39.5, 44.5, 49.5, 54.5, 59.5];
        // data to be sorted into frequency bins
        this.freqValues = [32, 44, 44, 42, 57,
            26, 51, 23, 33, 27,
            42, 46, 43, 45, 44,
            53, 37, 25, 38, 44,
            36, 40, 36, 48, 56,
            47, 40, 58, 45, 38,
            32, 39, 43, 31, 45,
            41, 37, 31, 39, 33,
            20, 50, 33, 50, 51,
            28, 51, 40, 52, 43];
    }
    async BuildProbabilityChart(canvasid) {
        let htmlcanvas = document.getElementById(canvasid);
        let charttitle = "Probability Example";
        let probchart =
QCSPCChartTS.ProbabilityChart.newProbabilityChart(htmlcanvas);
```

```
        if (!probchart)
            return;
        probchart.setPreferredSize(800, 600);
        //    init the Probability chart with the raw  frequencies of the dataset
        probchart.initProbabilityChartRaw(this.freqLimits, this.freqValues);
        probchart.MainTitle.TextString = "Normal Probability Plot";
        probchart.MainTitle.addNewLineTextString("Showing Estimated Accumulated
Frequencies");

probchart.ProbabilityPlot.setColor(QCSPCChartTS.ChartColor.RED); // sets both line
and fill color
                probchart.ProbabilityPlot.ChartObjAttributes.SymbolSize = 12;
                probchart.YAxis1.LineColor = QCSPCChartTS.ChartColor.GREEN;
                probchart.YAxis1.LineWidth = 3;
                probchart.ProbabilityPlot.LineColor = QCSPCChartTS.ChartColor.RED;
                probchart.ProbabilityPlot.ChartObjAttributes.SymbolSize = 12;

                probchart.YAxis2.LineColor = QCSPCChartTS.ChartColor.BLUE;
                probchart.YAxis2.LineWidth = 3;
                probchart.YAxisLab1.LineColor =
QCSPCChartTS.ChartColor.DARKMAGENTA;
        probchart.buildChart();
        probchart.updateDraw();
    }
```

[TypeScript]

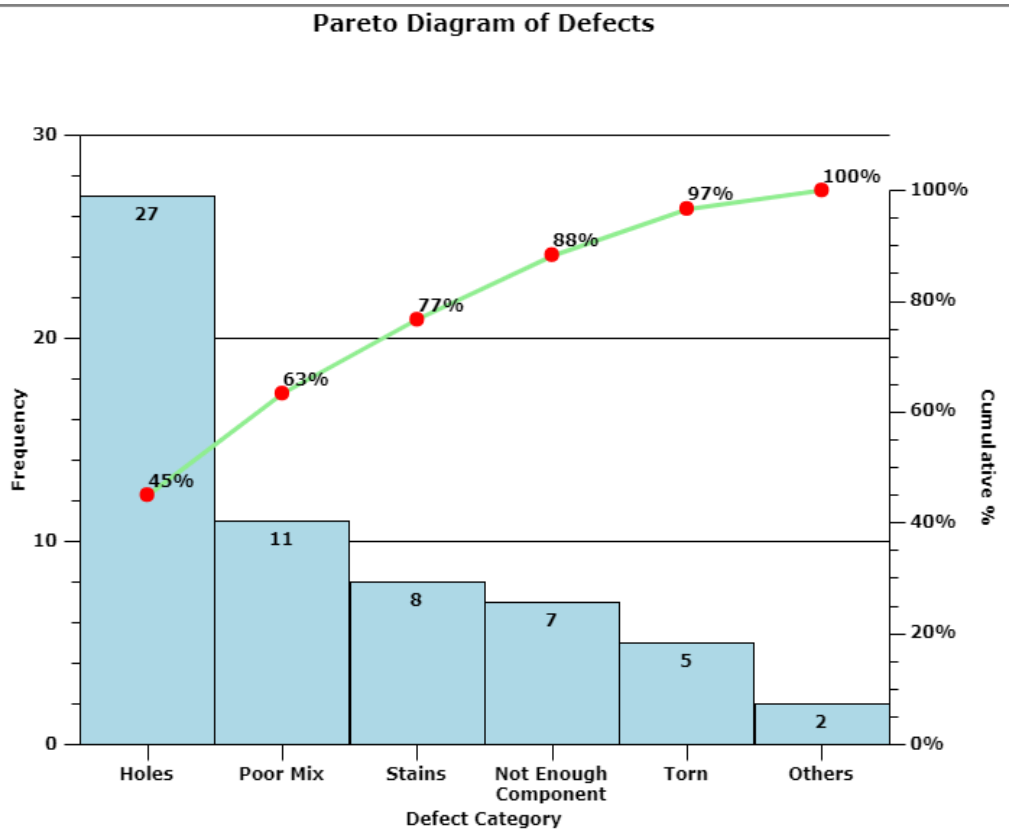
```
export class ProbabilityChart {
        // Frequency bins
          freqLimits: number[] = [19.5, 24.5, 29.5, 34.5, 39.5, 44.5, 49.5, 54.5,
59.5];
         // data to be sorted into frequency bins
          freqValues: number[] = [32,44,44,42,57,
                                  26,51,23,33,27,
                                  42,46,43,45,44,
                                  53,37,25,38,44,
                                  36,40,36,48,56,
                                  47,40,58,45,38,
                                  32,39,43,31,45,
                                  41,37,31,39,33,
                                  20,50,33,50,51,
                                  28,51,40,52,43];


    public constructor() {

    }




    public async BuildProbabilityChart(canvasid: string) {

        let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
        let charttitle: string = "Probability Example";

        let probchart: QCSPCChartTS.ProbabilityChart =
QCSPCChartTS.ProbabilityChart.newProbabilityChart(htmlcanvas);
        if (!probchart) return;

        probchart.setPreferredSize(800, 600);

    //    init the Probability chart with the raw  frequencies of the dataset
        probchart.initProbabilityChartRaw(this.freqLimits, this.freqValues);

        probchart.MainTitle.TextString = "Normal Probability Plot";
```

```
        probchart.MainTitle.addNewLineTextString("Showing Estimated Accumulated
Frequencies");

        probchart.ProbabilityPlot.setColor(QCSPCChartTS.ChartColor.RED); // sets
both line and fill color
        probchart.ProbabilityPlot.ChartObjAttributes.SymbolSize = 12;
        probchart.YAxis1.LineColor = QCSPCChartTS.ChartColor.GREEN;
        probchart.YAxis1.LineWidth = 3;
        probchart.ProbabilityPlot.LineColor = QCSPCChartTS.ChartColor.RED;
        probchart.ProbabilityPlot.ChartObjAttributes.SymbolSize = 12;

        probchart.YAxis2.LineColor = QCSPCChartTS.ChartColor.BLUE;
        probchart.YAxis2.LineWidth = 3;
        probchart.YAxisLab1.LineColor = QCSPCChartTS.ChartColor.DARKMAGENTA;

        probchart.buildChart();
        probchart.updateDraw();
    }
}
```

Pareto Diagrams

The Pareto diagram is special type of bar graph combined with a line plot, plotted in a chart that uses two different y-axis scales. The bar graph scale is a frequency scale that measures the number of times a specific problem has occurred. The line plot scale is a cumulative percentage scale.

Pareto Chart



The chart is easy to interpret. The tallest bar, the left-most one in a Pareto diagram, is the problem that has the most frequent occurrence. The shortest bar, the right-most one, is the problem that has the least frequent occurrence. Time spend on fixing the biggest problem will have the greatest affect on the overall problem rate. This is a simplistic view of actual Pareto analysis, which would usually take into account the cost effectiveness of fixing a specific problem. Never less, it is powerful communication tool that the SPC engineer can use in trying to identify and solve production problems.

Creating a Pareto Diagram

The **ParetoChart** class creates a standalone Pareto Diagram chart. It is a simple template where you need only supply data and set a few properties to create a proper frequency histogram. The example below is from the ParetoPlotUserControl1 file of the ParetoDiagram example program.

[JavaScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';

export async function BuildParetoChart(canvasid) {

        var htmlcanvas= document.getElementById(canvasid);
        var charttitle = "Pareto Example";


        var paretochart  =  QCSPCChartTS.ParetoChart.newParetoChart(htmlcanvas);
        if (!paretochart) return;


        paretochart.setPreferredSize(800, 600);


            paretochart.getMainTitle().setTextString("Pareto Diagram of Defects");
           // Build chart
           // add Pareto chart categories, values and strings
           paretochart.addCategoryItem(5, "Torn");
           paretochart.addCategoryItem(7, "Not Enough/nComponent");
           paretochart.addCategoryItem(2, "Others");
           paretochart.addCategoryItem(11, "Poor Mix");
           paretochart.addCategoryItem(27, "Holes");
           paretochart.addCategoryItem(8, "Stains");

            paretochart.getLineMarkerPlot().setShowDatapointValue(true);
            paretochart.buildChart();
            paretochart.updateDraw();

    }
```

[TypeScript]

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';


export class ParetoChart {

    public constructor() {
```

```
    }


    public async  BuildParetoChart(canvasid: string) {

        let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
        let charttitle: string = "Pareto Example";


        let paretochart: QCSPCChartTS.ParetoChart  =
QCSPCChartTS.ParetoChart.newParetoChart(htmlcanvas);
        if (!paretochart) return;


        paretochart.setPreferredSize(800, 600);


            paretochart.getMainTitle().setTextString("Pareto Diagram of Defects");
           // Build chart
           // add Pareto chart categories, values and strings
           paretochart.addCategoryItem(5, "Torn");
           paretochart.addCategoryItem(7, "Not Enough/nComponent");
           paretochart.addCategoryItem(2, "Others");
           paretochart.addCategoryItem(11, "Poor Mix");
           paretochart.addCategoryItem(27, "Holes");
           paretochart.addCategoryItem(8, "Stains");

            paretochart.getLineMarkerPlot().setShowDatapointValue(true);
            paretochart.buildChart();
            paretochart.updateDraw();

    }

}
```

All you have to do is supply the raw data, and the values of the frequency bins for which you want to accumulate values. The **ParetoChart** class auto-scale a coordinate system, creates the proper x- and y-axes, and draws the resulting probability plot as a scatter plot.

ParetoChart.initParetoChart Method

Initializes the x- and y-values of the data points plotted in the plot.

```
public static newParetoChart(context: Canvas ): ParetoChart

public static newParetoChartwValues(context: Canvas, categoryitems :number[] ,
stringitems: string[] ): ParetoChart

public initParetoChart( categoryitems: number [], stringitems: string [])
```

Parameters

context

The canvas the chart is placed in

categoryitems

The values for each category in the Pareto chart.

stringitems
The strings identifying each category in the Pareto chart.

You can add the category item values and string item strings one by one using the **addCategoryItem** method.

ParetoChart.addCategoryItem Method

Add an item to the categoryValues and categoryStrings arrays.

```
public addCategoryItem( itemfreq: number, itemstring: string): number
```

Parameters

itemfreq
The count of how many times this category has occurred.
itemstring
The string identifying the category item.

Note - Since JavaScript/TypeScript does not support properties in the same way that C# does, you set and get the values of the properties using the "set" and "get" in front of the property name, i.e. **getMainTitle** and **setMainTitle.**

Public Static Properties

DefaultChartFontString	set/get the default font used in the chart. This is a string specifying the name of the font.

Public Instance Constructors

ParetoChart	Initializes a new instance of the ParetoChart class.

Public Instance Properties

BarAttrib	set/get the default primary bar attribute object for the bars of the chart. Set attributes before buildChart.
BarDataValue	set/get the default numeric label template used to label the values of bar plot of the frequency histogram part of the chart. Set attributes before buildChart.
BarPlot	set/get the histogram bar plot object of the frequency histogram part of the chart.

BarWidth	set/get the default width value of the frequency histogram bars. Set attributes before buildChart.
CategoryStrings	set/get the StringArray object holding the strings used to label the categories of the Pareto plot. Set attributes before buildChart.
CategoryValues	set/get the DoubleArray object holding the category values used in building the Pareto plot. Set attributes before buildChart.
CoordinateSystem1	set/get the coordinate system object of the frequency histogram part of the chart.
CoordinateSystem2	set/get the coordinate system object of the cumulative frequency part of the chart.
CumulativeFreqDataset	set/get the dataset object used to hold the cumulative frequency values of the data plot.
Datatooltip	set/get the data tooltip object for the chart.
DefaultAxisLabelsFont	set/get default font object used for the axes labels and axes titles. Set attributes before buildChart.
DefaultFooterFont	set/get the default footer font. Set attributes before buildChart.
DefaultMainTitleFont	set/get the default chart title font. Set attributes before buildChart.
DefaultSubHeadFont	set/get the default chart title font. Set attributes before buildChart.
DefaultToolTipFont	set/get the default font object used for the tooltip.
Footer	set/get the charts footer object
LineMarkerPlot	set/get the line marker plot object displaying the cumulative frequency part of the chart.
LineMarkerPlotDataValue	set/get the default numeric template object used to label the line marker plot of the cumulative frequency part of the chart. Set attributes before buildChart.
MainTitle	set/get main title object of the chart.
PlotBackground	set/get the plot background object.
ResetOnDraw	set/get True the ChartView object list is cleared with each redraw
Scale2StartY	set/get the starting y-value for the cumulative frequency scale.
Scale2StopY	set/get the ending y-value for the cumulative frequency scale.
SymbolAttrib	set/get the default symbolAttrib attribute

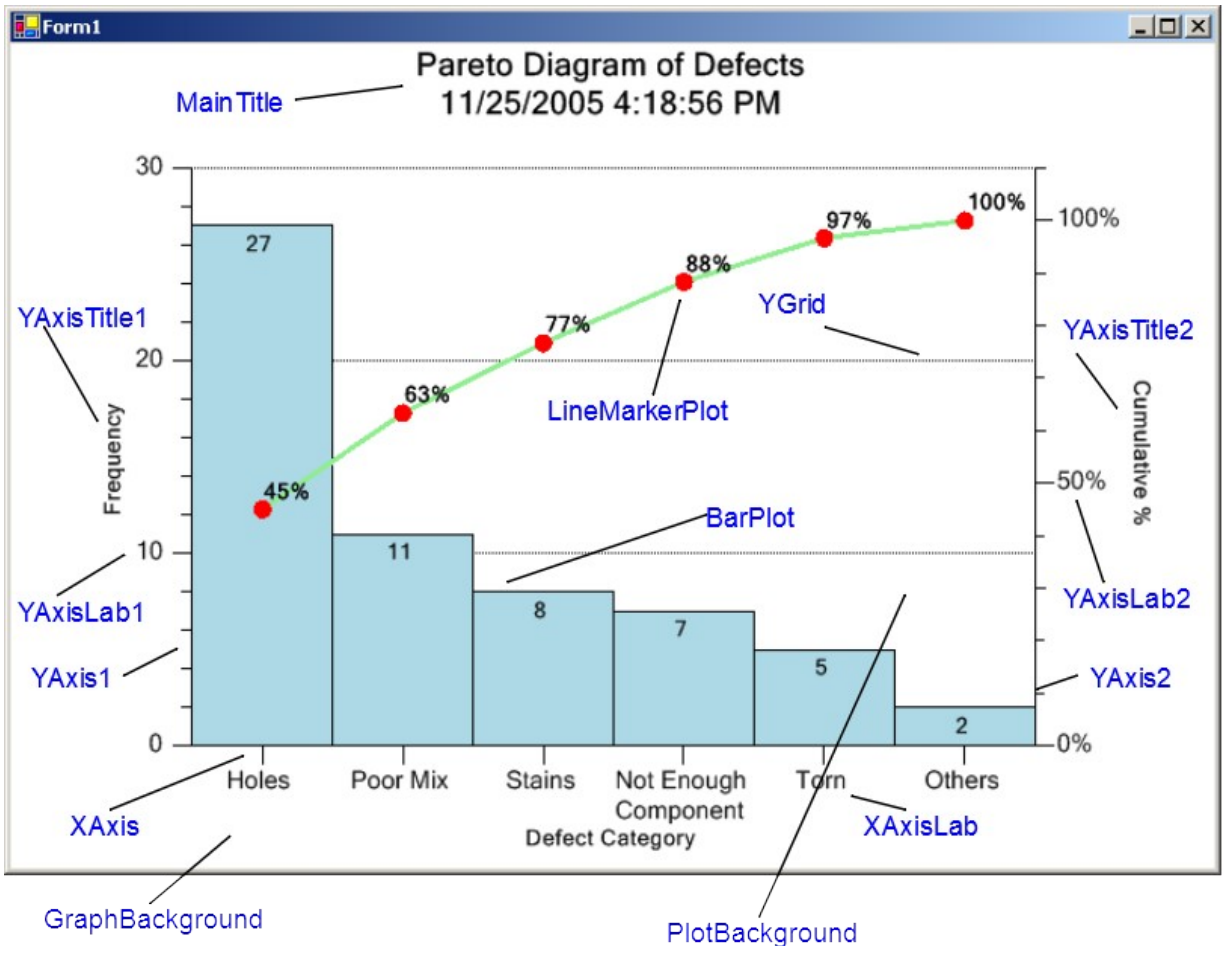
	object for the symbols of the chart. Set attributes before buildChart.
XAxis	set/get the x-axis of the chart object.
XAxisLab	set/get the x-axis string labels object of the chart.
XAxisTitle	set/get the x-axis title object of the of the chart.
YAxis1	set/get the y-axis object of the frequency histogram part of the chart.
YAxis2	set/get the y-axis object of the cumulative frequency part of the chart.
YAxisLab1	set/get the y-axis labels object of the frequency histogram part of the chart.
YAxisLab2	set/get the y-axis numeric labels object of the cumulative frequency part of the chart.
YAxisTitle1	set/get the y-axis title object of the frequency histogram part of the chart.
YAxisTitle2	set/get the y-axis title object of the cumulative frequency part of the chart.
YGrid	set/get the y-axis grid object of the chart.

Public Instance Functions

addCategoryItem	Add an item to the categoryValues and categoryStrings arrays.
buildChart	Builds the Pareto chart using the base objects ChartView.
initParetoChart	Initializes the category values, and the category strings for the Pareto chart.
initParetoChartsDatasets	Builds the histogram dataset, histogramDataset, using the values in frequencyValues and frequencyLimits.

The ParetoChart properties are documented in the QCSPCChartJSTSClassesIndex.html documentation file, located in the docs/docs/ subdirectory.

Changing Default Characteristics of the Chart



Once the graph is initialized (using the **initParetoChart** , or one of the **ParetoChart** constructors**)**, you can modify the default characteristics of each graph using these properties. For example, you can change the color of y-axis, and the y-axis labels using the **LineColor** property of those objects.

[JavaScript]

```
export async function BuildParetoChart(canvasid) {

        var htmlcanvas= document.getElementById(canvasid);
        var charttitle = "Pareto Example";

        var paretochart  =  QCSPCChartTS.ParetoChart.newParetoChart(htmlcanvas);
        if (!paretochart) return;

        paretochart.setPreferredSize(800, 600);
        paretochart.getMainTitle().setTextString("Pareto Diagram of Defects");
    // Build chart
    // add Pareto chart categories, values and strings
    paretochart.addCategoryItem(5, "Torn");
    paretochart.addCategoryItem(7, "Not Enough/nComponent");
    paretochart.addCategoryItem(2, "Others");
    paretochart.addCategoryItem(11, "Poor Mix");
    paretochart.addCategoryItem(27, "Holes");
    paretochart.addCategoryItem(8, "Stains");
```

```
paretochart.getLineMarkerPlot().getLineAttributes().setPrimaryColor( QCSPCChartTS.
ChartColor.BLUE);

paretochart.getLineMarkerPlot().getSymbolAttributes().setPrimaryColor( QCSPCChartT
S.ChartColor.BLACK);
    paretochart.getYAxis1().setLineColor( QCSPCChartTS.ChartColor.GREEN);
    paretochart.getYAxis1().setLineWidth( 3);
    paretochart.getYAxis2().setLineColor(QCSPCChartTS.ChartColor.BLUE);
    paretochart.getYAxis2().setLineWidth( 3);
    paretochart.getYAxisLab1().setLineColor(QCSPCChartTS.ChartColor.DARKMAGENTA);

        paretochart.getLineMarkerPlot().setShowDatapointValue(true);
        paretochart.buildChart();
        paretochart.updateDraw();

    }
```

[TypeScript]

```
export async function BuildParetoChart(canvasid) {

        var htmlcanvas= document.getElementById(canvasid);
        var charttitle = "Pareto Example";


        var paretochart  =  QCSPCChartTS.ParetoChart.newParetoChart(htmlcanvas);
        if (!paretochart) return;


        paretochart.setPreferredSize(800, 600);


        paretochart.getMainTitle().setTextString("Pareto Diagram of Defects");
    // Build chart
    // add Pareto chart categories, values and strings
    paretochart.addCategoryItem(5, "Torn");
    paretochart.addCategoryItem(7, "Not Enough/nComponent");
    paretochart.addCategoryItem(2, "Others");
    paretochart.addCategoryItem(11, "Poor Mix");
    paretochart.addCategoryItem(27, "Holes");
    paretochart.addCategoryItem(8, "Stains");


paretochart.getLineMarkerPlot().getLineAttributes().setPrimaryColor( QCSPCChartTS.
ChartColor.BLUE);

paretochart.getLineMarkerPlot().getSymbolAttributes().setPrimaryColor( QCSPCChartT
S.ChartColor.BLACK);
    paretochart.getYAxis1().setLineColor( QCSPCChartTS.ChartColor.GREEN);
    paretochart.getYAxis1().setLineWidth( 3);
    paretochart.getYAxis2().setLineColor(QCSPCChartTS.ChartColor.BLUE);
    paretochart.getYAxis2().setLineWidth( 3);
    paretochart.getYAxisLab1().setLineColor(QCSPCChartTS.ChartColor.DARKMAGENTA);

        paretochart.getLineMarkerPlot().setShowDatapointValue(true);
        paretochart.buildChart();
        paretochart.updateDraw();

    }
```

Chapter 10 - Regionalization for non-USA English Markets

SPCChartStrings

You will find all of the predefined strings in the software have been moved to the static class SPCChartStrings, which can be modified at run-time, or, if you have the QCSPCChartTS source code, at compile time. You can create multiple sets of strings, one for each unique region you sell to, and initialize the software to that set at run-time.

Apart from the strings, there are a couple other areas which benefit from regionalization. First is the use of the "," (comma) in some locales as the decimal separator, in place of the "." (period). Our software uses the standard numeric conversion routines found in JavaScript, for converting numeric values to strings, and these automatically take into account the proper format for the region recognized by the computer. So, you shouldn't have to do anything there.

Also, date/time values are subject to regional differences; specifically the order of the month-day-year in short form strings of the form 10/1/2011 ("M/dd/yyyy" US English format), compared to 1/10/2011 (("dd/M/yyyy" European format). There a couple of date/time formats in the list below, located at the enumerated values **defaultDateFormat** and **defaultTimeStampFormat.** You should change those to whatever time format you want. The format specification is that used by the .Net DateTime.ToString method.

The SPCChartStrings class uses an enumeration of internal strings which looks like this in the library.

```
export enum SPCStringEnum {

start = 0,
chartFont,
highAlarmStatus,
lowAlarmStatus,
shortStringNo,
shortStringYes,
dataLogUserString,
sPCControlChartDataTitle,
zeroEqualsZero,
timeValueRowHeader,
alarmStatusValueRowHeader,
```

numberSamplesValueRowHeader,
titleHeader,
.
.
.
"end"
];

The **SPChartStrings** class looks like:

```
export class SPCChartStrings {

static usEnglishStrings: string[] = [
"start",
"Arial",
"H",
"L",
"N",
"Y",
"",
"SPC Control Chart",
"zero",
"Time",
"Alarm",
"N° Inspected",
"Title: ",
.
.
.
"end"
];

 static currentDefaultStrings: string[] = SPCChartStrings.usEnglishStrings;

 public static getString(value: number): string ;
 public static setString(value: number, s: string): string ;
        .
        .
        .

}
```

The **SPCStringEnum** enumeration contains an item for every default string used in the software. Paired with that is the currentDefaultStrings array, which has one string element for every enumerated value in the **SPCStringEnum** enumeration. The software pulls the strings it uses from the currentDefaultStrings array. It is initialize by default with the usEnglishStrings array; the string values of which are listed in the tables below.

SPCStringEnum	Default Strings	Description
start=0	"start"	Marks the start of the enumeration
chartFont	"Microsoft Sans Serif"	default font string

highAlarmStatus	"H"	alarm status line - High short string
lowAlarmStatus	"L"	alarm status line - Low short string
shortStringNo	"N"	No short string
shortStringYes	"Y"	Yes short string
dataLogUserString	""	default data log user string
sPCControlChartDataTitle	"Variable Control Chart (X-Bar & R)"	Default chart title
zeroEqualsZero	"zero"	table zero string
timeValueRowHeader	"TIME"	TIME row header
alarmStatusValueRowHeader	"ALARM"	ALARM row header
numberSamplesValueRowHeader	"NO. INSP."	NO. INSP. row header
titleHeader	"Title: "	Title field caption
partNumberHeader	"Part No.: "	Part number field caption
chartNumberHeader	"Chart No.: "	Chart number field caption
partNameHeader	"Part Name: "	Part name field caption
operationHeader	"Operation:"	Operation field caption
operatorHeader	"Operator:"	Operator field caption
machineHeader	"Machine: "	Machine field caption
dateHeader	"Date: "	Date field caption
specificationLimitsHeader	"Spec. Limits: "	Spec limits field caption
gageHeader	"Gage: "	Chart number field caption
unitOfMeasureHeader	"Units: "	Chart number field caption
zeroEqualsHeader	"Zero Equals: "	Chart number field caption

defaultMean	"MEAN"	MEAN Calculated value row header
defaultMedian	"MEDIAN"	MEDIAN Calculated value row header
defaultRange	"RANGE"	RANGE Calculated value row header
defaultVariance	"VARIANCE"	VARIANCE Calculated value row header
defaultSigma	"SIGMA"	SIGMA Calculated value row header
defaultSum	"SUM"	SUM Calculated value row header
defaultSampleValue	"SAMPLE VALUE"	SAMPLE VALUE alculated value row header
defaultAbsRange	"ABS(RANGE)"	ABS(RANGE) Calculated value row header
defaultMovingAverage	"“MA”	
defaultCusumCPlus	""C+"	
defaultCusumCMinus	""C-"	
defaultEWMA	""EWMA"	
defaultPercentDefective	" "% DEF."	
defaultFractionDefective	""FRACT. DEF."	
defaultNumberDefective	""NO. DEF."	
defaultNumberDefects	""NO. DEF."	
defaultNumberDefectsPerUnit	""NO. DEF./UNIT"	
defaultNumberDefectsPerMillion	""DEF./MILLION"	
defaultPBar	""PBAR"	
defaultAttributeLCL	""LCLP"	
defaultAttributeUCL	""UCLP"	
defaultAbsMovingRange	"MR"	Moving Range Calculated value row header

defaultAbsMovingSigma	"MS"	Moving Sigam Calculated value row header
defaultX	"X"	Default string used to label centerline value of I-R chart.
defaultXBar	"XBAR"	Default string used to label centerline value for XBar chart
defaultRBar	"RBAR"	Default string used to label centerline value for Range chart
defaultTarget	"Target"	Default string used for target
defaultLowControlLimit	"LCL"	Default string used to label low control limit line
defaultLowAlarmMessage	"Low Alarm"	Default string used for low alarm limit message
defaultUpperControlLimit	"UCL"	Default string used to label high control limit line
defaultHighAlarmMessage	"High Alarm"	Default string used for high alarm limit message
defaultSampleRowHeaderPrefix	"Sample #"	Row header for Sample # rows
defaultDefectRowHeaderPrefix	"Defect #"	Row header for Defect # rows
batchColumnHead	"Batch #"	Default string used as the batch number column head in the log file.
timeStampColumn	"Time Stamp"	Default string used as the time stamp column head in the log file.
sampleValueColumn	"Sample #"	Default string used as the sample value column head in the log file.
notesColumn	"Notes"	Default string used as the notes value column head in the log file.
defaultDateFormat	"M/dd/yyyy"	Default date format used by the software.
defaultTimeStampFormat	"M/dd/yyyy HH:mm:ss"	Default full date/time format used by the software.
defaultDataLogFilenameRoot	"SPCDataLog"	Root string used for auto-naming of log data file.
dataLogFilename	"SPCDataLog"	Datalog default file name
frequencyHistogramXAxisTitle	"Measurements"	Frequency Histogram default x-axis title.

frequencyHistogramYAxisTitle	"Frequency"	Frequency Histogram default y-axis title.
frequencyHistogramMainTitle	"Frequency Histogram"	Frequency Histogram default main title.
paretoChartXAxisTitle	"Defect Category"	Pareto chart x-axis title
paretoChartYAxis1Title	"Frequency"	Pareto chart left y-axis title
paretoChartYAxis2Title	"Cumulative %"	Pareto chart right y-axis title
paretoChartMainTitle	"Pareto Diagram"	Pareto chart main title
probabilityChartXAxisTitle	"Frequency Bin"	Probability chart x-axis title
probabilityChartYAxisTitle	"% Population Under"	Probability chart y-axis title
probabilityChartMainTitle	"Normal Probability Plot"	Probability chart main title
basic	"Basic"	Basic rules string
weco	"WECO"	WECO rules string
wecowsupp	"WECO+SUPPLEMENTAL"	WECO and Supplemental Rules string
nelson	"Nelson"	Nelson rules string
aiag	"AIAG"	AIAG rules string
juran	"Juran"	Juran rules string
hughes	"Hughes"	Hughes rules string
gitlow	"Gitlow"	Gitlow rules string
duncan	"Duncan"	Duncan rules string
westgard	"Westgard"	Westgard rules string
primarychart	"Primary chart"	Used in alarm messages to specify the Primary Chart variable chart is in alarm
secondarychart	"Secondary chart"	Used in alarm messages to specify the Secondary Chart variable chart is in alarm

greaterthan	"greater than"	Used in alarm messages to specify that a greater than alarm limit has been violated
lessthan	"less than"	Used in alarm messages to specify that a less than alarm limit has been violated
above	"above"	Used in alarm messages to specify that values above a limit
below	"below"	Used in alarm messages to specify that values below a limit
increasing	"increasing"	Used in alarm messages to specify that values are increasing
decreasing	"decreasing"	Used in alarm messages to specify that values are decreasing
trending	"trending"	Used in alarm messages to specify that values are trending
within	"within"	Used in alarm messages to specify that values are within certain limits
outside	"outside"	Used in alarm messages to specify that values are outside certain limits
alternating	"alternating"	Used in alarm messages to specify that values are alternating about a limit value
centerline	"center line"	Used in alarm messages to specify the center line of the chart
r2s	"R2s"	Used in alarm messages to specify the R2s test
sigmaShort	"S"	alarm status line - sigma short string
beyondAlarmStatus	"B"	alarm status line - beyond short string
trendingAlarmStatus	"T"	alarm status line - trending short string
stratificationAlarmStatus	"S"	alarm status line - stratification short string
oscillationAlarmStatus	"O"	alarm status line - oscillation short string
r2sAlarmStatus	"R"	alarm status line - R2s short string
rule	"Rule"	used in alarm messages for word "Rule"

violation	"violation"	used in alarm messages for word "violation"
sigma	"sigma"	used in alarm messages for word "sigma"
target	"Target"	used in alarm messages for word "Target"
ucl	"UCL"	used in alarm messages for to designate Upper Control Limit
lcl	"LCL"	used in alarm messages for to designate Lower Control Limit
defaultCp	"Cp"	used to label Cp row of table
defaultCpl	"Cpl"	used to label Cpl row of table
defaultCpu	"Cpu"	used to label Cpu row of table
defaultCpk	"Cpk"	used to label Cpk row of table
defaultCpm	"Cpm"	used to label Cpm row of table
defaultPp	"PPp"	used to label Pp row of table
defaultPl	"Ppl"	used to label Pl row of table
defaultPu	"Ppu"	used to label Pu row of table
defaultPpk	"Ppk"	used to label Ppk row of table
end	"end"	Marks the end of the enumeration

The SPCStringEnum enumeration contains an item for every default string used in the software. Paired with that is the currentDefaultStrings array, which has one string element for every enumerated value in the SPCStringEnum enumeration. The software pulls the strings it uses from the currentDefaultStrings array. It is initialized by default with the usEnglishStrings array.

The enumerated values can be used to access and modify any specific string.

[JavaScript]

```
var  oldstring = QCSPCChartTS.SPCChartStrings.setString(
     QCSPCChartTS.SPCChartStrings.SPCStringEnum.defaultMean,"AVERAGE");

oldstring = QCSPCChartTS.SPCChartStrings.setString(
     QCSPCChartTS.SPCStringEnum.defaultDateFormat,"dd/M/yyyy");
```

[TypeScript]

```
let  oldstring: string = QCSPCChartTS.SPCChartStrings.setString(
     QCSPCChartTS.SPCChartStrings.SPCStringEnum.defaultMean,"AVERAGE");

oldstring = QCSPCChartTS.SPCChartStrings.setString(
     QCSPCChartTS.SPCStringEnum.defaultDateFormat,"dd/M/yyyy");
```

Where the static **setString** method returns the previous value for the string.

The **SPCChartStrings** module is used to define the default for the various **QCSPCChart** classes, when those classes are initialized. Trying to set the **SPCChartStrings** strings using **setString** after any of the SPC charts have been instantiated will not have the desired effect. Since the SPCChartStrings class is static, you can call it anytime. It does not need instantiation. Just make sure that you call it before any of the SPC Chart classes are instantiated, as in the example below.

[JavaScript]

```
 public async  BuildXBarRChart(canvasid: string) {

        var htmlcanvas = <QCSPCChartTS.Canvas>document.getElementById(canvasid);
        var spccharttype= QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
        var subgroupsize = 5;
        var numberpointsinview= 12;
        var charttitle = "XBar-R Example";


       var oldstring =
QCSPCChartTS.SPCChartStrings.setString(  QCSPCChartTS.SPCStringEnum.defaultMean,"M
ean");
       oldstring =
QCSPCChartTS.SPCChartStrings.setString(QCSPCChartTS.SPCStringEnum.defaultDateForma
t, "M/dd/yyyy");
```

```
        var xbarrchart  =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
Subgroupsize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
        if (!xbarrchart) return;

        .
.
.


    }
```

[TypeScript]

```
 public async  BuildXBarRChart(canvasid: string) {

        let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
        let spccharttype: number =
QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
        let subgroupsize: number = 5;
        let numberpointsinview: number = 12;
        let charttitle: string = "XBar-R Example";


       let oldstring: string =
QCSPCChartTS.SPCChartStrings.setString(  QCSPCChartTS.SPCStringEnum.defaultMean,"M
ean");
       oldstring =
QCSPCChartTS.SPCChartStrings.setString(QCSPCChartTS.SPCStringEnum.defaultDateForma
t, "M/dd/yyyy");


        let xbarrchart: QCSPCChartTS.SPCChartBase  =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
Subgroupsize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
        if (!xbarrchart) return;

        .
.
.


    }
```

You can change every string used in the software, line by line, using the technique above.

Change the default language

You can initialize the currentDefaultStrings array for languages other than English. We have include preliminary translations of all of the strings to the following additional languages: Spanish, Portuguese, French, German and Italian. You can change the currentDefaultStrings using a call to the static **initSPCChartStrings** method.

public static initSPCChartStrings(i: number): boolean

method, passing in one of the SPCChartStrings static constants: US_ENGLISH, ES_SPANISH, PT_PORTUGUESE, FR_FRENCH, DE_GERMAN or IT_ITALIAN.

The method always returns true. For example:

[JavaScript]

```
var  done =
QCSPCChartTS.SPCChartStrings.initSPCChartStrings(QCSPCChartTS.SPCChartStrings.ES_S
PANISH);
```

[TypeScript]

```
let  done: boolean =
QCSPCChartTS.SPCChartStrings.initSPCChartStrings(QCSPCChartTS.SPCChartStrings.ES_S
PANISH);
```

Chapter 11 - Using SPC Control Chart Tools for JavaScript/TypeScript to 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 QCSPCChartTS library file (qcspcchartts.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 { XBarRChart} from './XBarRChart.js';
```

The "./"' in the folder specification above represents the current folder that contains the HTML file you are working with.

Even though the TypescriptExamples 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 2019) 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 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/JavascriptExamples/XBarRChart/XBarRChart.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/JavascriptExamples/XBarRChart/XBarRChart.html

The main folder containing both the JavaScript and TypeScript examples is Quinn-Curtis/JSTS. In that folder is a batch file which will start the npm "http-server" local server,

setting the directory it is located in as the root directory of the test server. All the batch file contains is:

http-server ./

where http-server starts the npm server, and "./" specifies that the current directory is to be the root folder of the server. In order to start the npm "http-server" that way it is assumed that you have installed the npm "http-server" command line server globally according to the instructions here: https://www.npmjs.com/package/http-server You will need a relatively current version of npm to properly install the http-server package.

Linux (Ubuntu) users: If you plan to use Linux (Ubuntu), see the section at the end of this chapter for issues unique to that development environment.

Visual Studio Code

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

```
npm install -g typescript
```

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

npm -v

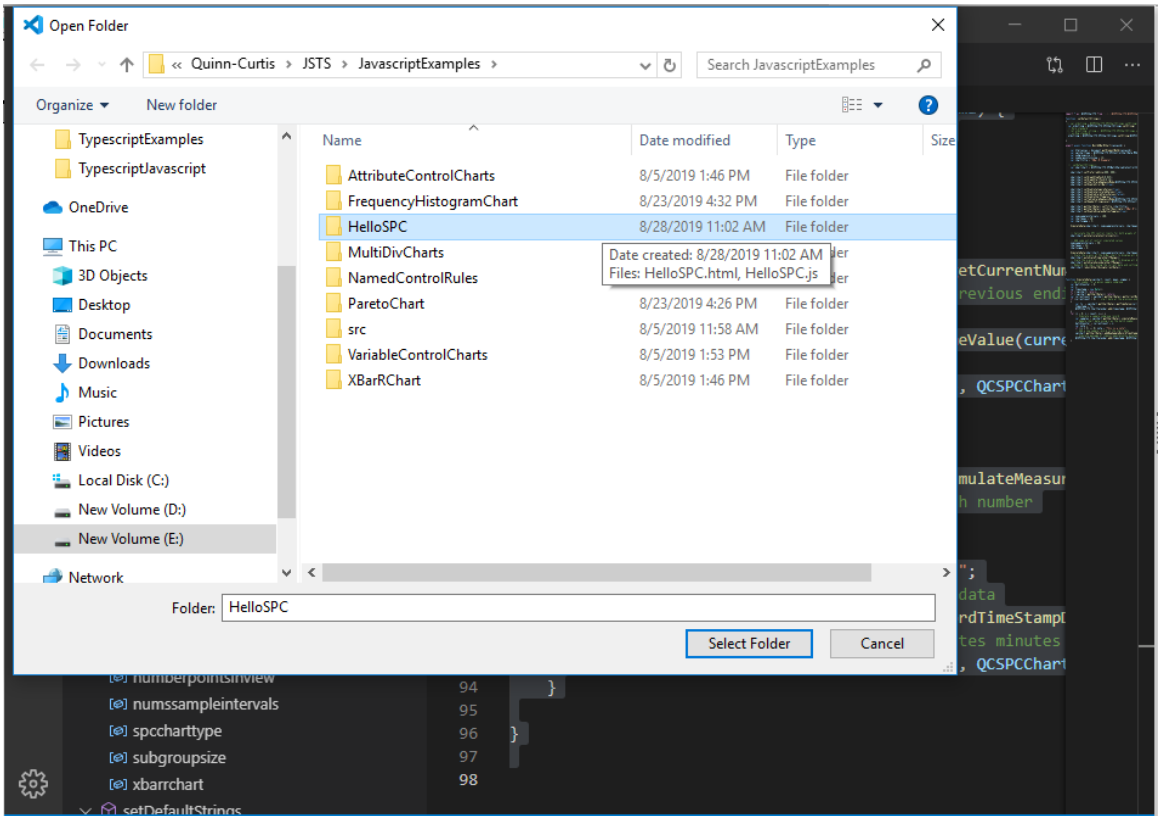
from the command prompt. If you need to install npm, or upgrade it, you will find resources on the web which tell you how to do that.

Using JavaScript Only

While we recommend that you use a TypeScript intermediary to interact with the QCSPCChartTS library, you can also call it directly from JavaScript. While QCSPCChartTS 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.

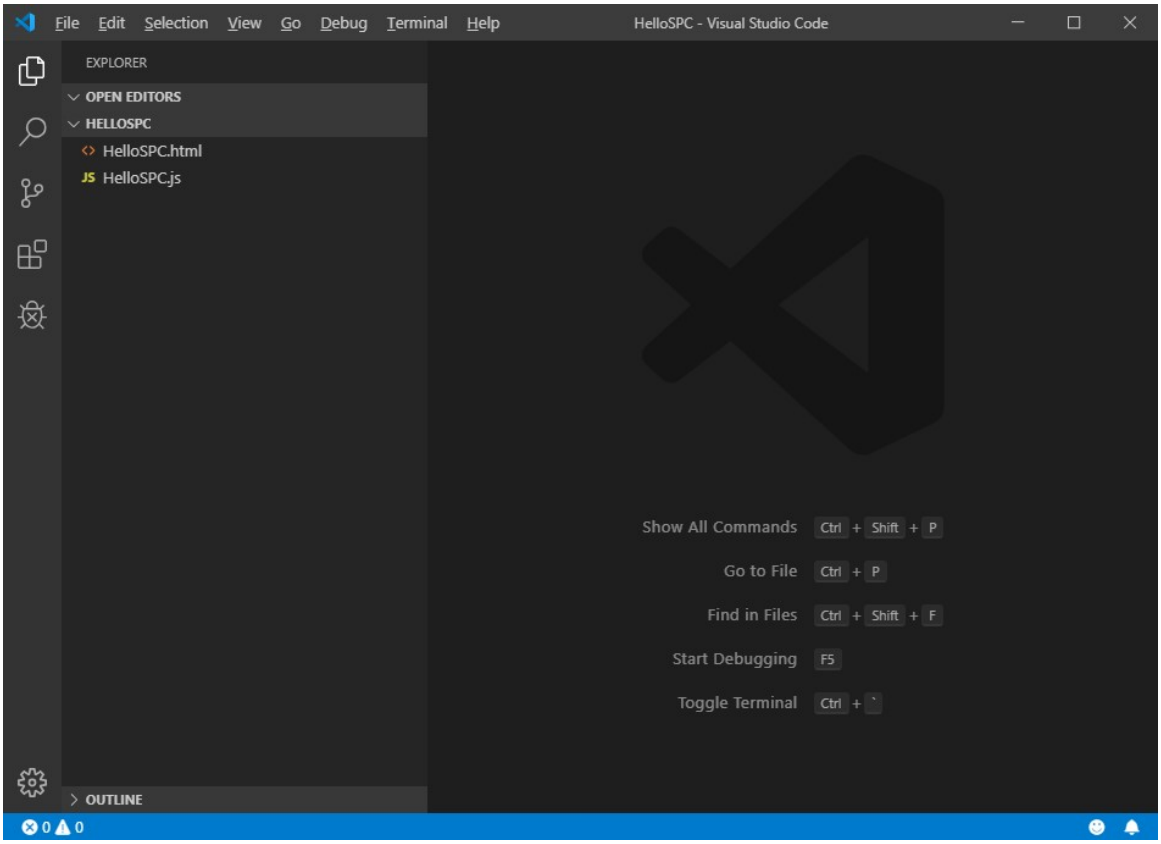
Start off by creating a folder unique to your intended code, under our JSTS/JavacriptExamples/ folder. We will use the folder name HelloSPC. In its simplest

form,that folder will contain just two files: your test HTML page (HelloSPC.html) and the JavaScript source page (HelloSPC.js). In this example, the HelloSPC.html page can just be a bare-bones HTML page, the same as our JavascriptExamples/XBarRChart.html. The only difference is in the embedded filename used to access the HelloSPC.js JavaScript file.



From your JavaScript Development Environment (Visual Studio Code in our case), open the HelloSPC folder using the IDE File | Open Folder option, and select the HelloSPC folder you created.

Next, create two new files in the HelloSPC directory: HelloSPC.html and HelloSPC.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 HelloSPC.html and HelloSPC.js . So you should end up with a folder named HelloSPC containing two blank files, HelloSPC.html and HelloSPC.js.



Copy the code below into the HelloSPC.html file.

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

    <div id="buttondiv">
        <button type="button" id="xbarr_menuitem">XBar-R</button>
    </div>

    <div id="canvasdiv">
        <canvas id="spcChartCanvas1" width="800" height="600"></canvas>
     </div>
        <script type="module">

          import { BuildXBarRChart} from './HelloSPC.js';


          function xbarrchart() {
              BuildXBarRChart("spcChartCanvas1");
          }


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

        </script>
```

```
        <br>
        <br>

</body>
</html>
```

Note that the import statement looks to the current directory for the HelloSPC.js file.

```
import { BuildXBarRChart} from './HelloSPC.js';
```

When the button (id = xbarr_menuitem) is pressed, it triggers a call to the BuildXBarRChart function imported from the HelloSPC.js file. Note that the event handler (xbarrchart) for the button:

```
          document.getElementById("xbarr_menuitem").onclick = xbarrchart;
```

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 xbarrchart 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 xbarrchart function has been defined, it will work.

Save the HelloSPC.html file and go to the HelloSPC.js file. You can copy the code below into that file. It is exactly the same code as found in the JavascriptExamples/XBarRChart/ XBarRChart.js file.

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';


export async function BuildXBarRChart(canvasid) {

    var htmlcanvas = document.getElementById(canvasid);
    var spccharttype = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    var subgroupsize = 5;
    var numberpointsinview = 12;
    var charttitle = "XBar-R Example";

    var xbarrchart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);

    xbarrchart.setPreferredSize(800, 600);

    xbarrchart.setGraphStopPosX(0.825);
    xbarrchart.setGraphStartPosX(0.18);

xbarrchart.setChartAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_SYM
BOL);
    xbarrchart.setEnableScrollBar(true);

    xbarrchart.setEnableCategoryValues(true);
    xbarrchart.setEnableCalculatedValues(true);
```

```
    xbarrchart.setEnableAlarmStatusValues(false);
    xbarrchart.setEnableChartToggles(true);

xbarrchart.setTableAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_BAR
);

xbarrchart.setHeaderStringsLevel( QCSPCChartTS.SPCControlChartData.HEADER_STRINGS_
LEVEL1);

    xbarrchart.getChartData().setTitle (charttitle);
    xbarrchart.getChartData().setChartDescriptor("XBar-R");
    xbarrchart.setEnableDisplayOptionToggles(true);

    var numssampleintervals = 100;
    var chartmean = 30;
    var chartsigma = 5;

    SimulateData(xbarrchart, numssampleintervals, chartmean, chartsigma);


    // Calculate the SPC control limits for both graphs of the current SPC chart
(X-Bar R)
    xbarrchart.autoCalculateControlLimits();

    // Add some out of control simulated values
    numssampleintervals = 50;
    chartmean = 32;
    chartsigma = 6;

    SimulateData(xbarrchart, numssampleintervals, chartmean, chartsigma);
    // Scale the y-axis of the X-Bar chart to display all data and control limits
    xbarrchart.autoScalePrimaryChartYRange();
    // Scale the y-axis of the Range chart to display all data and control limits
    xbarrchart.autoScaleSecondaryChartYRange();
    // Rebuild the chart using the current data and settings
    xbarrchart.rebuildChartUsingCurrentData();

}


function SimulateData(spcchart, count, mean, sigma) {
    // batch number for a given sample subgroup
    var batchCounter = 0;
    var i;
    var timestamp = new Date();
    if (!spcchart) return;
    if (!spcchart.getChartData()) return;
    var currentcount = spcchart.getChartData().getCurrentNumberRecords();
    if (currentcount > 0) // start date at the previous ending date plut time
increment
    {
        var ts  = spcchart.getChartData().getTimeValue(currentcount-1);
        timestamp = ts;
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
    }
    for (i = 0; i < count; i++) {
        // Simulate a sample subgroup record
        var samples =
spcchart.getChartData().simulateMeasurementRecordMeanRange(mean, sigma);
        // Update chart data using i as the batch number
        batchCounter = currentcount + i;
        var note = "";
        if ((i % 5) == 0) note = "This is a note";
        // Add a new sample record to the chart data

spcchart.getChartData().addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter
, timestamp, samples, note);
        // Simulate passage of timeincrementminutes minutes
        QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
```

```
    }

}
```

Note that the QCSPCChartTS library is being imported from its location in the file system, relative to the HelloSPC.js file.

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
```

The “import * as QCSPCChartTS” in the import statement assigns all classes, functions, properties and constants in the software to the namespace QCSPCChartTS, so you must preface any reference to the formal name of any class, static method within a class, or static constants, found within the QCSPCChart library, by explicitly referencing that namespace.

```
    var spccharttype = QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
    var subgroupsize = 5;
    var numberpointsinview = 12;
    var charttitle = "XBar-R Example";

    var xbarrchart =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
```

The HelloSPC.js file consists of just two function. The first BuildXBarRChart, builds the chart, and it in turn call the data simulation function SimulateData.

In broad terms, first you create an instance of the control chart using one of the static control chart constructors:

QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartTypeSubgroupSize

passing in the required parameters. 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.

In the example, get and set the chart properties using the xbarrchart instance of the chart returned by the static constructor.

```
    xbarrchart.setEnableCategoryValues(true);
    xbarrchart.setEnableCalculatedValues(true);
    xbarrchart.setEnableAlarmStatusValues(false);
    xbarrchart.setEnableChartToggles(true);
```

This will look very familiar if you have used QCSPCChart on any of the other programming languages we support (C#, VB, and Java, etc.). Since JavaScript and TypeScript do not support the property syntax of .Net, all properties values are assigned and retrieved using setters and getters, i.e. a prefix of set or get in front of the property name.

In many instances you need to get at instances of classes within the chart, ChartData (an instance of SPCControlChartData), ChartTable (an instance of SPCGeneralizedTableDisplay), PrimaryChart or SecondaryChart (instances of SPCControlPlotObjectData), and ControlLimitValues (an array of SPCControlLimitRecord) to name the most common. Internally, these properties can be null while the chart constructor is instantiating, but once that is done they are never null (or undefined). So it is safe to use code like seen below, where you retrieve the instance of the object using a getter call (getChartData()).

```
    xbarrchart.getChartData().setTitle (charttitle);
    xbarrchart.getChartData().setChartDescriptor("XBar-R");
```

This is not allowed in TypeScript if you have strict null checking enabled, which all of our TypeScript examples do. Strict error checking can be annoying in TypeScript because it flags so many things as errors, but its overall usefulness cannot be overstated. Because the value returned by getChartData() is marked as returning a union of SPCControlChartData and null. And the compiler won't let you reference a function in ChartData unless it absolutely sure that the value returned by getChartData() is not null. To work around this you use code like this:

[TypeScript Code]

```
let chartData: SPCControlChartData | null = xbarrchart.getChartData();
if (chartData)
{
    chartData.setTitle (charttitle);
    chartData.setChartDescriptor("XBar-R");
}
```

The equivalent JavaScript code would look like:

[JavaScript Code]

```
var chartData = xbarrchart.getChartData();
if (chartData)
{
    chartData.setTitle (charttitle);
    chartData.setChartDescriptor("XBar-R");
}
```

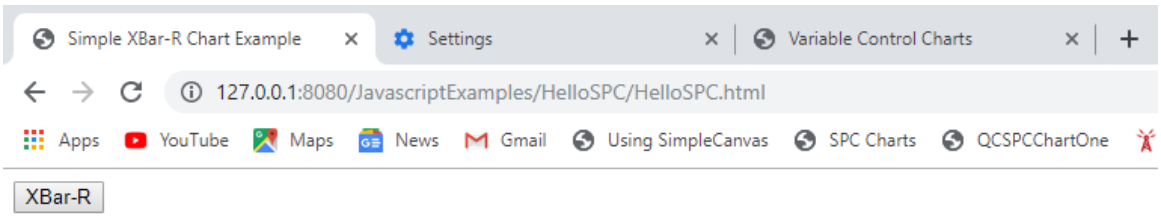
This applies to all of the class instances internal to the main chart class.

But since JavaScript does not check for nulls, you can do it either way.

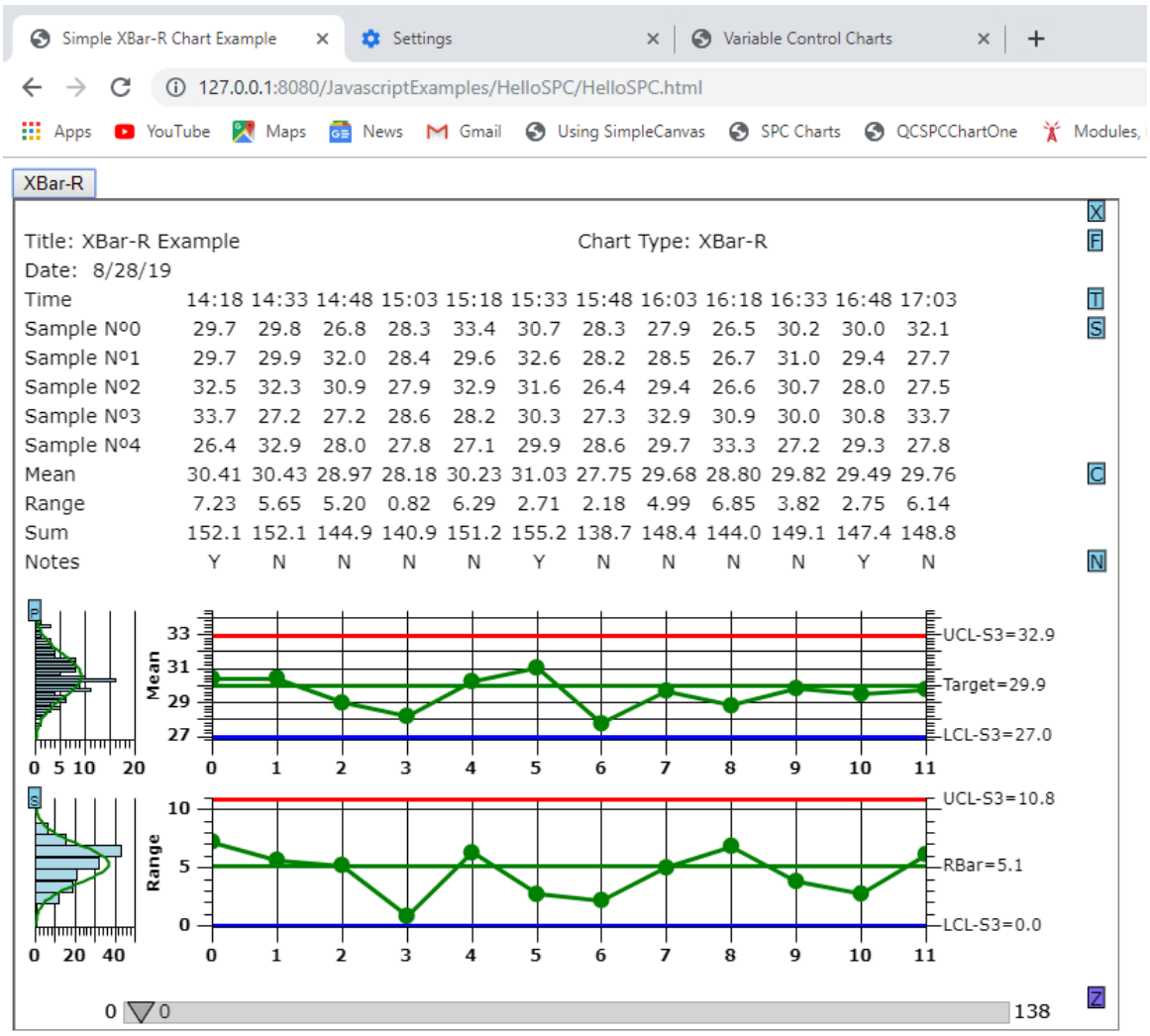
Assuming you have the two files setup correctly, and you have started some sort of test server which references the Quinn-Curtis/JSTS folder as its root directory, you should be able to run the test program by setting a browser to load the file at the URL:

http://127.0.0.1:8080/JavascriptExamples/HelloSPC/HelloSPC.html

The result will be the following display:



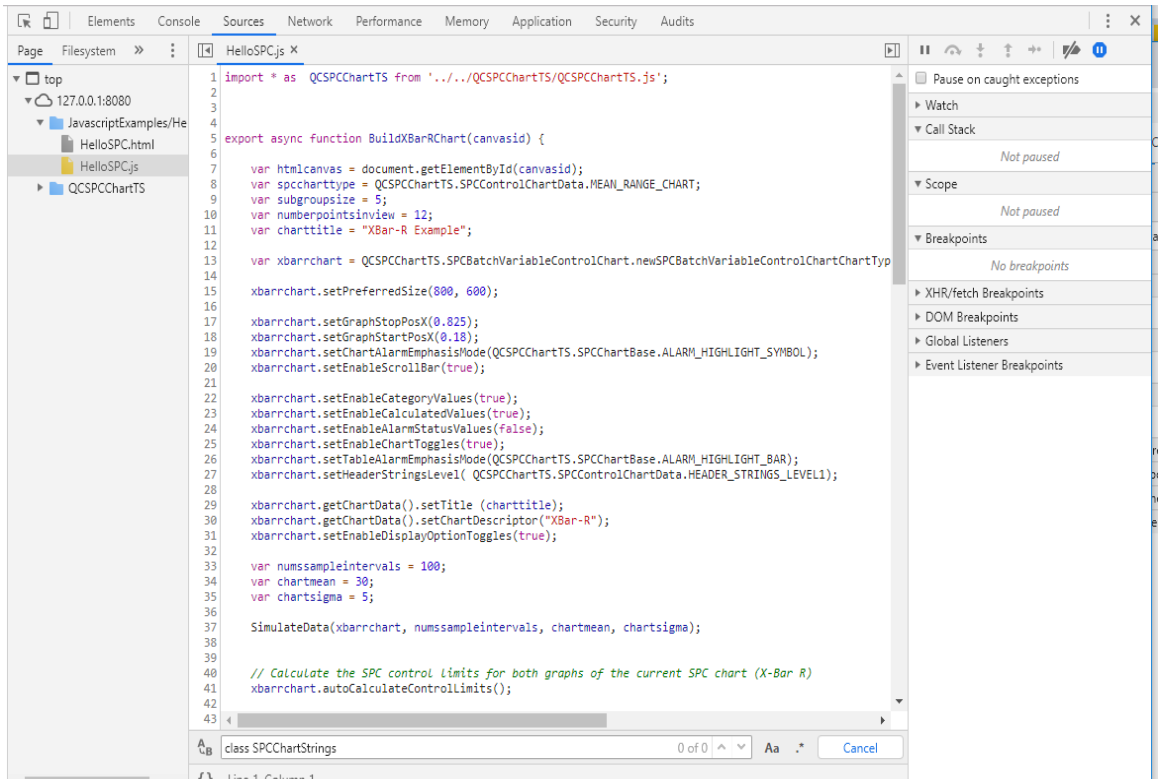
showing just the button. Click the button and the chart will appear.



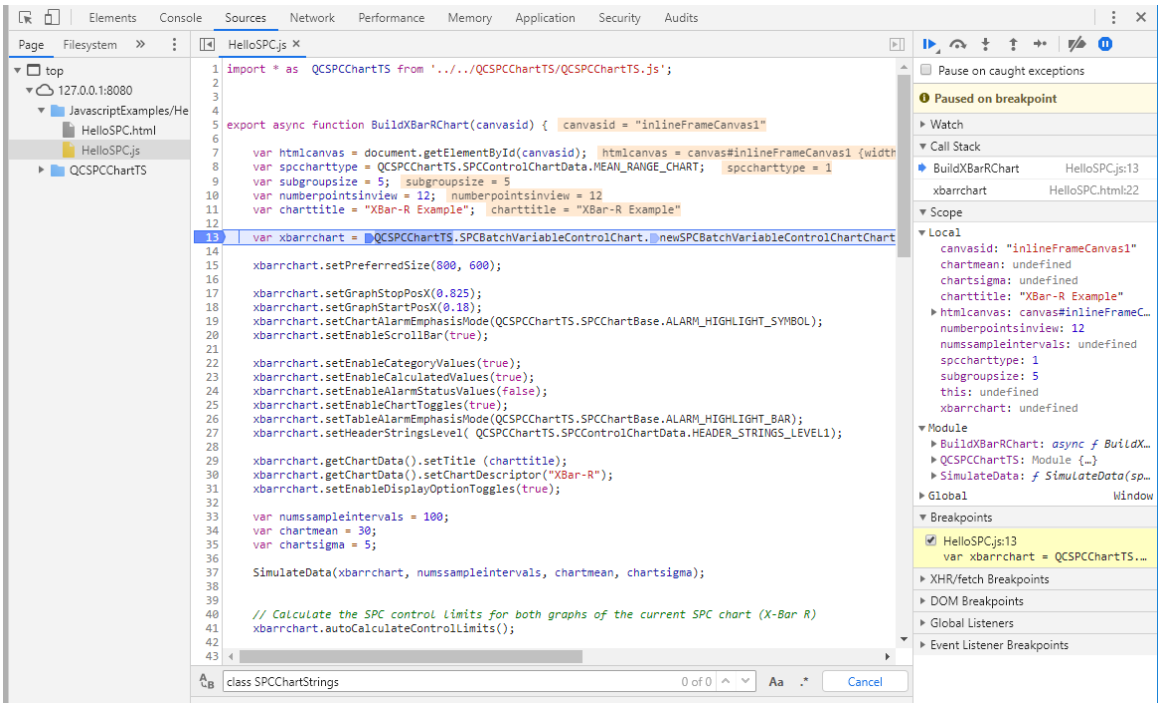
Debugging

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 HelloSPC.html file you should see the XbarRChart 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 HelloSPC.js file, or the HelloSPC.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 HelloSPC.js file, and then click the Xbar-R button in the left hand side of the browser, program execution will stop in the code window when the breakpoint is hit.



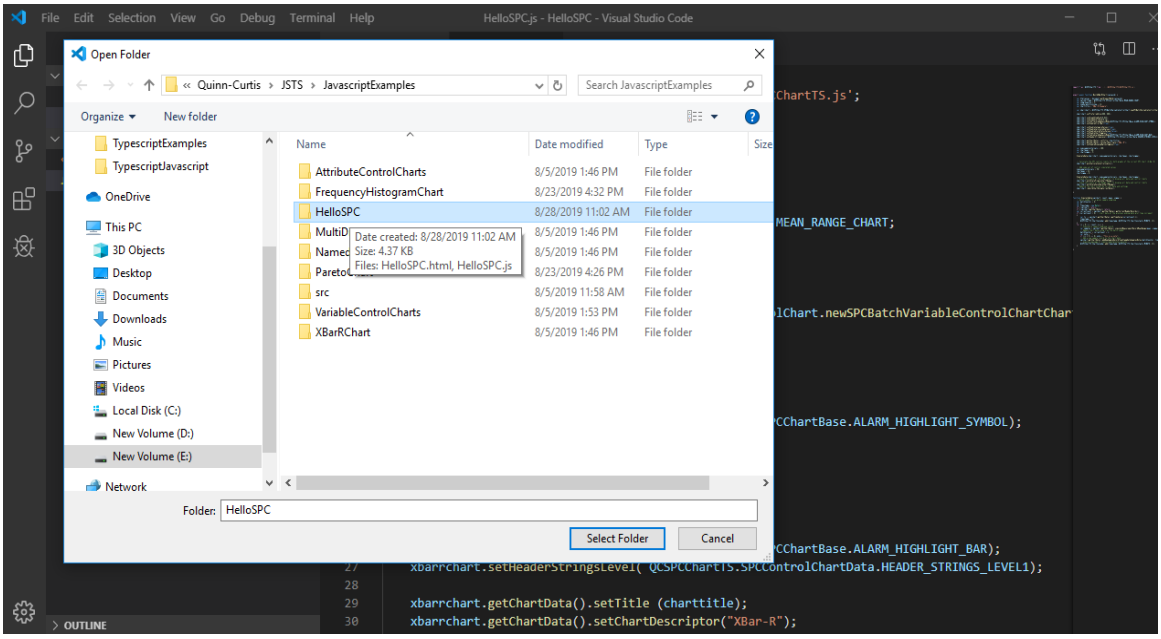
Using TypeScript

The QCSPCChartTS 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 qcspcchartts.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 qcspcchartts.js file is located in the Quinn-Curtis/JSTS/QCSPCChartTS folder. So your TypeScript source files which reference the QCSPCChartTS library must import it from that location. We specify the location of the qcspcchartts.js file relative to the the source folder of the file we will be working with, using:

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
```

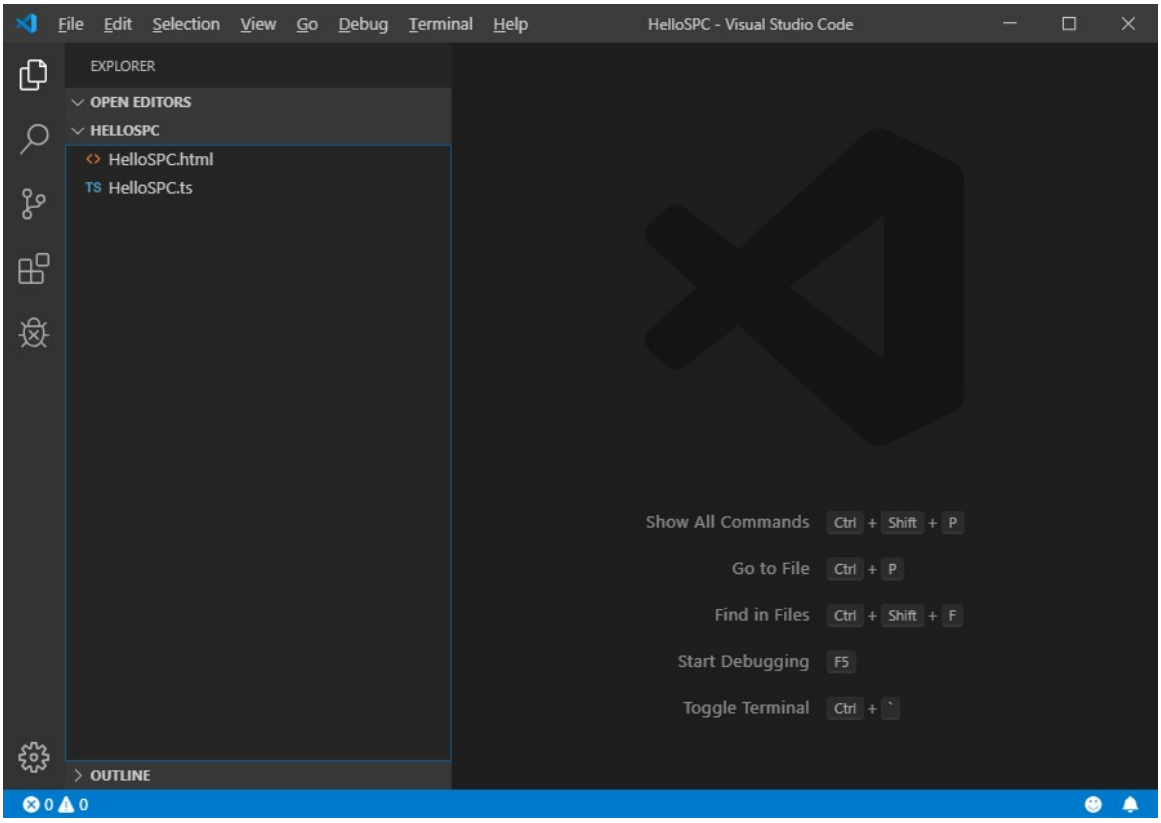
Start off by creating a folder unique to your intended code, under our JSTS/TypescriptExamples/ folder. We will use the folder name HelloSPC. In its simplest form, that folder will contain just two files: your test HTML page (HelloSPC.html) and the JavaScript source page (HelloSPC.ts). In this example, the HelloSPC.html page can just be an bare-bones HTML page, the same as our TypescriptExamples/XBarRChart.html. The only difference is in the embedded filename used to access the HelloSPC.js JavaScript file. Even though your HelloSPC.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 HelloSPC folder using the IDE File | Open Folder option, and select the HelloSPC folder you created.

Next, create two new files in the HelloSPC directory: HelloSPC.html and HelloSPC.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 HelloSPC.html and HelloSPC.ts . So you should end up with a folder named HelloSPC containing two blank file, HelloSPC.html and HelloSPC.ts.



Copy the code below into the HelloSPC.html file.

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

    <div id="buttondiv">
        <button type="button" id="xbarr_menuitem">XBar-R</button>
    </div>

    <div id="canvasdiv">
        <canvas id="spcChartCanvas1" width="800" height="600"></canvas>
     </div>
        <script type="module">

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

     var spcchart = new HelloSPC();


          function xbarrchart() {
            spcchart.BuildXBarRChart("spcChartCanvas1");
          }


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

```
        </script>

        <br>
        <br>

</body>
</html>
```

Note that the import statement looks to the current directory for the HelloSPC.js file.

```
import { BuildXBarRChart} from './HelloSPC.js';

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

     var spcchart = new HelloSPC();
```

The import statement loads the HelloSPC.js file and makes the HelloSPC class inside available as the class HelloSPC. When that is completed, an instance of the HelloSPC class found in that file, is instantiated using new HelloSPC(). Until you compile the HelloSPC project, the HelloSPC.js file will not exist. When the project is compiled, the HelloSPC.ts file is compiled into the HelloSPC.js file and placed in the same folder, HelloSPC, from which it is loaded when needed.

When the button (id = xbarr_menuitem) is pressed, it triggers a call to the BuildXBarRChart function imported from the HelloSPC.js file. Note that the event handler (xbarrchart) for the button:

```
          document.getElementById("xbarr_menuitem").onclick = xbarrchart;
```

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 xbarrchart 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 xbarrchart function has been defined, it will work.

Save the HelloSPC.html file and go to the HelloSPC.ts file. You can copy the code below into that file. It is exactly the same code as found in the TypescriptExamples/XBarRChart/XBarRChart.js file, except that the encomapassing class is HelloSPC, instead of XbarChart.

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';


export class HelloSPC {

    public constructor() {

    }


    public async  BuildXBarRChart(canvasid: string) {
```

```
        let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
        let spccharttype: number =
QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
        let subgroupsize: number = 5;
        let numberpointsinview: number = 12;
        let charttitle: string = "XBar-R Example";

        let xbarrchart: QCSPCChartTS.SPCChartBase  =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
SubgroupSize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
        if (!xbarrchart) return;

        xbarrchart.setPreferredSize(800, 600);

        xbarrchart.setGraphStopPosX(0.825);
        xbarrchart.setGraphStartPosX(0.18);

xbarrchart.setChartAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_SYM
BOL);
        xbarrchart.setEnableScrollBar(true);

        xbarrchart.setEnableCategoryValues(true);
        xbarrchart.setEnableCalculatedValues(true);
        xbarrchart.setEnableAlarmStatusValues(false);
        xbarrchart.setEnableChartToggles(true);

xbarrchart.setTableAlarmEmphasisMode(QCSPCChartTS.SPCChartBase.ALARM_HIGHLIGHT_BAR
);

xbarrchart.setHeaderStringsLevel(QCSPCChartTS.SPCControlChartData.HEADER_STRINGS_L
EVEL1);

        xbarrchart.setEnableDisplayOptionToggles(true);
        let chartdata: QCSPCChartTS.SPCControlChartData | null =
xbarrchart.getChartData();
        if (chartdata) {
            chartdata.setTitle(charttitle);
            chartdata.setChartDescriptor("XBar-R");
        }
        let numssampleintervals: number = 100;
        let chartmean: number = 30;
        let chartsigma: number = 5;

        this.SimulateData(xbarrchart, numssampleintervals, chartmean, chartsigma);


        // Calculate the SPC control limits for both graphs of the current SPC
chart (X-Bar R)
        xbarrchart.autoCalculateControlLimits();

    // Add in some out of control values
    numssampleintervals = 50;
    chartmean = 32;
    chartsigma = 8;

    this.SimulateData(xbarrchart, numssampleintervals, chartmean, chartsigma);

        // Scale the y-axis of the X-Bar chart to display all data and control
limits
        xbarrchart.autoScalePrimaryChartYRange();
        // Scale the y-axis of the Range chart to display all data and control
limits
        xbarrchart.autoScaleSecondaryChartYRange();
        // Rebuild the chart using the current data and settings
        xbarrchart.rebuildChartUsingCurrentData();

    }
```

```
    SimulateData(spcchart: QCSPCChartTS.SPCChartBase , count: number, mean:
number, sigma: number) {
        // batch number for a given sample subgroup
        let batchCounter = 0;
        let i = 0;
        let timestamp = new Date();
        if (!spcchart) return;
        let chartdata: QCSPCChartTS.SPCControlChartData | null =
spcchart.getChartData();
        if (!chartdata) return;
        let currentcount: number = chartdata.getCurrentNumberRecords();
        if (currentcount > 0) // start date at the previous ending date plut time
increment
        {
            let ts : Date = chartdata.getTimeValue(currentcount-1);
            timestamp = ts;
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }
        for (i = 0; i < count; i++) {
            // Simulate a sample subgroup record
            let samples: QCSPCChartTS.DoubleArray =
chartdata.simulateMeasurementRecordMeanRange(mean, sigma);
            // Update chart data using i as the batch number
            batchCounter = currentcount + i;
            let note: string = "";
            if ((i % 5) == 0) note = "This is a note";
            // Add a new sample record to the chart data
            chartdata.addNewSampleRecordBatchNumberDateSamplesNotes(batchCounter,
timestamp, samples, note);
            // Simulate passage of timeincrementminutes minutes
            QCSPCChartTS.ChartCalendar.add(timestamp,
QCSPCChartTS.ChartConstants.MINUTE, 15);
        }

    }

}
```

In the HelloSPC.ts file, the body of the program is setup as a TypeScript class, named HelloSPC. It has three member functions: the public default constructor, the public BuildXBarRChart member function which is called from the HelloSPC.html HTML file, and the local SimulateData member function. 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 HelloSPC, found under JavascriptExamples/HelloSPC.

Note that the QCSPCChartTS library is being imported from its location in the file system, relative to the HelloSPC.js file.

```
import * as  QCSPCChartTS from '../../QCSPCChartTS/qcspcchartts.js';
```

The “import * as QCSPCChartTS” in the import statement assigns all classes, static functions within a class, properties and constants in the software to the namespace QCSPCChartTS, so you must preface any reference to the formal name of any classes or static constants, found within the QCSPCChart library by explicitly referencing that namespace.

```
        let htmlcanvas: QCSPCChartTS.Canvas =
<QCSPCChartTS.Canvas>document.getElementById(canvasid);
        let spccharttype: number =
QCSPCChartTS.SPCControlChartData.MEAN_RANGE_CHART;
        let subgroupsize: number = 5;
```

```
        let numberpointsinview: number = 12;
        let charttitle: string = "XBar-R Example";

        let xbarrchart: QCSPCChartTS.SPCChartBase  =
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
Subgroupsize(htmlcanvas, spccharttype, subgroupsize, numberpointsinview);
        if (!xbarrchart) return;
```

In broad terms, first you create an instance of the control chart using one of the static control chart constructors

```
QCSPCChartTS.SPCBatchVariableControlChart.newSPCBatchVariableControlChartChartType
Subgroupsize
```

passing in the required parameters. TypeScript is very strict on parameter checking so you will need to get them right before the file will compile.

In the example, get and set the chart properties using the xbarrchart instance of the chart returned by the static constructor.

```
    xbarrchart.setEnableCategoryValues(true);
    xbarrchart.setEnableCalculatedValues(true);
    xbarrchart.setEnableAlarmStatusValues(false);
    xbarrchart.setEnableChartToggles(true);
```

This will look very familiar if you have used QCSPCChart on any of the other programming languages we support (C#, VB, and Java, etc.). Since JavaScript and TypeScript do not support the property syntax of .Net, all properties values are assigned and retrieved using setters and getters, i.e. a prefix of set or get in front of the property name.

In many instances you need to get at instances of classes within the SPC chart object, ChartData (an instance of SPCControlChartData), ChartTable (an instance of SPCGeneralizedTableDisplay), PrimaryChart or SecondaryChart (instances of SPCControlPlotObjectData), and ControlLimitValues (an array of SPCControlLimitRecord) to name the most common. Internally, these properties can be null while the chart constructor is instantiating, but once that is done they are never null (or undefined). However, if you use the “strict” mode of TypeScript, it is **not** OK to use code like seen below, where you retrieve the instance of the object using a getter call (getChartData()).

```
    xbarrchart.getChartData().setTitle (charttitle);
    xbarrchart.getChartData().setChartDescriptor("XBar-R");
```

This is not allowed in TypeScript if you have strict null checking enabled, which all of our TypeScript examples do. Strict error checking can be annoying in TypeScript because it flags so many things as errors, but its overall usefulness cannot be overstated. Because the value returned by getChartData() is marked as returning a union of SPCControlChartData and null, the compiler won't let you reference a function in ChartData unless it absolutely sure that the value returned by getChartData() is not null. To work around this you use code like this:

[TypeScript Code]

```
let chartData: SPCControlChartData | null = xbarrchart.getChartData();
if (chartData)
{
    chartData.setTitle (charttitle);
    chartData.setChartDescriptor("XBar-R");
}
```

The equivalent JavaScript code would look like:

[JavaScript Code]

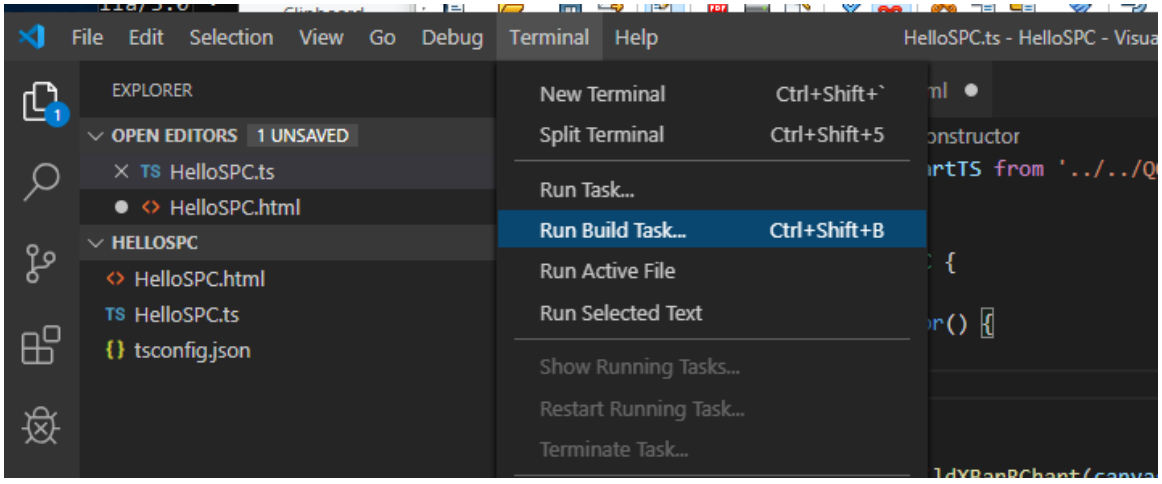
```
var chartData = xbarrchart.getChartData();
if (chartData)
{
    chartData.setTitle (charttitle);
    chartData.setChartDescriptor("XBar-R");
}
```

Note that chartData is defined as SPCControlChartData | null, which mean it can assume the value of a SPCControlChartData object, or null. This allows you to check for null before you access member functions and constants of the chartData object.

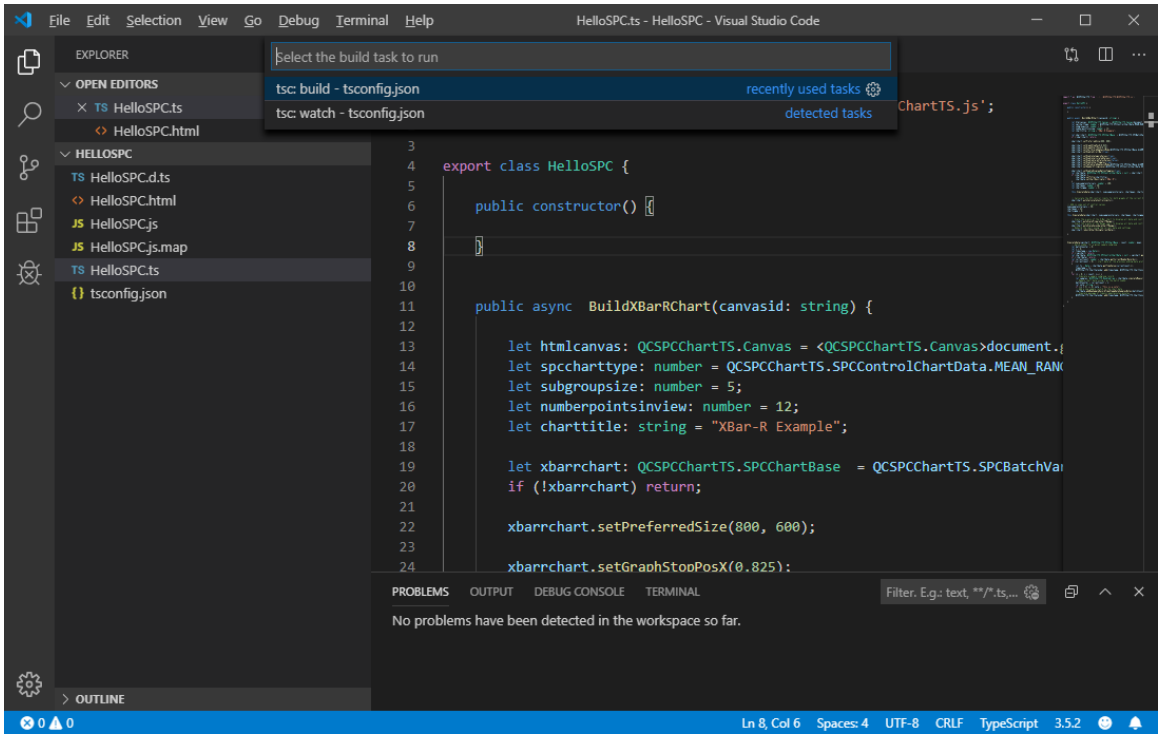
This applies to all of the class instances internal to the main chart class.

Unlike the JavaScript version of the same program, once the HelloSPC.ts TypeScript file is setup correctly you will need to compile it in order to generate the HelloSPC.js file. In order to compile the project you need a tsconfig.json file in the project folder. Just copy the one from the TypescriptExamples/XbarRChart example and place it in the TypescriptExamples/HelloSPC folder. See Chapter 4 for more information about the options found in the tsconfig.json file.

Once that is in place, compile the HelloSPC.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).



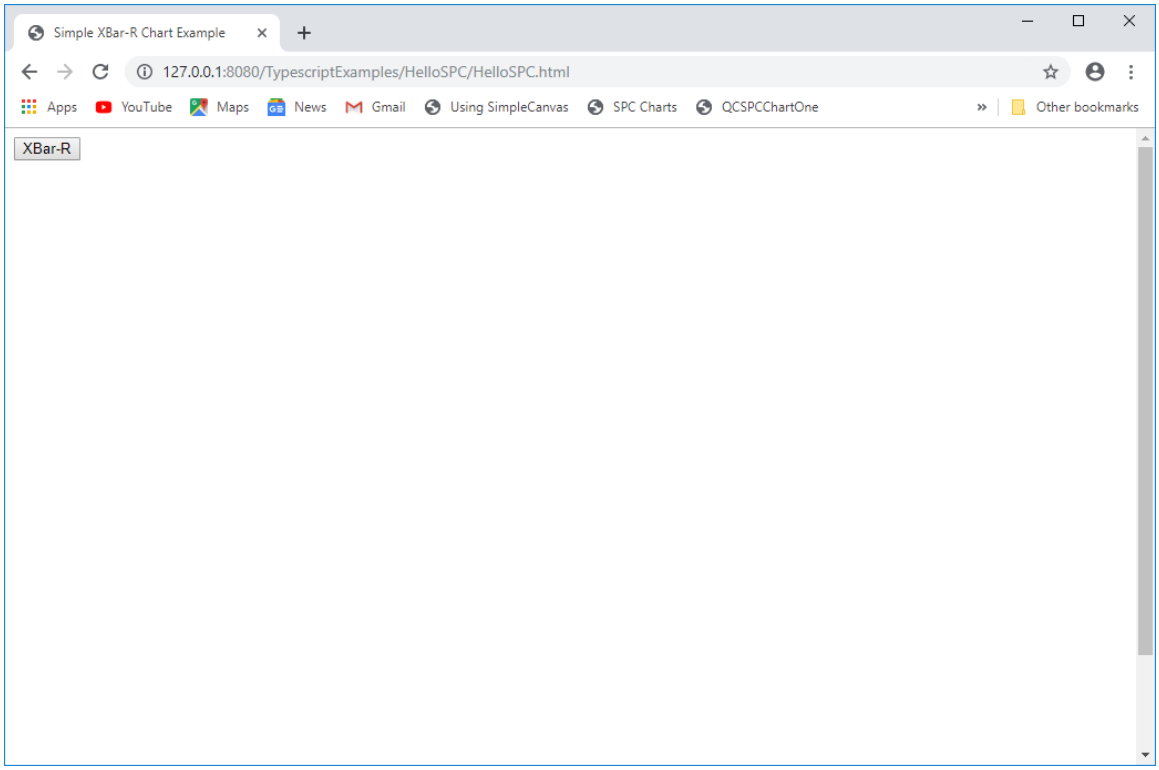
This will use the tsconfig.json file in the current directory to control the TypeScript to JavaScript compilation. It will only take a second. After that you will either have a list of errors to fix, or the JavaScript equivalent of the original TypeScript file, with the filename HelloSPC.js and located in the HelloSPC folder, next to the other files.

Other files also created during compilation are the HelloSPC.d.ts file (contains TypeScript type declarations for the HelloSPC.js file), and the HelloSPC.js.map (contains a mapping of the HelloSPC.js file to the HelloSPC.ts file) which is used in debugging the HelloSPC.ts source when you are actually executing the associated HelloSPC.js JavaScript.

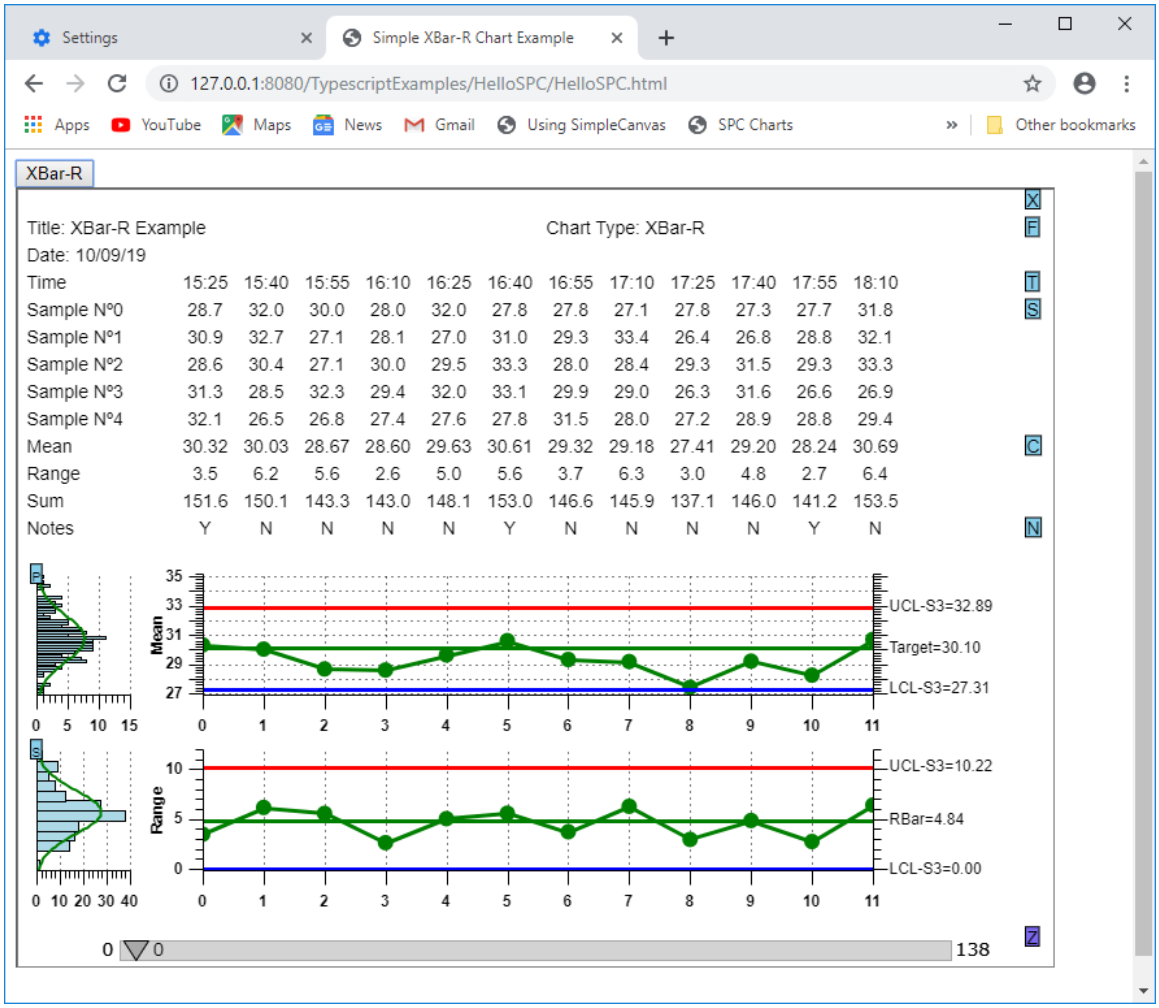
Assuming everything is setup correctly, and you have started some sort of test server which references the Quinn-Curtis/JSTS folder as its root directory, you should be able to run the test program by setting a browser to load the file at the URL:

http://127.0.0.1:8080/TypescriptExamples/HelloSPC/HelloSPC.html

The result will be the following display:



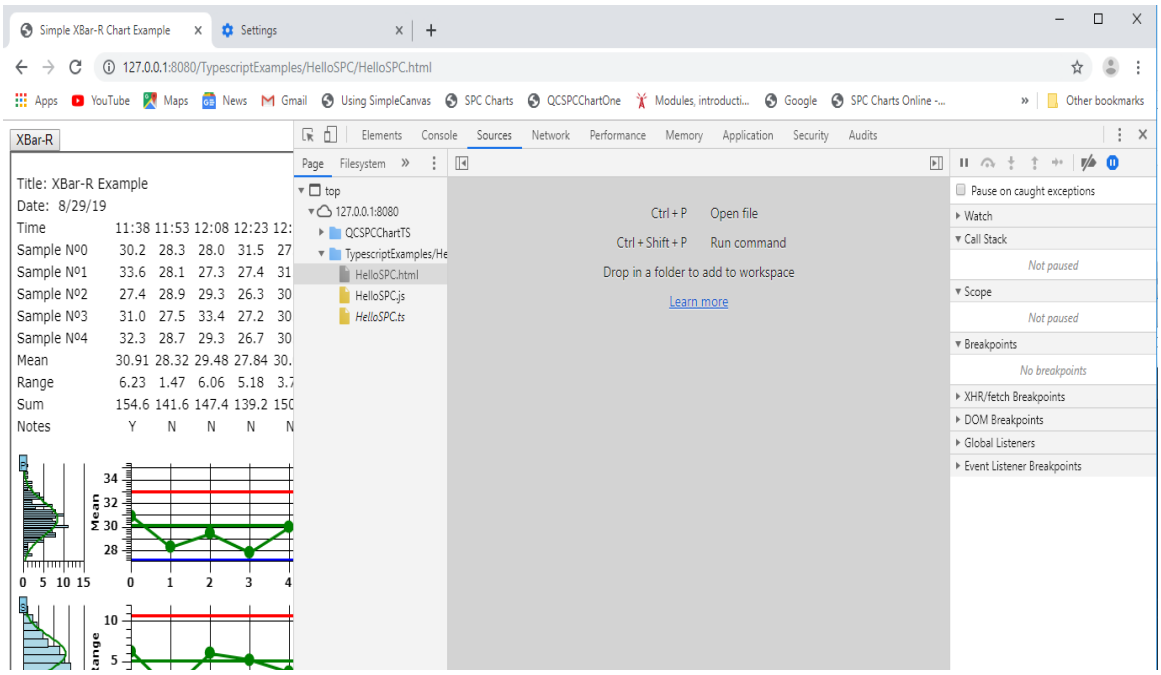
showing just the **Xbar-R** button. Click the button and the chart will appear.



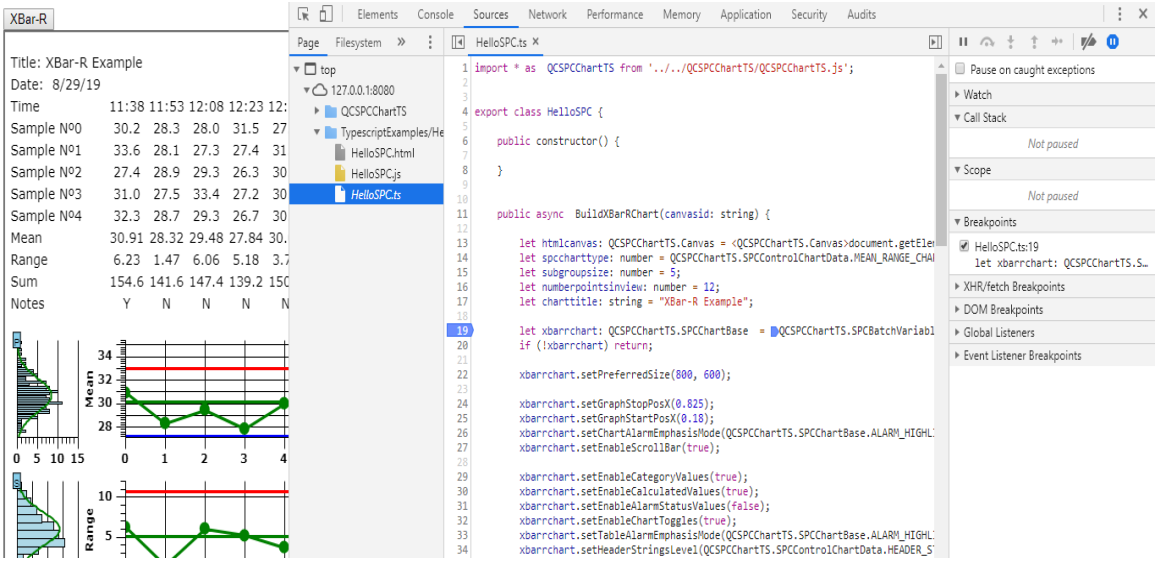
Debugging

If for some reason the chart does not appear you will have start debugging.

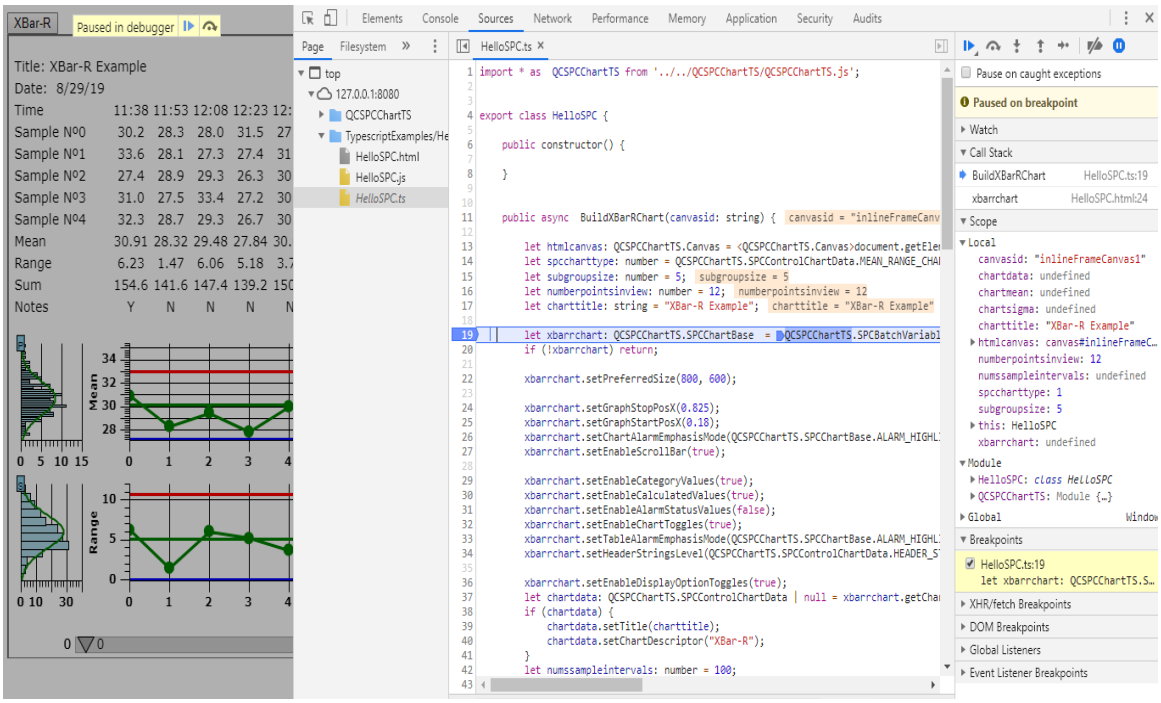
There are ways to setup Visual Studio Code for integrated debugging. But it looks more complicated than we want to explain. Instead, we choose to just use the integrated F12 debugging available in all of the major browsers. Once you point the browser to the HelloSPC.html file you should see the XbarRChart 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 HelloSPC.ts file, HelloSPC.js file, or the HelloSPC.html file to view in the main window. It is the presence of the HelloSPC.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 HelloSPC.ts file, and then click the Xbar-R button in the left hand side of the browser, program execution will stop in the code window when the breakpoint is hit.



Using QCSPCChart 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 QCSPCChartTS is spelled in lower case: qcspcchartts.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/TypescriptExamples/XBarRChart for the XbarRChart example.

Using QCSPCChart 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 SPC 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/TradAspNetJSTSProj.zip. Copy the two projects into the Quinn-Curtis\JSTS\AspNetVS2019 folder . They will have their own copies of the *qcspcchartts.js* library (found in each project root, under the *QCSPCChartTS* 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 qcspcchartts.js file you have in your Quinn-Curtis\JSTS\QCSPCChartTS 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\AspNetVS2019 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 (QCSPCChartTS and VariableControlCharts) which hold the qcspcchartts.js library, and the SPC Variable control charts examples. We then

added some JavaScript inside the HTML of one of the various HTML page templates found in the project. And that displays an Xbar-R chart. In all of the examples, the QCSPCChartTS and VariableControlCharts folders are exactly the same. The VariableControlCharts.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. Here are some note on what to do.

Where to place the QCSPCChartTS and VariableControlCharts 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 QCSPCChartTS library underneath the example program folder, so that the qcspcchartts.js file is accessible to the JavaScript which displays the chart. So you will find the QCSPCChartTS and VariableControlCharts folders duplicated under all of the Asp.Net example program. In order to point the example program VariableControlCharts.js to the qcspcchartts.js library, it uses the the import statement:

```
import * as  QCSPCChartTS from './../QCSPCChartTS/qcspcchartts.js';
```

Rather than place the QCSPCChart 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  QCSPCChartTS from './../Scripts/QCSPCChartTS/qcspcchartts.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 QCSPCChartTS and VariableControlCharts folders underneath the wwwroot folder. We placed the VariableControlCharts folder directly under the wwroot folder, and the QCSPCChartTS folder under the 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 VariableControlCharts.js to the qcspcchartts.js library, it uses the the import statement:

```
import * as  QCSPCChartTS from './../lib/QCSPCChartTS/qcspcchartts.js';
```

at the top of the VariableControlCharts.js. This is the relative path location, from the VariableControlCharts folder, to the qcspcchartts.js file, in the QCSPCChartTS 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 VariableControlCharts.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="spcChartCanvas1" width="800" height="600"></canvas>
</div>

<script type="module">
    import { BuildXBarRChart, AddDataXBarRChart } from
'./VariableControlCharts/VariableControlCharts.js';
    BuildXBarRChart("spcChartCanvas1");

    function updateData() {

        if (document) {
            var mean = Number.parseFloat(document.getElementById("MeanTextBox").value);
            var stddev = Number.parseFloat(document.getElementById("StdDevTextBox").value);
            var numtoadd = Number.parseInt(document.getElementById("NumToAddTextBox").value,
10);

            AddDataXBarRChart(mean, stddev, numtoadd);
        }
        return false;
    }

    document.getElementById("AddDataButton").onclick = updateData;
</script>

<div class="col-md-2">
    <p>
        Mean: <input id="MeanTextBox" type="text" value="31" />
    </p>
    <p>
        Standard Deviation: <input id="StdDevTextBox" type="text" value="5" />
    </p>
    <p>
        Number to add: <input id="NumToAddTextBox" type="text" value="20" />
    </p>
    <p>
        <input id="AddDataButton" type="button" value="Add Data" />
    </p>
    <p>
        <a class="btn btn-default" href="https://go.microsoft.com/fwlink/?LinkId=301948">Learn
more &raquo;</a>
    </p>
</div>
```

Project name	**Modified HTML block**
WebFormApplication1	Added to the default.aspx file
AspWebAppSinglePageWebApplication1	Added to the Views/Home/_Home.cshtml file
AspNetCoreWebApplication1	Added to the Pages/Index.cshtml file
ASPNetCoreMVCWebApplication1	Added to the Views/Home/Index.cshtml file

Note: Since HTML script blocks marked as modules are loaded asynchronous with all other elements of the page, you cannot assign an event (onclick) reference to the button, when the button is created in HTML. Instead you must do it inside the script, so that you are sure that the reference to the JavaScript function exists. That is why there is the line

```
document.getElementById("AddDataButton").onclick = updateData;
```

at the end of the script, assigning the AddDataButton.onclick event handler to the updateData function inside the script block.

Index