

# LightningChart® Ultimate SDK

## User's Manual

## About this document

This document is a brief User's Manual, reference of **Arction LightningChart Ultimate Software Development Kit**. Only essential key features are explained. Hundreds of classes, properties or methods are not described in this document. Run the provided demo applications for getting a quick preview of some LightningChart features. For code examples, take a look at the source code of included demo applications.

All code examples in this document are written in C# language. Majority of demo applications provide code preview in C# and Visual Basic .NET, as well.

**And remember, don't hesitate to contact support ([support@arction.com](mailto:support@arction.com)) if you have any question!**

**Applies to LightningChart Ultimate edition, v.8.3.2**



Copyright Arction Ltd 2009-2018. All rights reserved.

**LightningChart is registered trademark by Arction Ltd.**

[www.arction.com](http://www.arction.com)

[www.lightningchart.com](http://www.lightningchart.com)

# Contents

1. Overview.....	15
1.1 Chart editions .....	15
1.2 Components .....	16
1.3 Namespaces.....	17
2. Installation.....	18
2.1 Before installing.....	18
2.2 Run the setup wizard.....	18
2.3 Adding Arction components manually to Visual Studio Toolbox.....	18
2.4 Configuring Visual Studio 2010-2017 help manually.....	19
2.4.1 Visual Studio 2010 .....	19
2.4.2 Visual Studio 2012-2017.....	20
2.5 Code parameters and tips by Visual Studio IntelliSense .....	21
2.6 Selecting target framework.....	21
3. License management.....	22
3.1 Adding license.....	22
3.2 Removing a license .....	24
3.3 Extracting Deployment Key .....	24
3.4 Applying Deployment Key in your application .....	24
3.5 Running with Deployment Key on development computer.....	26
3.6 Running with debugger .....	26
3.7 Trial period .....	27
3.8 Floating licenses .....	27
4. LightningChartUltimate component.....	28
4.1 Adding from toolbox into Windows Forms project.....	28
4.1.1 Properties .....	28
4.1.2 Event handlers.....	29
4.1.3 Best practices, version updates in mind.....	29
4.2 Adding from toolbox into WPF project .....	29
4.2.1 Properties .....	29
4.2.2 Event handlers.....	30
4.3 Adding into Blend WPF project .....	30
4.3.1 Best practices, version updates in mind.....	31
4.3.2 Preventing blurring of the chart .....	31
4.4 Object model .....	32
4.4.1 Differences between Windows forms and WPF.....	33
4.5 Setting background fill.....	34

4.5.1	Setting transparent background.....	36
4.6	Configuring appearance / performance settings .....	37
5.	ViewXY .....	40
5.1	Axis layout options .....	42
5.1.1	Automatic margins .....	43
5.1.2	Setting how axes are placed .....	43
5.1.2.1	X axis automatic placement.....	43
5.1.2.2	Y axis automatic placement.....	45
5.1.3	Graph segments and Y axes placement in them .....	48
5.1.3.1	Layered .....	48
5.1.3.2	Stacked .....	48
5.1.3.3	Segmented.....	49
5.1.4	Axis grid strips.....	50
5.1.5	Other AxisLayout options .....	51
5.2	Y axes .....	52
5.2.1	AxisY class properties .....	52
5.2.2	Tick value labels formatting.....	52
5.2.3	Value type.....	53
5.2.4	Range setting .....	54
5.2.5	Restoring range.....	54
5.2.6	Divisions.....	54
5.2.7	Grid .....	54
5.2.8	Custom ticks .....	55
5.2.9	Reversed X and Y axis .....	56
5.2.10	Logarithmic axes.....	56
5.2.10.1	Exponential presentation for 10 base .....	56
5.2.10.2	Natural logarithm .....	57
5.2.11	Converting between axis values and screen coordinates .....	57
5.2.12	MiniScale .....	58
5.3	X axis.....	58
5.3.1	Real-time monitoring scrolling .....	58
5.3.1.1	None .....	58
5.3.1.2	Stepping.....	59
5.3.1.3	Scrolling .....	59

5.3.1.4	Sweeping .....	61
5.3.1.5	Triggering.....	61
5.3.2	Scale breaks .....	63
5.4	ViewXY series, general.....	65
5.5	Point line series .....	66
5.5.1	Line style .....	66
5.5.2	Points style.....	67
5.5.3	Coloring points individually .....	67
5.5.4	Adding points.....	67
5.5.5	Adding points, alternative way.....	68
5.6	Sample data series.....	69
5.6.1	Y precision.....	70
5.6.2	Adding points.....	70
5.7	Freeform point line series .....	70
5.8	Advanced line coloring of line series .....	72
5.8.1	Y-value based coloring of line and fill with value-range palette .....	72
5.8.2	Custom shaping and coloring with CustomLinePointColoringAndShaping event.....	73
5.9	High-low series .....	74
5.9.1	Fill, line and point styles .....	74
5.9.2	Limits.....	75
5.9.3	Coloring by value-range palette .....	76
5.9.4	Adding data.....	76
5.10	Area series .....	77
5.10.1	Adding data.....	78
5.11	Bars .....	79
5.12	Stock series .....	82
5.12.1	Setting data to StockSeries .....	83
5.12.2	Setting X axis to date display .....	84
5.12.3	Custom formatting of appearance .....	84
5.12.4	Applying Scale breaks .....	85
5.13	PolygonSeries .....	85
5.13.1	Setting data to a Polygon.....	85
5.13.2	Enabling complex / intersecting fills.....	86
5.14	LineCollections.....	87

5.14.1	Setting data to a LineCollection.....	88
5.15	IntensityGridSeries .....	89
5.15.1	Setting intensity grid data .....	91
5.15.2	Creating intensity grid data from bitmap file .....	92
5.15.3	Fill styles .....	93
5.15.4	Rendering as pixel map.....	93
5.15.5	ValueRangePalette .....	94
5.15.6	Wireframe.....	95
5.15.7	Contour lines .....	96
5.15.8	Contour line labels.....	97
5.16	IntensityMeshSeries .....	98
5.16.1	Setting intensity mesh data, when geometry changes .....	99
5.16.2	Setting intensity mesh data, when geometry does not change .....	100
5.16.2.1	Creating the series and geometry .....	100
5.16.2.2	Updating the values periodically .....	100
5.17	Bands .....	101
5.18	Constant lines .....	101
5.19	Annotations .....	102
5.19.1	Controlling target and location.....	103
5.19.2	Using mouse to move, rotate and resize.....	104
5.19.3	Adjusting appearance .....	104
5.19.4	Size settings .....	105
5.19.5	Keeping text area visible.....	105
5.19.6	Displaying annotation over axes.....	105
5.19.7	Clipping inside graph .....	105
5.19.8	Controlling the Z order .....	106
5.19.9	LayerGrouping performance optimization .....	106
5.19.10	Converting between axis values and screen coordinates .....	106
5.20	Legend box.....	107
5.20.1	Hiding / showing a series from legend box .....	107
5.20.2	Preventing a series from listing itself in the legend box .....	108
5.20.3	Selecting in which legend box to show a specific series .....	108
5.20.4	Hiding check boxes .....	108
5.20.5	Hiding icons .....	108

5.20.6	Hiding intensity series palette scales.....	108
5.20.7	Controlling positions.....	109
5.20.8	Allocating space for legend boxes between graph segments .....	109
5.20.9	Alignment of legend boxes in segment gap .....	110
5.20.10	Selecting in which graph segment to show a legend box .....	110
5.20.11	Horizontal alignment of several legend boxes sharing the same margin .....	110
5.20.12	Resizing and moving legend boxes .....	111
5.21	Zooming and panning.....	112
5.21.1	Zooming with touch screen .....	113
5.21.2	Panning with touch screen .....	113
5.21.3	Left mouse button action .....	113
5.21.4	Right mouse button action .....	113
5.21.5	RightToLeftZoomAction.....	114
5.21.6	Zooming with mouse button .....	114
5.21.6.1	Zoom in/out by clicking .....	114
5.21.6.2	Zoom in with rectangle.....	115
5.21.6.3	Configuring zoom out rectangle .....	115
5.21.7	Zooming with mouse wheel .....	115
5.21.8	Zooming and panning with mouse wheel, over axis .....	115
5.21.9	Panning with mouse button .....	116
5.21.10	Enabling/disabling Ctrl, Shift and Alt.....	116
5.21.11	Zoom in/out with code .....	116
5.21.12	Zooming an axis by code .....	116
5.21.13	Rectangle zooming about a configurable origin.....	116
5.21.14	Linking Y axes zoom with same units .....	117
5.21.15	Automatic Y fit.....	118
5.21.16	Aspect ratio.....	118
5.21.17	Excluding specific X or Y axes from zooming and panning operations.....	119
5.22	DataBreaking by NaN or other value.....	120
5.23	ClipAreas.....	122
5.24	Maps .....	123
5.25	Vector maps.....	124
5.25.1	Selecting active map.....	124
5.25.2	Aspect ratio.....	125

5.25.3	Layers and their appearance settings.....	126
5.25.3.1	Setting individual fill and border style for each layer item .....	127
5.25.4	Mouse interactivity.....	128
5.25.5	Background photos.....	129
5.25.6	Combining other series with maps.....	130
5.25.7	Importing maps from ESRI shape file data .....	132
5.25.7.1	Programming interface for importing shp data .....	132
5.25.7.2	Dialogs .....	133
5.25.7.2.1	Shapefile Selection Dialog .....	133
5.25.7.2.2	Select Record Encoding and Invalid Name Fields .....	134
5.25.7.2.3	Layer data selection dialog.....	135
5.25.7.2.4	Item filter .....	137
5.25.8	Importing and replacing map layers.....	137
5.26	Tile maps.....	139
5.26.1	HERE.....	140
5.27	Line series cursors .....	141
5.27.1	Solving the data values in the position of LineSeriesCursor.....	142
5.27.1.1	Accurate method, solving Y value by X value, by using the data points array .....	143
5.27.1.2	Coarse method, solving Y screen coordinate by X coordinate, by using the data points array	143
5.28	Event markers.....	144
5.28.1	Chart event markers .....	145
5.28.2	Line series event markers.....	146
5.29	Persistent series rendering layers .....	147
5.29.1	Creating the layer .....	148
5.29.2	Clearing the layer.....	148
5.29.3	Adjusting layer alpha .....	149
5.29.4	Rendering data into the layer.....	149
5.29.5	Disposing the layer .....	149
5.29.6	Anti-aliasing data in the layer .....	150
5.29.7	Getting list of layers.....	150
5.29.8	Some layer limitations you should be aware of .....	150
5.30	Persistent series rendering intensity layers .....	151
5.30.1	Creating the layer .....	152

5.30.2	Clearing the layer.....	152
5.30.3	Changing palette colors.....	152
5.30.4	Adjusting the intensity effect of new trace and decay of old traces.....	152
5.30.5	Rendering data into the layer.....	153
5.30.6	Disposing the layer .....	153
5.30.7	Anti-aliasing data in the layer.....	153
5.30.8	Getting list of layers.....	153
6.	View3D.....	154
6.1	3D model and dimensions.....	155
6.1.1	World coordinates.....	155
6.2	Walls.....	156
6.3	FrameBox.....	157
6.4	Camera.....	157
6.4.1	Predefined cameras.....	159
6.5	Lights.....	159
6.5.1	Directional light .....	159
6.5.2	Point of light .....	159
6.5.3	Lights and materials.....	160
6.5.4	Predefined lighting schemes .....	161
6.6	Axes .....	162
6.6.1	Location .....	162
6.6.2	Orientation .....	163
6.6.3	CornerAlignment .....	164
6.7	3D series, general .....	164
6.8	PointLineSeries3D.....	165
6.8.1	Point styles.....	165
6.8.2	Line styles .....	167
6.8.3	Adding points.....	167
6.8.3.1	Points .....	168
6.8.3.2	PointsCompact.....	168
6.8.4	Coloring points individually .....	169
6.8.5	Setting points sizes individually.....	169
6.8.6	Multi-coloring line .....	170
6.8.7	Displaying millions of scatter points.....	170

6.9	SurfaceGridSeries3D.....	172
6.9.1	Setting surface grid data.....	173
6.9.2	Creating surface from bitmap file.....	174
6.9.3	Fill styles .....	175
6.9.4	Contour palette .....	177
6.9.5	Wireframe mesh.....	178
6.9.5.1	Some notes when using wireframe simultaneously with fill.....	180
6.9.6	Contour lines .....	181
6.9.7	Scrolling surface data .....	182
6.10	SurfaceMeshSeries3D.....	185
6.10.1	Setting surface mesh data .....	186
6.10.2	Visualizing point clouds in 3D .....	187
6.11	WaterfallSeries3D.....	188
6.12	BarSeries3D .....	189
6.12.1	Bars grouping.....	189
6.12.2	Bar styles.....	192
6.12.3	Setting bar series data .....	194
6.12.4	Showing bars horizontally .....	194
6.13	MeshModels .....	196
6.13.1	Loading a model .....	196
6.13.2	Positioning, scaling and rotating the model .....	197
6.13.3	Enabling fill and wireframe.....	197
6.13.4	Custom-coloring fill.....	198
6.13.5	Custom-coloring wireframe.....	199
6.13.6	Reverse vertices winding order .....	199
6.13.7	Constructing MeshModel programmatically from vertices .....	200
6.13.7.1	Updating the bitmap fill efficiently.....	201
6.13.8	Tracing the model with mouse.....	201
6.14	VolumeModels .....	202
6.14.1	Loading of data .....	203
6.14.2	Properties .....	203
6.14.3	Ray Function .....	205
6.14.4	Threshold.....	207
6.14.5	Slice Range.....	208

6.14.6	Sampling Rate Options .....	209
6.14.7	Smoothness .....	211
6.14.8	EmptySpaceSkipping.....	212
6.14.9	Opacity.....	213
6.14.10	Brightness and Darkness .....	214
6.15	Rectangle3D objects .....	214
6.16	Polygon3D objects.....	216
6.17	Zooming, panning and rotating .....	218
6.17.1	Mouse wheel zooming .....	218
6.17.2	Box zooming .....	218
6.17.3	Rotating and panning .....	219
6.17.4	Zooming with touch screen .....	220
6.17.5	Panning with touch screen .....	220
6.17.6	Using mouse wheel over an axis .....	220
6.17.7	Zooming, rotating and panning by code.....	220
6.18	Clipping objects within axis ranges.....	220
6.19	Annotations .....	221
7.	ViewPie3D.....	222
7.1	Properties .....	223
7.2	Pie slices.....	223
7.3	Setting data by code.....	224
7.4	Viewing pie chart in 2D.....	225
8.	ViewPolar.....	226
8.1	Axes .....	227
8.1.1	Reversed axes .....	228
8.1.2	Setting rotation angles of the scales .....	229
8.1.3	Setting divisions.....	230
8.2	PointLineSeries .....	231
8.2.1	Setting data.....	231
8.2.2	Palette coloring.....	232
8.2.3	Custom shaping and coloring with CustomLinePointColoringAndShaping event.....	232
8.3	AreaSeries.....	233
8.3.1	Setting data.....	233
8.4	Sectors .....	234
8.5	Annotations .....	234

8.6	Markers.....	235
8.7	Zooming and panning.....	236
9.	ViewSmith.....	237
9.1	Axis.....	237
9.2	PointLineSeries .....	242
9.3	Setting data.....	243
9.4	Annotations .....	244
9.5	Markers.....	244
10.	Setting color theme .....	246
11.	Scrollbars .....	246
12.	Export and printing.....	248
12.1.1	Bitmap image export .....	248
12.1.2	Vector image export.....	248
12.1.3	Copy to clipboard.....	248
12.1.4	Capturing to byte array .....	249
12.1.5	Setting output stream for continuous frame writing .....	249
12.1.6	Printing .....	250
13.	LightningChart performance .....	251
13.1	Selecting the correct API edition .....	251
13.2	Set the rendering options correctly.....	251
13.3	Updating chart data or properties.....	251
13.4	Line series tips .....	252
13.5	Intensity series tips.....	253
13.6	3D surface series tips.....	253
13.7	Maps tips .....	253
13.8	Hardware .....	254
14.	LightningChart error and exception handling .....	254
15.	ChartManager component .....	255
15.1	Chart interoperation, drag-drop.....	255
15.2	Memory management enhancement .....	255
16.	SignalGenerator component .....	256
16.1	Sampling frequency, Output interval and Factor .....	256
16.2	Sine waveforms .....	257
16.3	Square waveforms.....	258
16.4	Triangle waveforms .....	259
16.5	Noise waveforms.....	260
16.6	Frequency sweeps .....	260
16.7	Amplitude sweeps .....	261

16.8	Starting and stopping .....	261
16.9	Multi-channel generator with master-slave configuration .....	261
16.10	Output data stream .....	262
17.	SignalReader component .....	263
17.1	Key properties .....	263
17.2	Opening file quickly for playback .....	263
18.	AudioInput component .....	265
18.1	Properties .....	265
18.2	Methods .....	265
18.3	Events .....	266
18.4	Usage (WinForms) .....	266
18.4.1	Creation .....	266
18.4.2	Event handling .....	266
18.4.3	Configuring .....	267
18.4.4	Starting .....	267
18.4.5	Stopping.....	268
18.5	Usage (WPF) .....	268
18.5.1	Creation .....	268
19.	AudioOutput component .....	269
19.1	Properties .....	269
20.	SpectrumCalculator component .....	270
21.	Headless mode .....	272
21.1.1	Headless Rendering .....	272
21.1.1.1	Additional initialization options.....	272
21.1.1.2	Capturing images.....	273
21.1.2	Limitations and Requirements .....	274
21.1.2.1	Threads.....	274
21.1.2.2	Chart Update .....	274
21.1.2.3	Engine support.....	274
21.1.2.4	Licensing .....	274
21.1.3	Example solution .....	275
22.	Using Windows Forms chart in WPF application.....	277
22.1	How about using Arction Windows Forms controls in WPF? .....	277
22.2	Should I use Arction.WinForms.LightningChartUltimate in WPF? .....	277
23.	Using LightningChart in C++ applications .....	280
23.1	Install required C++/CLR packages .....	280
23.2	Setting Visual Studio project .....	281

23.3	Creating LightningChart application in C++ project.....	283
24.	Dispose pattern .....	286
24.1	Chart created in code .....	286
24.1.1	Chart disposing .....	286
24.1.2	Objects disposing.....	286
25.	Object model notes .....	287
25.1	Sharing objects between other objects.....	287
26.	Deployment / distribution of LightningChart assemblies.....	289
26.1	Referenced assemblies .....	289
26.2	License key.....	290
26.3	Obfuscating your application code.....	290
26.4	Obfuscating LightningChart code .....	290
26.5	XML files of Arction assemblies .....	290
27.	Troubleshooting .....	291
27.1	Updating from older version .....	291
27.2	Web support.....	293
27.3	Running in Virtual Machine platforms .....	293
28.	Credits.....	293
28.1	Intel Math Kernel library .....	293
28.2	Open-source projects .....	293

# 1. Overview

LightningChart Ultimate SDK is an add-on to Microsoft Visual Studio, consisting of data visualization related software components and tool classes for *WPF (Windows Presentation Foundation) and Windows Forms .NET* platforms.

Arction components are delivered for serious scientific, engineering, measurement and trading solutions, execution performance and very advanced features in special focus.

LightningChart components use low-level DirectX9 and DirectX11 GPU acceleration instead of slower GDI/GDI+ or WPF Graphics APIs. LightningChart has fallback to DirectX11/DirectX10 WARP software rendering when GPU is not accessible, such as in some virtual machine platforms.

## 1.1 Chart editions

For WPF, LightningChart component is available in different binding level editions, to balance between different performance and MVVM (Model-View - View-Model) bindability needs.

Chart edition	Properties binding	Series data binding	Per-data-point binding	Performance
<i>WPF (non-bindable)</i>	No	No	No	Excellent
<i>WPF (semi-bindable)</i>	Yes	Yes	No	Very good
<i>WPF (bindable)</i>	Yes	Yes	Yes	Good
<i>WinForms</i>	No	No	No	Best

Table 1-1. Bindability and performance matrix.

**As a general starting point, Arction recommends Semi-bindable API.**

- For best performance in WPF and multithreading benefits, select non-bindable chart.
- For good tradeoff between WPF bindability and performance, select semi-bindable chart.
- For full WPF MVVM design pattern support, select fully bindable chart.

Semi-bindable chart API is very similar to LightningChart v.6's WPF chart, but comes with extended properties binding that cover also objects created in collections.

Different chart editions can be used in the same application. So it's possible to make basic charts with fully bindable chart and bind the data and for performance-critical tasks, use the non-bindable chart.

Semi-bindable and binadble charts collection properties (such as ViewXY axes, 3D lights) are empty by default so it will support XAML editor in full. In Non-bindable and WinForms collections are prefilled with default items.

**NOTE! Non-Bindable WPF chart is not intended to be configured in XAML at all. Use it in code-behind.**

## 1.2 Components

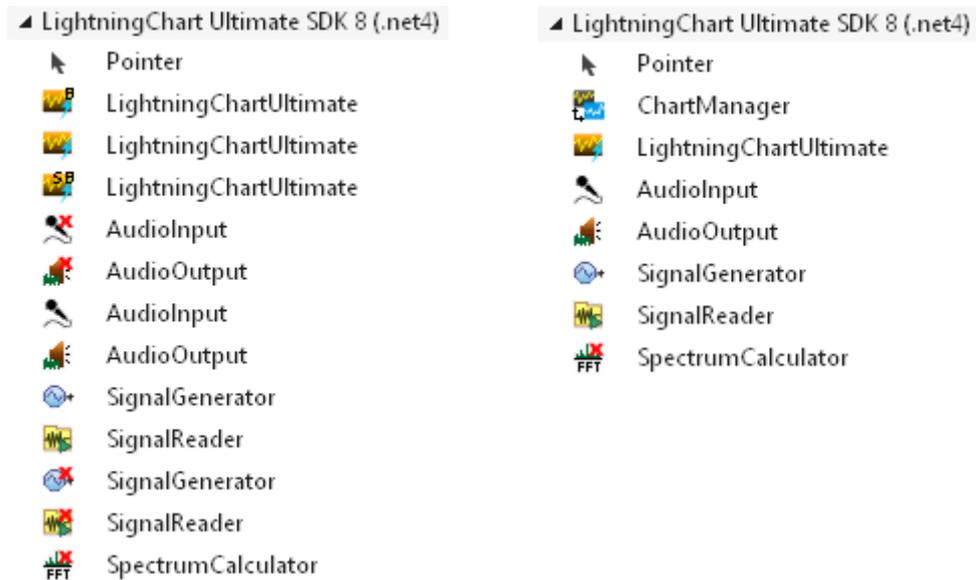


Figure 1-1. On the left, WPF toolbox components. On the right, WinForms toolbox components

### Charting assembly

 **LightningChartUltimate** The chart component. Visualizes data in various presentations.

*In top corner of icon, SB = Semi-bindable WPF chart and B = Bindable WPF chart*

 **ChartManager** Controls interoperation of multiple charts components and real-time measurement memory management. See chapter 15.

### SignalTools assembly

Components that don't have an UI, are marked with **X**.

 **AudioInput** Reads waveform audio stream from a sound device. Line-in or microphone-in connectors are typical options available in a sound device. The real-time stream can be forwarded to other controls. See chapter 18.

-  **AudioOutput** Plays back real-time data stream through the sound device, to speakers or line-output for example. It doesn't have to be an audio stream, any sampled real-time signal can be used. See chapter 19.
-  **SignalGenerator** Generates signal from multiple configurable waveform components. See chapter 16.
-  **SignalReader** Reads waveform data from a signal file, such as PCM formatted WAV. See chapter 17.
-  **SpectrumCalculator** Converts signal data (time domain) to spectrum (frequency domain), by using FFT (Fast Fourier Transform). Also contains methods for backwards conversion, frequency domain to time domain. See chapter 20.

### 1.3 Namespaces

Chart edition	Assembly name	Namespace root	XML namespace
<i>WPF (non-bindable)</i>	Arction.Wpf.Charting. LightningChartUltimate.dll	Arction.Wpf. Charting	<code>xmlns:lcunb="http://schemas.arction.com/charting/ultimate/"</code>
<i>WPF (semi-bindable)</i>	Arction. Wpf.SemibindableCharting. LightningChartUltimate.dll	Arction.Wpf. SemibindableCharting	<code>xmlns:lcusb="http://schemas.arction.com/semibindablecharting/ultimate/"</code>
<i>WPF (fully bindable)</i>	Arction. Wpf.BindableCharting. LightningChartUltimate.dll	Arction.Wpf. BindableCharting	<code>xmlns:lcufb="http://schemas.arction.com/bindablecharting/ultimate/"</code>
<i>WinForms</i>	Arction. WinForms.Charting. LightningChartUltimate.dll	Arction.WinForms. Charting	N/A

Table 1-2. Assembly names and namespaces of all LightningChart Ultimate editions.

## 2. Installation

### 2.1 Before installing

Check your computer configuration meets the requirements

- DirectX 9.0c (shader model 3) level graphics adapter or newer, or DirectX11 compatible operating system for rendering without graphics hardware. DirectX11 compatible graphics hardware recommended.
- Windows Vista, 7, 8 or 10, as 32 bit or 64 bit, and Windows Server 2008 R2 or higher
- Visual Studio 2010-2017 for development, not required for deployment
- .NET framework v. 4.0 or newer installed

### 2.2 Run the setup wizard

Right-click on the **LightningChart Ultimate SDK v8.exe**. The setup will install the components into Visual Studio toolbox. Also it installs the help files associated with the toolbox controls. If components or help install fails, install them manually as instructed the following sections.

If you are trialing LightningChart, you probably are using **SetupDownloader.exe**, which downloads and installs the SDK, and running LightningChart Ultimate SDK v8.exe explicitly is not required.

### 2.3 Adding Arction components manually to Visual Studio Toolbox

#### WinForms

1. Open Visual studio. Create a new **WinForms** project. Right-click on Toolbox, select **Add Tab** and give name "Arction"
2. Right-click on Arction tab, and select **Choose items...**
3. In **Choose Toolbox items** window, Select **.NET Framework Components** page. Click **Browse...**

Browse **Arction.WinForms.Charting.LightningChartUltimate.dll** and **Arction.WinForms.SignalProcessing.SignalTools.dll**, from the folder you installed the components, typically **C:\Program Files (x86)\Arction\LightningChart Ultimate SDK v.8\LibNet4**, and click open. The components can be now found in the toolbox.

## WPF

1. Open Visual studio. Create a new **WPF** project. Right-click on Toolbox, select **Add Tab** and give name "Arction"
2. Right-click on Arction tab, and select **Choose items...**
3. In **Choose Toolbox items** window, Select **WPF Components** page. Click **Browse...**

Browse **Arction.Wpf.Charting.LightningChartUltimate.dll**, **Arction.Wpf.SemibindableCharting.LightningChartUltimate.dll**, **Arction.Wpf.BindableCharting.LightningChartUltimate.dll** and **Arction.Wpf.SignalProcessing.SignalTools.dll**, from the folder you installed the components, typically **C:\Program Files (x86)\Arction\LightningChart Ultimate SDK v.8\LibNet4**, and click open. The components can be now found in the toolbox.

## 2.4 Configuring Visual Studio 2010-2017 help manually

This chapter gives you the information how to install LightningChart Ultimate help content manually. You need this information if Visual Studio 2010-2017 does not have any local help content installed. When you install LightningChart Ultimate and there isn't any local help content installed, LightningChart Ultimate's help will not install.

After these steps you are able to view LightningChart Ultimate's help from Visual Studio 2010-2017. You can either press F1 on LightningChart Ultimate's classes, properties etc. or use Microsoft Help Viewer to browse the help content.

### 2.4.1 Visual Studio 2010

Follow these steps to manually install LightningChart Ultimate help content on Visual Studio 2010:

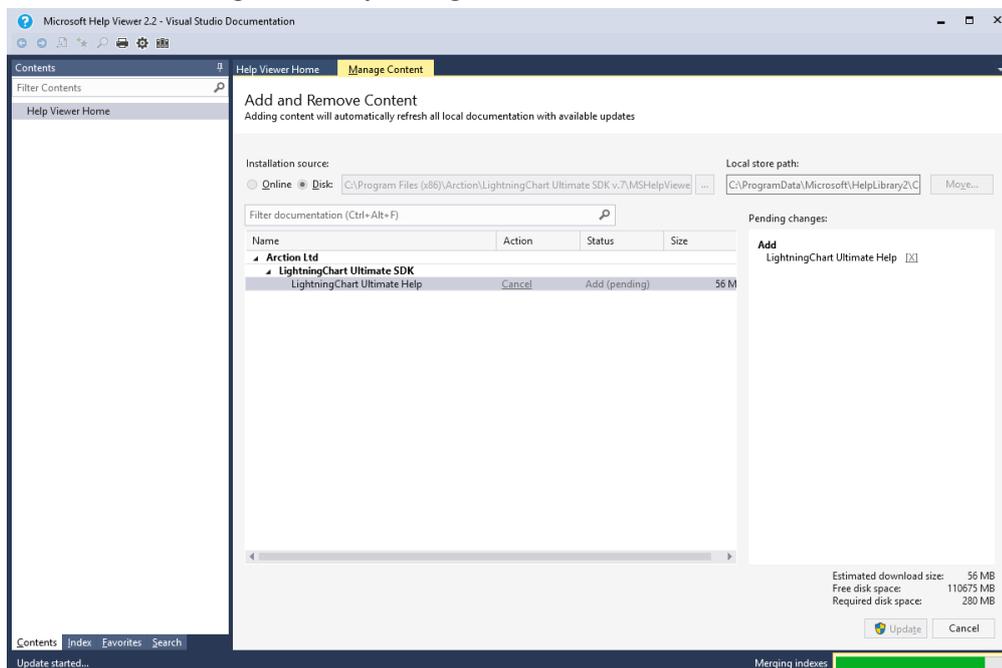
1. Open Visual Studio 2010.
2. Select **Help -> Manage Help Settings**.
3. On Help Library Manager, click **Settings** link.
4. Make sure that **I want to use local help** is selected.
5. If **I want to use local help** is selected, click **Cancel** to go back to Help Library Manager. Otherwise click **OK**.
6. Click **Install content from disk** link.
7. Click **Browse** button and go to the folder where LightningChart Ultimate is installed, by default the path is **C:\Program Files (x86)\Arction\LightningChart Ultimate SDK v.8\MSHelpViewer**.
8. Select **HelpContentSetup.msha** and click **Open** button.
9. Click **Next** button.

10. Next to LightningChart Ultimate Help there is **Add** link. Click it and make sure that **Status** column value changes to **Update Pending**.
11. Click **Update** button. If Help Library Manager asks you if you want to proceed, click **Yes** button. Help library update begins.
12. After help library is updated, click **Finish** button to close Help Library Manager.

## 2.4.2 Visual Studio 2012-2017

Follow these steps to manually install LightningChart Ultimate help content on Visual Studio 2012-2017:

1. Open Visual Studio 2012, 2013, 2015 or 2017.
2. Select **HELP -> Add and Remove Help Content**.
3. After Microsoft Help Viewer starts, select **Manage Content**.
4. Select **Disk** under **Installation source**.
5. Click the button with three dots to browse files.
6. Go to the folder where LightningChart Ultimate is installed, by default the path is **C:\Program Files (x86)\Arction\LightningChart Ultimate SDK v.8\MSHelpViewer**
7. Select **HelpContentSetup.msha** and click **Open** button.
8. Next to LightningChart Ultimate Help there is **Add** link. Click it and make sure that **Status** column value changes to **Add pending**.



9. Click **Update** button. If Help Library Manager asks you if you want to continue, click **Yes** button. Help library update begins.
10. After help library is updated you can close Microsoft Help Viewer.
11. In Visual Studio Menu / Help, select Set **Help Preference : Launch in Help Viewer**

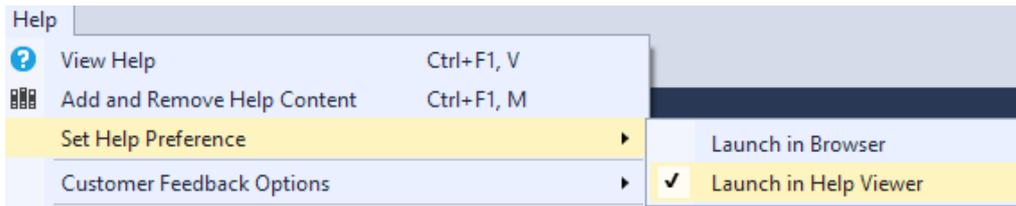


Figure 2-1. Setting help preference.

## 2.5 Code parameters and tips by Visual Studio IntelliSense

IntelliSense may not show code hints when typing LightningChart related code, if the LightningChartUltimate.dll file is referenced from Global Assembly Cache and the controls are not installed by the automatic toolbox installer. Remove the LightningChartUltimate.dll file from References list of your project. Then add it again by browsing from the install directory (typically *C:\Program files (x86)\Arction\LightningChart Ultimate SDK v.8\LibNet4*)

## 2.6 Selecting target framework

In C# project, the framework selection can be made in **Project -> Properties -> Application -> Target framework**.

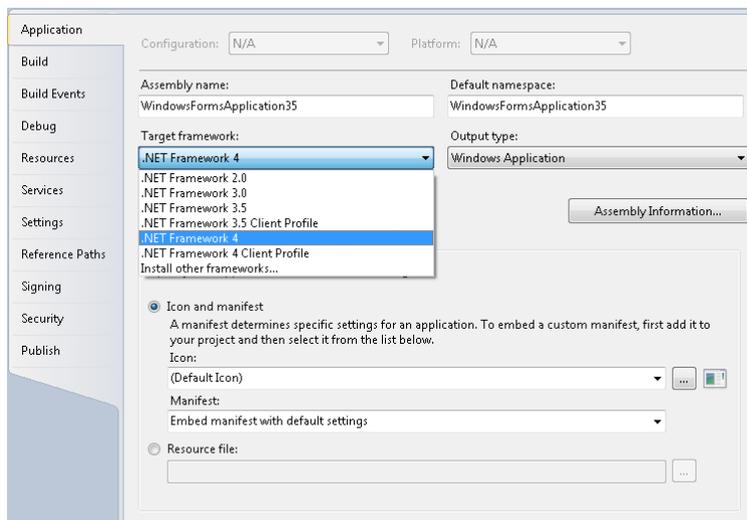


Figure 2-2. Selecting target framework in C# project.

In Visual Basic project, the framework can be selected in **Project -> Options -> Compile -> Advanced compile options -> Target framework**.

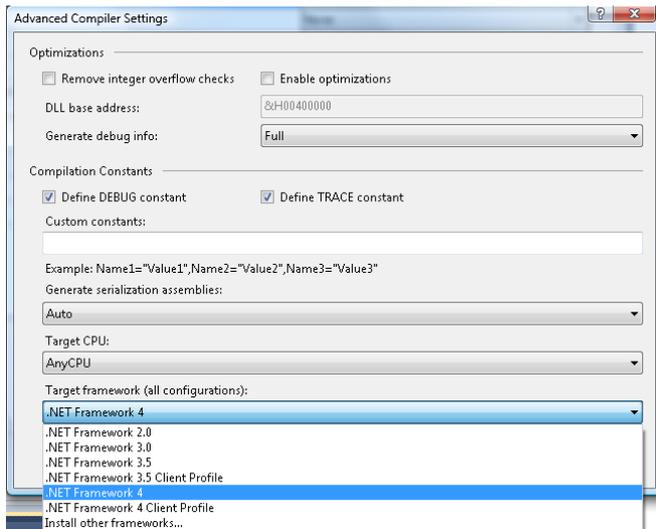


Figure 2-3. Selecting target framework in Visual Basic project.

- Select **.NET Framework 4**, **.NET Framework 4 Client Profile** or **.NET Framework 4.5, or 4.6**.

The LightningChart Ultimate SDK controls will appear in the Visual Studio toolbox only if the correct .NET framework is selected.

### 3. License management

#### 3.1 Adding license

Manage licenses by running the License Manager application, from the Windows start menu Programs / Arction / LightningChart Ultimate SDK / **License Manager**.

Arction components use a license key protection system. You can use the components only with a valid license. License has information of:

- Enabled features, such as ViewXY, View3D, ViewPie3D, Maps, ViewPolar, ViewSmith, Volume Rendering, Signal Tools
- WPF / WinForms / Both technologies
- To how many computers the license can be activated to (1 as standard).
- Subscription expiry date (updates and support ending date)
- Tech support inclusivity
- Per-developer license or Floating license
- Student license

When you drag an Arction component from Toolbox into your application first time, you may be asked to give a license key, in license manager window. Add all your license keys at once from the license file you were delivered. Click **Add from a file...** and browse your **.alf** file.

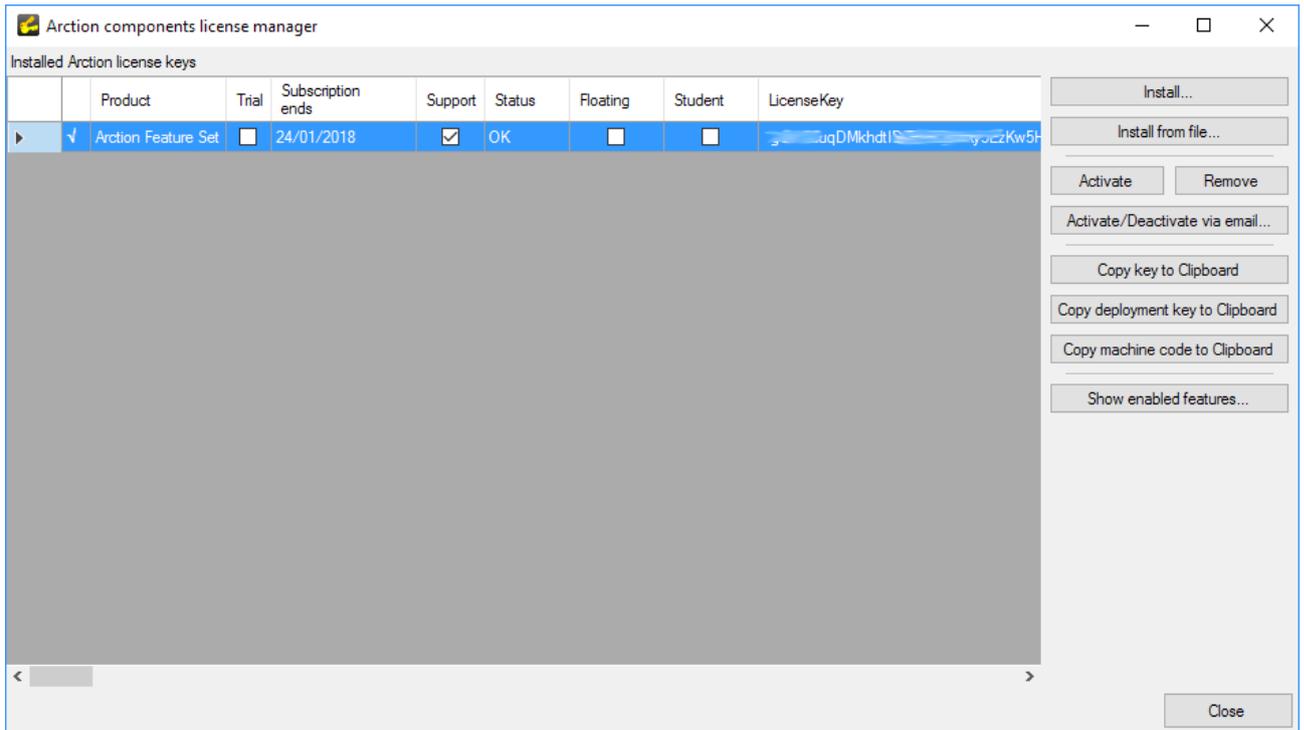


Figure 3-2. Add your license key in LicenseManager.

Per-developer licenses are activated to Arction License Server over internet automatically after adding the license.

If internet connection is not available, use “Activate / Deactivate via e-mail” function.

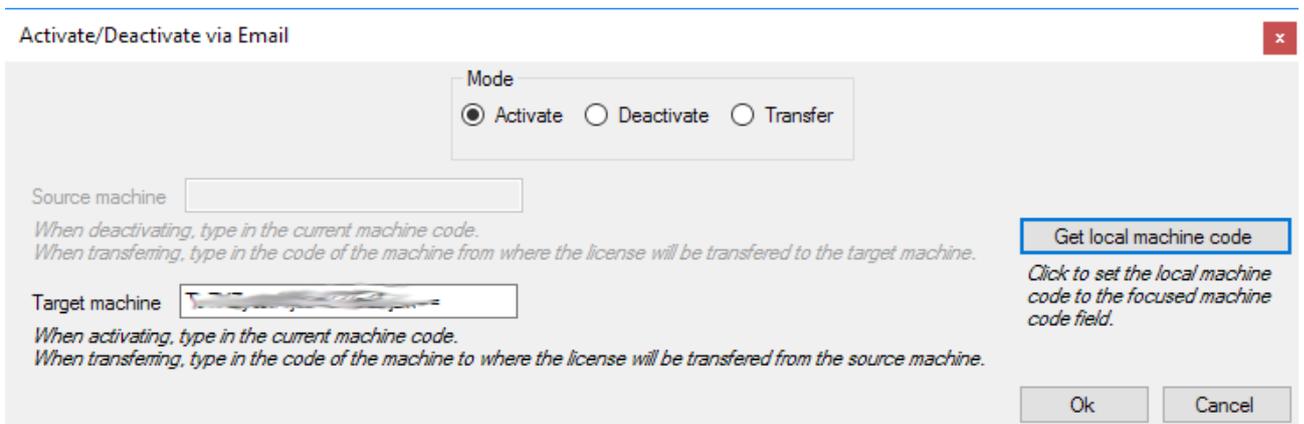


Figure 3-3. Activating a license off-line by e-mail.

Follow on-screen instructions to send e-mail message to Arction licensing team at [licensing@arction.com](mailto:licensing@arction.com).

Arction will provide instructions how to install the license off-line. Expect a reply in 2 business days.

**Note, activation/deactivation over telephone is not available, as the key codes contain thousands of characters.**

**Note, from LightningChart v.7.1 onwards, ChartManager component does not need a license key.**

**Note, from LightningChart v.8.0 onwards, LIC format license keys are not supported. ALF license is needed. If you haven't received ALF license, please contact Arction.**

## 3.2 Removing a license

License can be removed from the system with Remove button. It needs on-line connection to deactivate it automatically. If internet connection is not available, use *“Activate / Deactivate via e-mail” function. Use Mode = Deactivate.*

After the license has been deactivated, it can be installed into another computer.

## 3.3 Extracting Deployment Key

To be able to run LightningChart applications in computers you deploy your software into, a Deployment Key has to be applied in code. Deployment Key can be extracted from a license key by pressing **Copy deployment key to Clipboard** button.

For more detailed instruction on deployment, see chapter 26.

## 3.4 Applying Deployment Key in your application

In code, use static **SetDeploymentKey** methods for the components. Call the **SetDeploymentKey** methods somewhere before the components need to be used. The best place to call it would be *static constructor* of the class you use the chart, or in the application's main class.

## WinForms

Here's an example how to apply it at the static constructor method of the Program class, that is created by default for every WinForms application.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    static class Program
    {
        static Program()
        {
            //Set Deployment Key for Arction components
            string deploymentKey = "VMaLgCAA06k01RgiNIBJABVcG.R..Kikfd...";

            Arction.WinForms.Charting.LightningChartUltimate.SetDeploymentKey(deploymentKey);
            Arction.WinForms.SignalProcessing.SignalGenerator.SetDeploymentKey(deploymentKey);
            Arction.WinForms.SignalProcessing.AudioInput.SetDeploymentKey(deploymentKey);
            Arction.WinForms.SignalProcessing.AudioOutput.SetDeploymentKey(deploymentKey);
            Arction.WinForms.SignalProcessing.SpectrumCalculator.SetDeploymentKey(deploymentKey);
            Arction.WinForms.SignalProcessing.SignalReader.SetDeploymentKey(deploymentKey);
        }

        // Rest of the class ...
    }
}
```

## WPF

Here's an example how to apply it in the beginning of App.xaml.cs, at the static constructor of the App class.

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Windows;
using Arction.Wpf.SignalProcessing;

namespace WpfApplication1
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {
        static App()
        {
            // Set Deployment Key for Arction components
            string deploymentKey = "- DEPLOYMENT KEY FROM LICENSE MANAGER";
        }
    }
}
```

```

GOES HERE-";

// Set Deployment Key for fully bindable chart, if you use it
Arction.Wpf.BindableCharting.LightningChartUltimate
.SetDeploymentKey(deploymentKey);

// Set Deployment Key for semi-bindable chart, if you use it
Arction.Wpf.SemibindableCharting.LightningChartUltimate
.SetDeploymentKey(deploymentKey);

// Set Deployment Key for non-bindable chart, if you use it
Arction.Wpf.Charting.LightningChartUltimate
.SetDeploymentKey(deploymentKey);

// Setting of deployment key to other Arction components
SignalGenerator.SetDeploymentKey(deploymentKey);
AudioInput.SetDeploymentKey(deploymentKey);
AudioOutput.SetDeploymentKey(deploymentKey);
SpectrumCalculator.SetDeploymentKey(deploymentKey);
SignalReader.SetDeploymentKey(deploymentKey);

    }
}
}

```

**NOTE!** Without setting Deployment Key in the application, it enters into 30 days trial mode in the target machine (applies computers where a Development license key hasn't been installed).

### 3.5 Running with Deployment Key on development computer

If you are running an application, in which a deployment key has been applied with *SetDeploymentKey*, on a computer where a development license has been installed to, the library **prioritizes the development license key**. It might lead into user or debugging confusion when deployment key has higher level of features included (e.g. Gold pack) than locally installed license (e.g. Silver pack). As a developer, you must be aware of this limitation.

*Arction recommends all licenses to be of same type, within the whole team.*

### 3.6 Running with debugger

With Deployment Key set correctly, when running the project from Visual Studio with debugger attached, and no development license key is found from the system, the chart enters into slow rendering mode, max FPS is ~1, and the chart shows message text over the chart.

*Direct developing and debugging with LightningChart without developer license key, is forbidden by LightningChart EULA.*

### 3.7 Trial period

The trial period is usable for 30 days, and after that, you must purchase a license to continue using the product. All projects built with a trial license will work also after updating to proper license. You will be shown a trial version nag message when running the chart application built with a trial license.

### 3.8 Floating licenses

Floating licenses can be installed to unlimited count of computers. Number of concurrent developers has been configured by Arction. Only the purchased count of concurrent users can develop with LightningChart at same time. After ending LightningChart development by a developer, there's about 10-15 minutes timeout until another developer can start using it.

Deployment key has to be set similarly than with per-developer licenses.

Floating licenses are controlled by Arction Licensing Server by default. Continuous internet connection is required while developing.

*Customer-side floating license controller* is also available. Development computers connect to a service running in customer's organization via local area network. On-line communication with Arction or other parties doesn't take place. With licenses, Arction provides separate instructions for installing the controller service and floating licenses.

## 4. LightningChartUltimate component

### 4.1 Adding from toolbox into Windows Forms project

Add **LightningChartUltimate** control from your toolbox to the form. The chart appears into the form and properties are shown in **Properties window**.

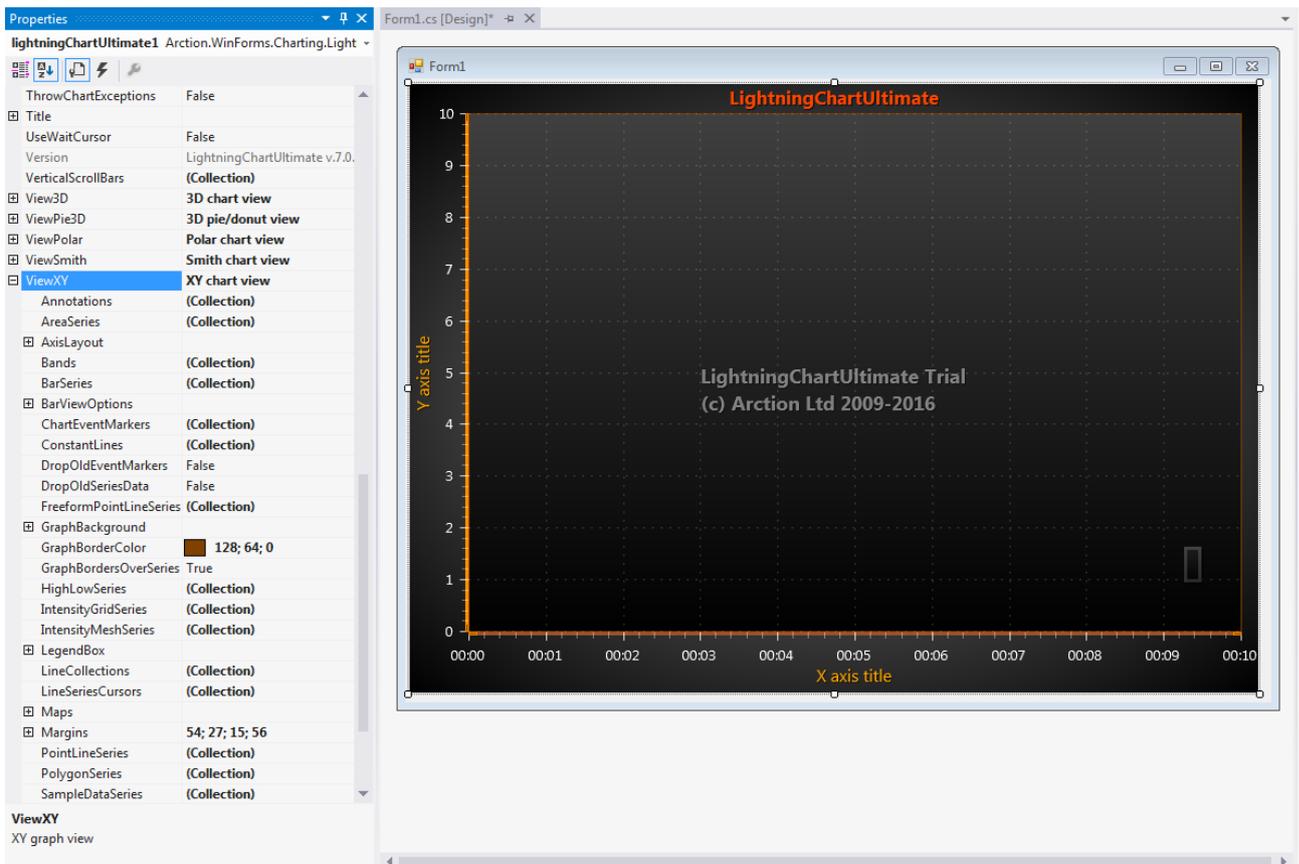


Figure 4-1. LightningChartUltimate control added into Windows Forms designer.

#### 4.1.1 Properties

You can modify the properties freely, and insert new series and other objects in their collections. Series data points must be given by code.

### 4.1.2 Event handlers

Event handlers of the chart main level can be assigned with the property grid. For objects that you have added to the collections, events handlers must be assigned in code.

### 4.1.3 Best practices, version updates in mind

Chart properties data is serialized in **.resx** file in your Visual studio project. **LightningChartUltimate** API tends to change a little bit with version updates, and that may lead into incompatible serialization for the new version to exist in the **.resx** file.

When learning **LightningChartUltimate** API, using property grid is handy.

For easiest updates, it's strongly recommended to create the chart object, add all series, event handlers etc. in code. The project then loads correctly and you get possible errors in the compile time and fixing them is easy when compared to fixing **.resx** file. With **.resx** file, you may lose the some property definitions but in code, they are always specified.

## 4.2 Adding from toolbox into WPF project

Add **LightningChartUltimate** Semi-bindable or Bindable WPF control from your toolbox to your Window or other container. The chart appears into the designer and properties are shown in **Properties window**. XAML editor shows the contents and modifications to the chart default properties.

### 4.2.1 Properties

You can modify the properties freely, and insert new series and other objects in their collections.

- In Semi-bindable chart, series data points must be given by code
- In Bindable chart, series data points can be given by binding data to the series, or in code behind

## 4.2.2 Event handlers

Event handlers of the chart main level can be assigned with the property grid. For objects that you have added to the collections, events handlers must be assigned in code.

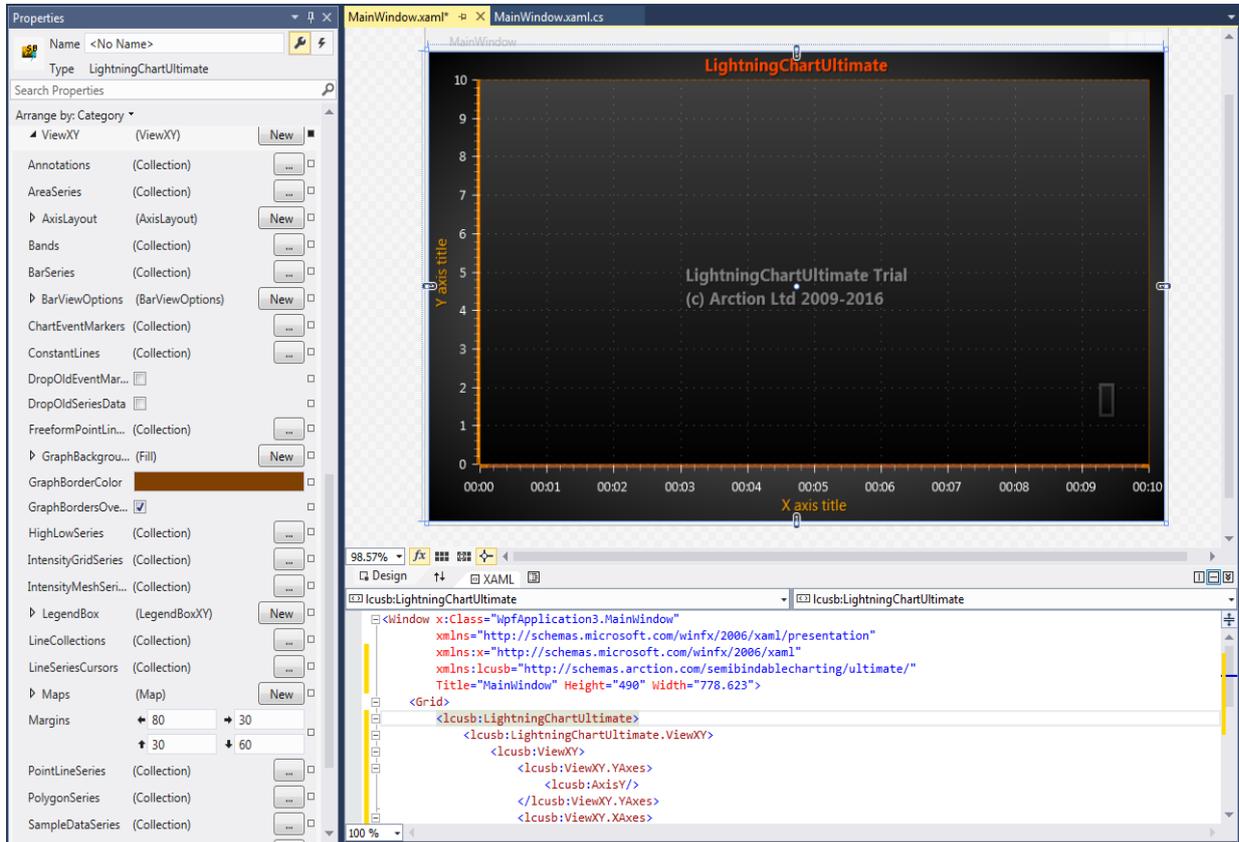


Figure 4-2. LightningChartUltimate control added into WPF designer.

## 4.3 Adding into Blend WPF project

In **Projects** tab, go to **References**. Right-click and select **Add reference...** Browse Arction.WPF.LightningChartUltimate.dll from `c:\program files (x86)\Arction\LightningChart Ultimate SDK v.8\LibNet4`.

Go to **Assets** tab. Write "Lightning" in the Search box. It lists **LightningChartUltimate** row in the search results. Drag-drop the object into your WPF window.

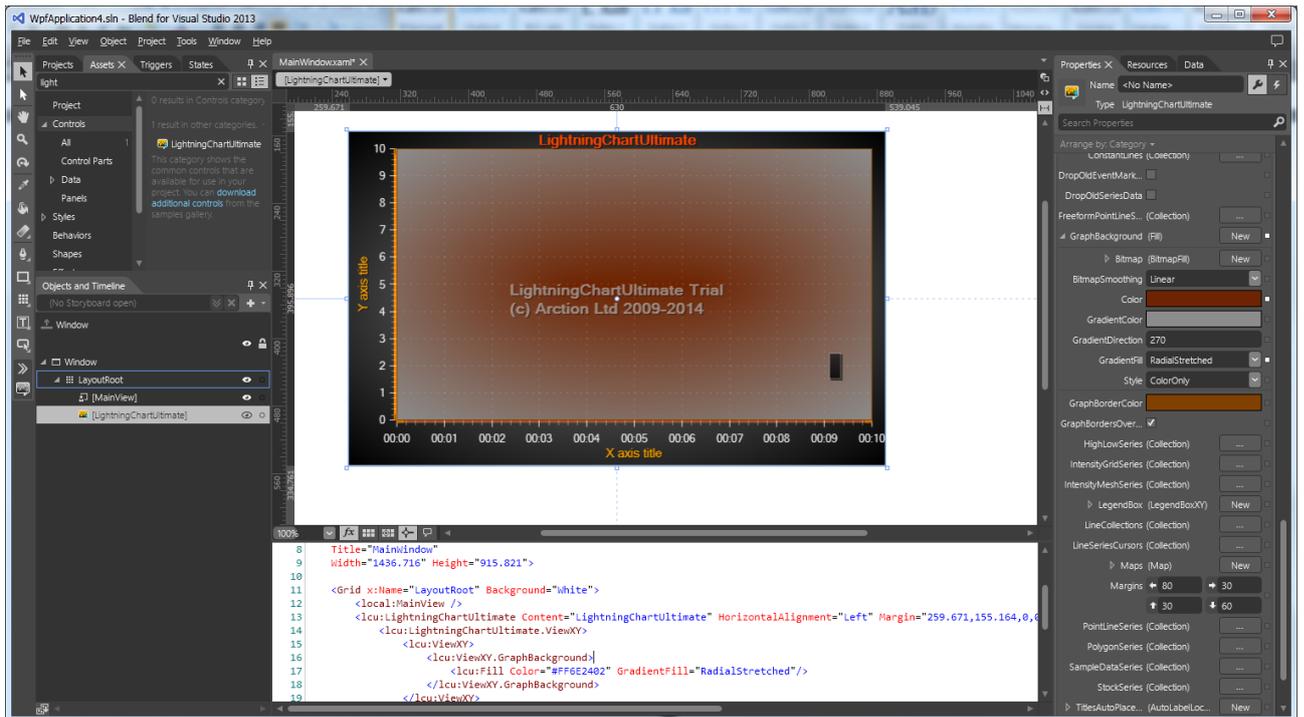


Figure 4-3. LightningChartUltimate control added into Blend For Visual Studio 2013 designer.

### 4.3.1 Best practices, version updates in mind

Chart properties data is stored in XAML. New version may have slightly different property set, and may cause the LightningChartUltimate object not to appear in the designer. Relevant XAML modifications are then needed. The XAML tags tree may be huge and editing it may be quite difficult.

For easiest updates, it's strongly recommended to create the chart object and set its layout and alignment relevant properties in designer. Set everything else in code. Or create the chart object in code too.

### 4.3.2 Preventing blurring of the chart

This is common feature of WPF and not related to the chart itself, but clearly becomes visible in accurate rendering of LightningChart.

To prevent the chart to appear blurred, set **UseLayoutRounding = True** of the control that is **parent** to the chart. It may still appear blurred in the designer, but will look sharp when you run the application. The parent control may be for example **Grid**, **Canvas**, **DockManager** etc.

## 4.4 Object model

The object model of LightningChart can be learned best by using *Properties editor* of Visual Studio.

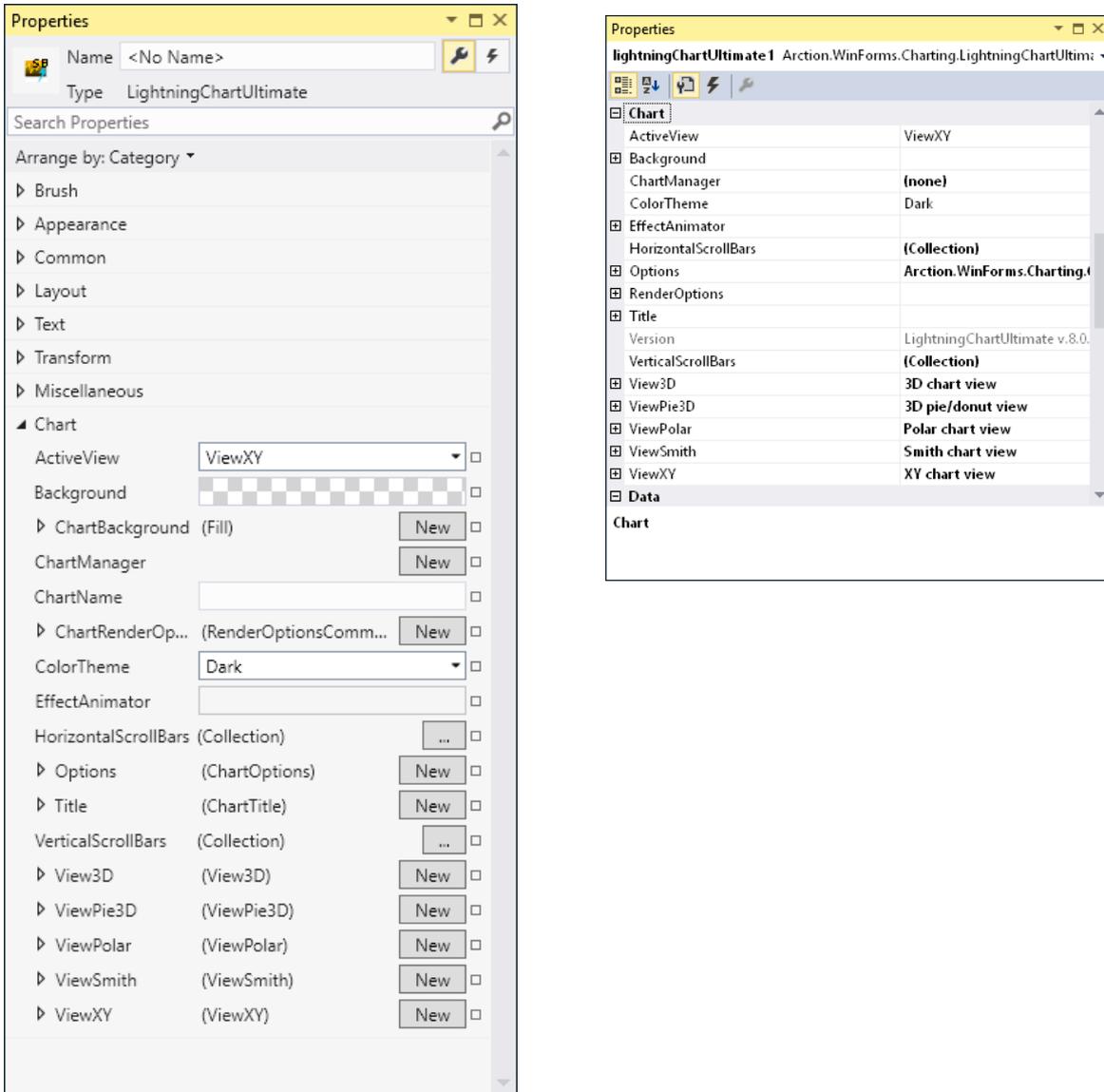


Figure 4-4. LightningChart specific properties can be found under *Chart* category in both Windows Forms and WPF Properties window. By expanding the nodes, or in WPF creating new objects, you will notice a huge set of properties exist in overall.

#### 4.4.1 Differences between Windows forms and WPF

The property tree and object model between Windows Forms and WPF are almost identical, regarding the Chart category. The main differences are:

	Windows Forms	WPF
<b>Rendering options property</b>	RenderOptions	ChartRenderOptions
<b>Background fill property</b>	Background	ChartBackground
<b>Fonts</b>	System.Drawing.Font	Arction.WPF.LightningChartUltimate.WPFFont
<b>Colors</b>	System.Drawing.Color	System.Windows.Media.Color

In the following chapters, Windows forms property names are referred unless otherwise denoted.

LightningChart has these main views:

- **ViewXY** (see chapter 5)
- **View3D** (see chapter 6)
- **ViewPie3D** (see chapter 7)
- **ViewPolar** (see chapter 8)
- **ViewSmith** (see chapter 9)

The visible view can be changed by setting **ActiveView** property. Default view is **ViewXY**.

## 4.5 Setting background fill

All views have common background fill.

- Use **chart.Background** in WinForms.
- Use **chart.ChartBackground** in WPF.

The background fill supports

- Solid color fills. Set **GradientFill = Solid** and use **Color** to define the color.
- Gradient fills, going from **Color** to **GradientColor**. Set GradientFill = Linear / Radial / RadialStretched / Cylindrical. Use **GradientDirection** to control the fill direction in Linear and Cylindrical gradients.
- Bitmap fills, with different tiling and stretching options. Bitmap tint and alpha also are supported to, to make translucent bitmap fills.

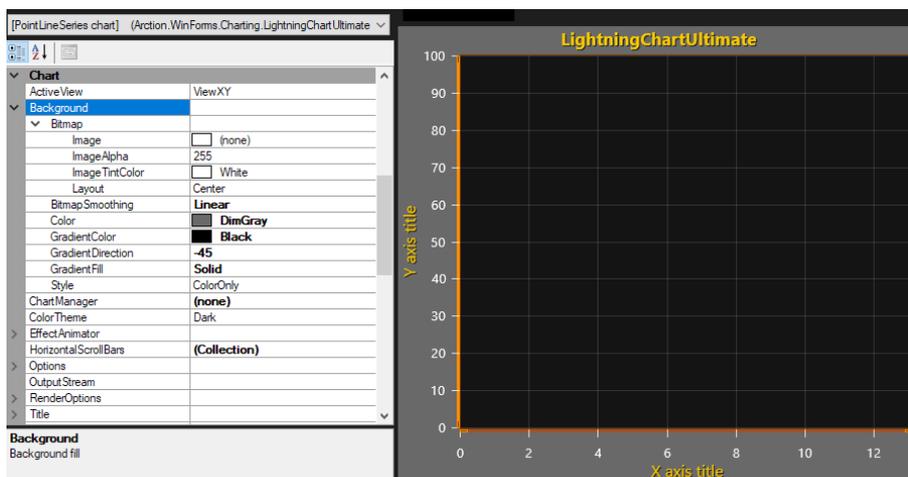


Figure 4-5. Setting Background background, underneath ViewXY. GradientFill = Solid, and Color = DimGray.

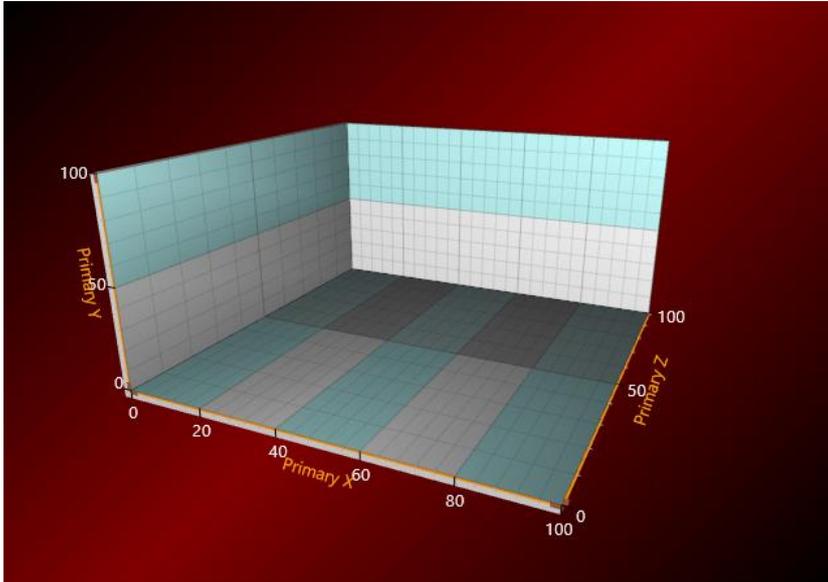


Figure 4-6. Setting Background to gradient cylindrical, under View3D. GradientFill = Cylindrical. Color = Maroon, GradientColor = Black. GradientDirection = -45 degrees.

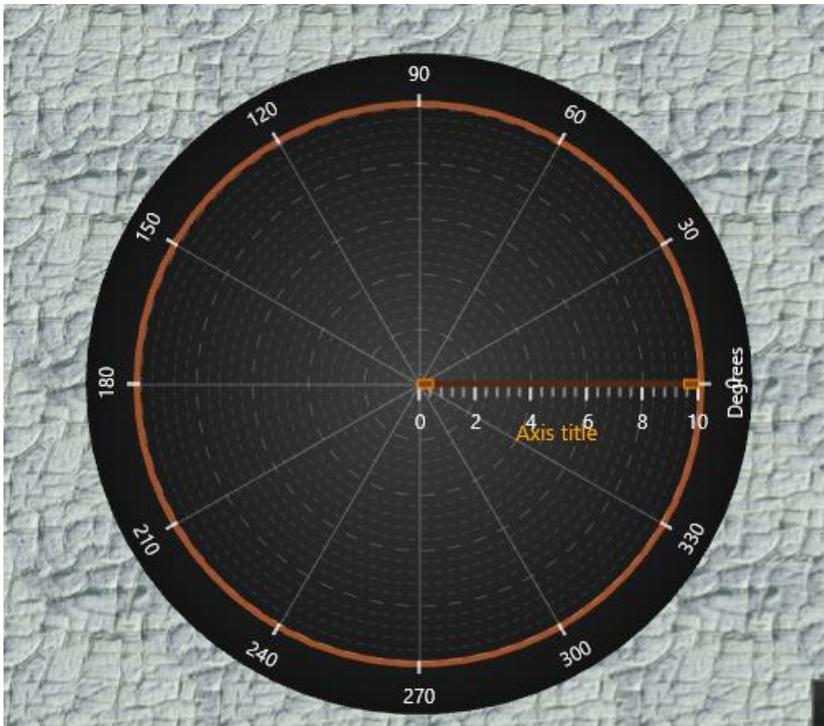


Figure 4-7. Setting Background to tiled bitmap fill. Style = Bitmap, a picture set to Bitmap.Image, and Bitmap.Layout = Tile, under ViewPolar.

### 4.5.1 Setting transparent background

In WPF, you can make the chart appear transparent, so objects you have placed underneath the chart, will show through.

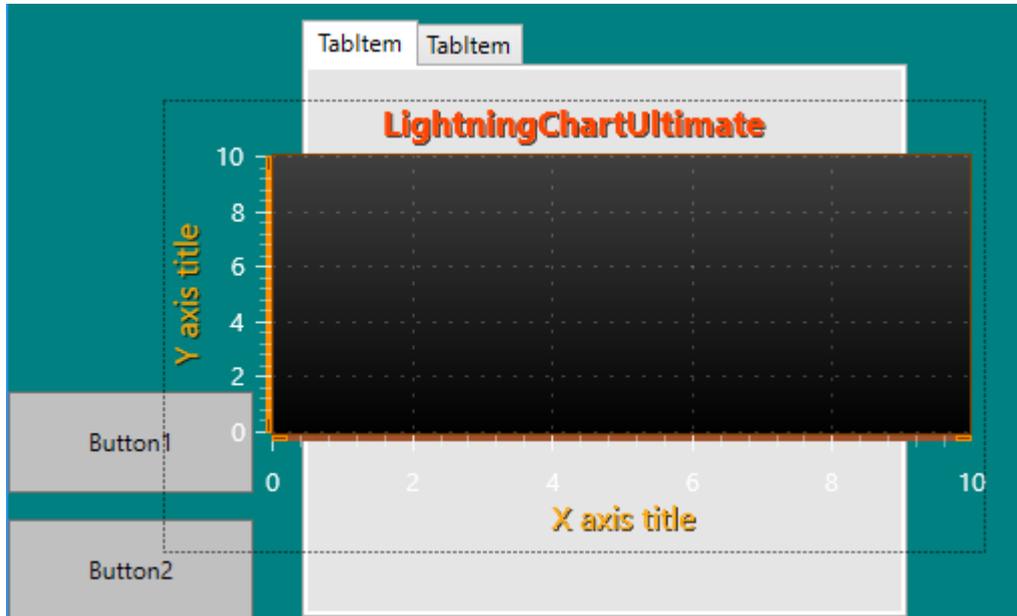


Figure 4-8. Transparent background in WPF chart.

Set `ChartBackground.Color = #00000000` (Black transparent).

**Note!** Do NOT set 'Transparent' (`#00FFFFFF`). It won't show through.

WinForms does not support transparent background of controls.

## 4.6 Configuring appearance / performance settings

**RenderOptions** (*ChartRenderOptions* in WPF) contain properties for configuring appearance and performance.

RenderOptions	
AntiAliasLevel	4
D2DEnabled	True
DeviceType	Auto
FontsQuality	Mid
ForceDeviceCreateOnResize	False
FrameRateLimit	40
GPUPreference	PreferHighPerformanceGraphics
HeadlessMode	False
InvokeRenderingInUIThread	False
LineAAType2D	ALAA
LineAAType3D	QLAA
LineOffset	0.5,0.5
RemoteDesktopVendorId	0
UpdateOnResize	True
UpdateOnResizeTimeInterval	1000
UpdateType	Sync
ViewXY	
GDILineSeriesCompression	True
GDIRendering	False
LineSeriesEnhancedAntiAliasing	Off
WaitForVSync	False

Figure 4-9. RenderOptions properties.

### DeviceType

**Auto** is an alias to AutoPreferD11 option. This is the default setting.

**AutoPreferD9** prefers DirectX9 hardware rendering, and automatically selects device in this order: HW9 -> HW11 -> SW11 -> SW9 based on availability. Falls back to WARP (SW11) software rendering when hardware is not available.

**AutoPreferD11** prefers DirectX11 hardware rendering, and automatically selects device in this order: HW11 -> HW9 -> SW11 -> SW9 based on availability. Falls back to WARP (SW11) software rendering when hardware is not available. **Use this as a general high-performance and best appearance setting.** Visual appearance is better than with DirectX9 renderer.

**HardwareOnlyD9** uses hardware 9 rendering only.

**HardwareOnlyD11** uses hardware 11 rendering only.

**SoftwareOnlyD11** uses DirectX11 WARP, very fast when compared to DirectX9 reference rasterizer, but slower than hardware options)

**SoftwareOnlyD9** uses DirectX9 reference rasterizer (very slow)

**None** if chart is hidden, or inactive in background, setting **DeviceType** to **None** will free graphics resources to other charts.

### **GPUPreference**

Selection applicable to machines with dual graphics adapter systems, mainly laptops having integrated low-performance Graphics Processing Unit (GPU) in the CPU/chipset, and higher performance graphics GPU (e.g. AMD or Nvidia).

**SystemSetting** uses option selected in the graphics settings of Windows, or AMD or Nvidia control panel.

**PreferHighPerformanceGraphics** uses high-performance GPU if it exists in the system. Gives better performance in general, but may lead into higher energy consumption.

**PreferLowPowerGraphics** uses slower integrated GPU, even if high-performance GPU has been installed in the system.

For best performance – select **PreferHighPerformanceGraphics**.

### **FontsQuality**

**Low** gives best performance, the fonts are not anti-aliased, so select font typeface carefully to get acceptable appearance.

**Mid** gives best almost similar performance than **Low**, and has simple anti-aliasing around the fonts. This is the default setting.

**High** gives best appearance but has a significant performance hit.

**NOTE:** Transparent background is not applicable for DirectX 11 rendering with **High** quality setting. For DirectX9 it works. This is a rendering technology limitation.

### **AntiAliasLevel**

The overall scene anti-aliasing factor. Availability depends on the hardware. Higher values give better appearance, but cost of performance. Set 0 or 1 to maximize the performance.

### **WaitForVSync**

**Recommendation: keep as default value.** When enabled, holds rendering until display's next refresh is taking place (e.g. next multiple of 1/60 s). Only recommended temporarily e.g. when

synchronization with external screen capture application is used to prevent striping, or when image on the screen in top of the screen is not in sync with bottom of the screen. It may show as broken waveform data. Significant performance hit when enabled, especially in WPF.

### ***UpdateType***

**Sync (default):** Chart is updated synchronously. Chart gets updated either when last EndUpdate() call or setting a property (or calling a method) causes some changes in the Chart. Property change (without BeginUpdate...EndUpdate) leads to immediate new frame rendering.

**Async:** Chart is updated on async fashion. The chart will update as fast as possible after property changes, but chart will render new frame at some later point. This might make it easier to use chart on some cases.

**LimitedFrameRate:** Frame rate is limited to value specified in FrameRateLimit property. 0 = unlimited. E.g. if you want max. 10 refreshes / second, set 10. This is similar to the Async option, but prevents new frames to be rendered right after first one, thus reducing framerate, but sparing system resources.

**NOTE:** Ensure correct thread handling also in LimitedFrameRate and Async modes. If chart updates asynchronously, and you update chart properties at the same time, a conflict may occur and crash the chart or application.

### ***InvokeRenderingInUIThread***

When using a background thread in the application, all UI updates from the thread must go through Invoke (**Control.Invoke()** in WinForms, and **Dispatcher.Invoke()** in WPF).

The rendering part will use internal Invoke to UI thread, when enabled.

The default value is False, as you should also take care of setting properties and calling methods in thread-safe way, even when this property is enabled, to prevent thread collision in internal states of the chart.

### ***HeadlessMode***

Setting it to True, allows using the chart in a background service, console application or other application without user interface. See chapter 21.

## 5. ViewXY

**ViewXY** allows presenting various point-line series, area series, high-low series, intensity series, heat maps, bar series, bands, line series cursors etc. in Cartesian, XY graph format. Series are bound to X and Y axes, and they are using the value range of the assigned axes.

**ViewXY** also shows the geographical maps, see section 5.24.

ViewXY	XY chart view
Annotations	(Collection)
AreaSeries	(Collection)
AutoSpaceLegendBoxes	False
> AxisLayout	
Bands	(Collection)
BarSeries	(Collection)
> BarViewOptions	
ChartEventMarkers	(Collection)
ConstantLines	(Collection)
DropOldEventMarkers	False
DropOldSeriesData	False
FreeformPointLineSeries	(Collection)
> GraphBackground	
GraphBorderColor	 128; 64; 0
GraphBordersOverSeries	True
HighLowSeries	(Collection)
IntensityGridSeries	(Collection)
IntensityMeshSeries	(Collection)
LegendBoxes	(Collection)
LineCollections	(Collection)
LineSeriesCursors	(Collection)
> Maps	
> Margins	61; 28; 12; 58
PointLineSeries	(Collection)
PolygonSeries	(Collection)
SampleDataSeries	(Collection)
StockSeries	(Collection)
> TitlesAutoPlacement	
XAxes	(Collection)
YAxes	(Collection)
> ZoomPanOptions	

Figure 5-1. ViewXY object tree.

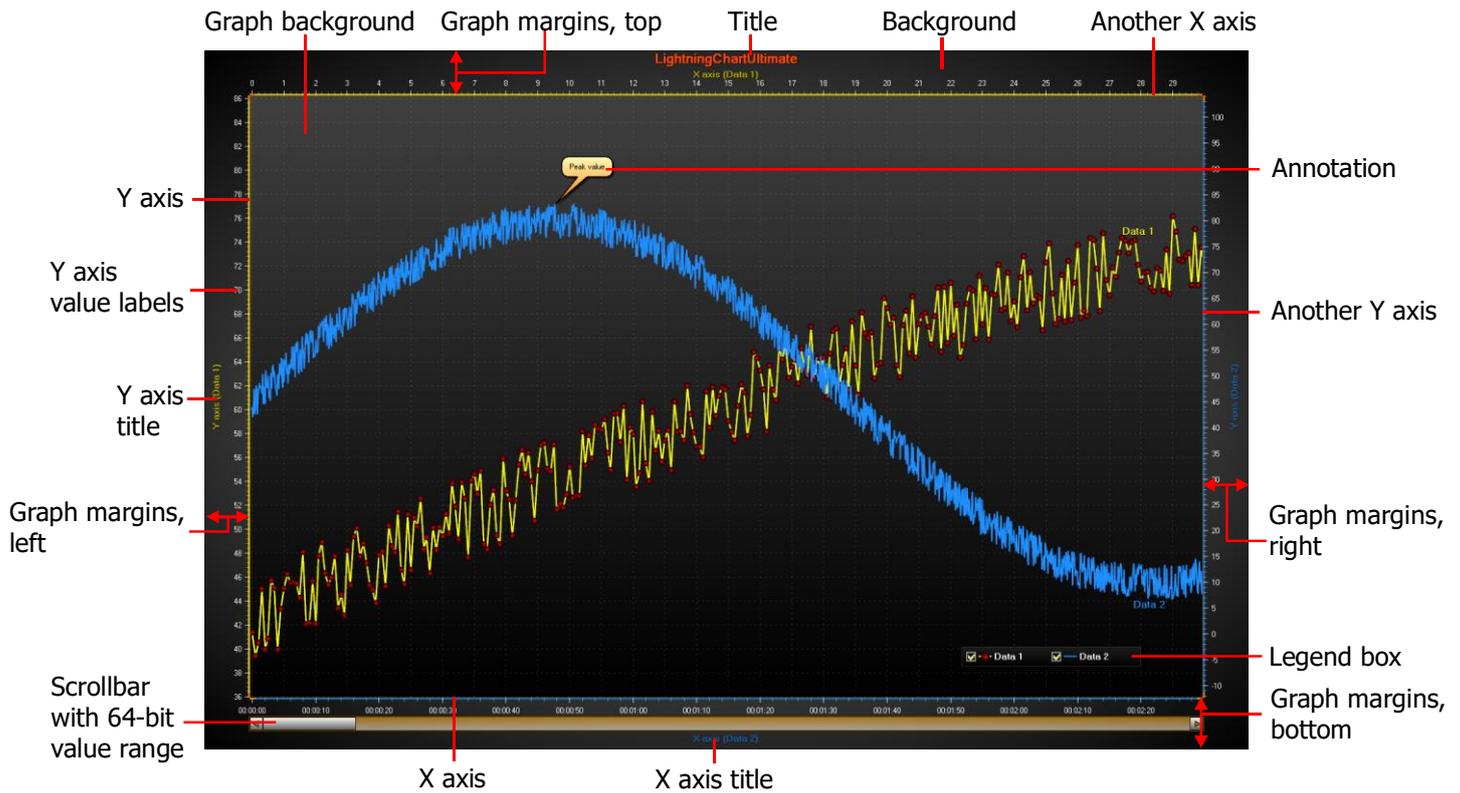


Figure 5-2. Quick overview of ViewXY parts

### Graph margins

Margins are adjusted automatically by default, by axis count and their settings. By setting ***ViewXY.AxisLayout.AutoAdjustMargins = False***, ***Margins*** property applies. If you want the graph to fill whole control area, set all margins to 0.

### Graph border

The border is drawn around the graph area, in the location of margins. The color can be set with ***GraphBorderColor*** property.

### Background

Set the background fill with ***Background (ChartBackground in WPF)*** property. There are plenty of filling options available. See 4.5.

### Graph background

Set the graph background fill with ***GraphBackground*** property. Graph is the area where all grids, series, series cursors, event markers etc. are rendered.

### Title

This is the main title for the chart. Set the text, shadow, color, text border, rotation, font, alignment etc. with ***Title.Text***, ***Title.Shadow...*** properties.

### Y axes

See section 5.2.

### X axis

See section 5.3.

### Annotations

See section 5.19.

### Legend box

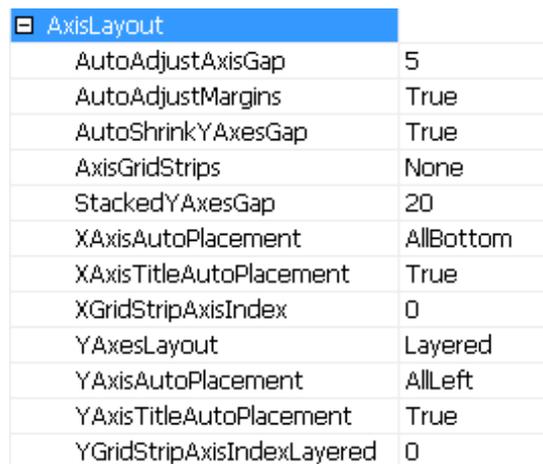
See chapter 5.20.

### Scrollbar

A scrollbar having unsigned 64-bit value range, to support massive count of sample indices directly. In fact, **HorizontalScrollBars** and **VerticalScrollBars** are collection properties in the chart root level, but they are aware of ViewXY's margins. See chapter 10.

## 5.1 Axis layout options

The general properties adjusting axis placement, automatic margins etc. can be found in **ViewXY.AxisLayout** properties and sub-properties.



AxisLayout	
AutoAdjustAxisGap	5
AutoAdjustMargins	True
AutoShrinkYAxesGap	True
AxisGridStrips	None
StackedYAxesGap	20
XAxisAutoPlacement	AllBottom
XAxisTitleAutoPlacement	True
XGridStripAxisIndex	0
YAxesLayout	Layered
YAxisAutoPlacement	AllLeft
YAxisTitleAutoPlacement	True
YGridStripAxisIndexLayered	0

Figure 5-3. AxisLayout property tree.

### 5.1.1 Automatic margins

When **AutoAdjustMargins** is **enabled**, the graph size is adjusted so that there's enough space for all the axes and chart title. When that's **disabled**, **ViewXY.Margins** property applies and allows setting margins manually.

In the run time, you can retrieve the margins rectangle in pixels by calling **ViewXY.GetMarginsRect** method. That applies both automatic and manual margins, and is useful when you need to do screen-coordinate based computation or object placement.

### 5.1.2 Setting how axes are placed

#### 5.1.2.1 X axis automatic placement

**XAxisAutoPlacement** controls how the X axes are placed vertically.



Figure 5-4. **XAxisAutoPlacement = AllBottom**. Three X axes added, all are positioned below the graph.

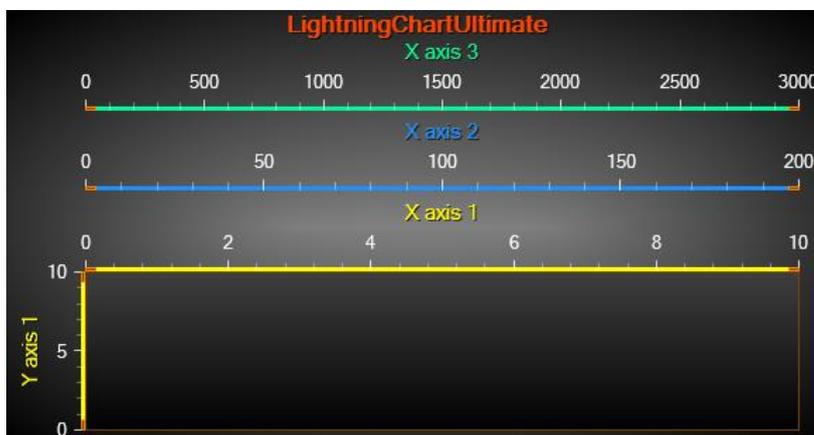


Figure 5-5. **XAxisAutoPlacement = AllTop**. All X axes are positioned above the graph.

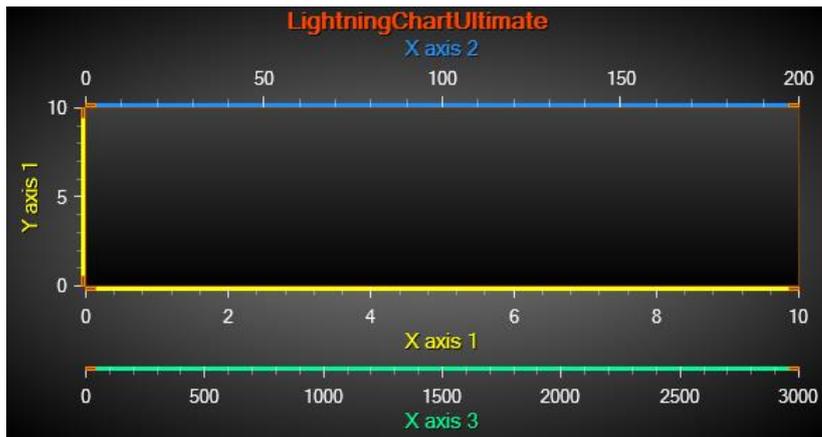


Figure 5-6. XAxisAutoPlacement = BottomThenTop. Axes are distributed below and above the graph, every other axis to the opposite side, starting from bottom.

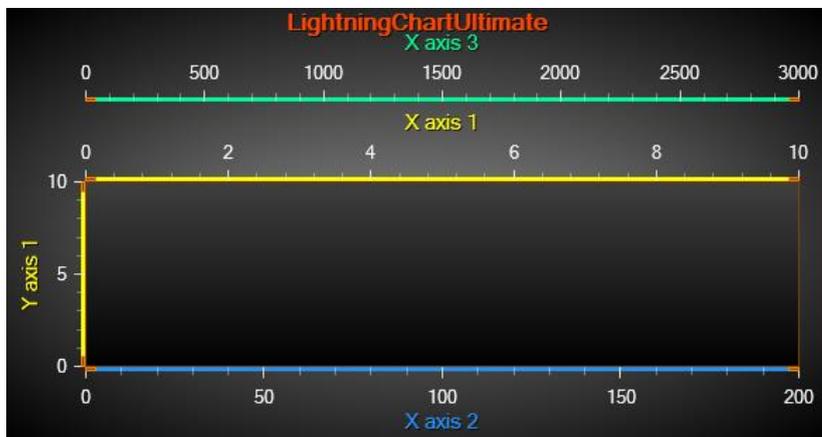


Figure 5-7. XAxisAutoPlacement = TopThenBottom. Axes are distributed below and above the graph, every other axis to the opposite side, starting from top.

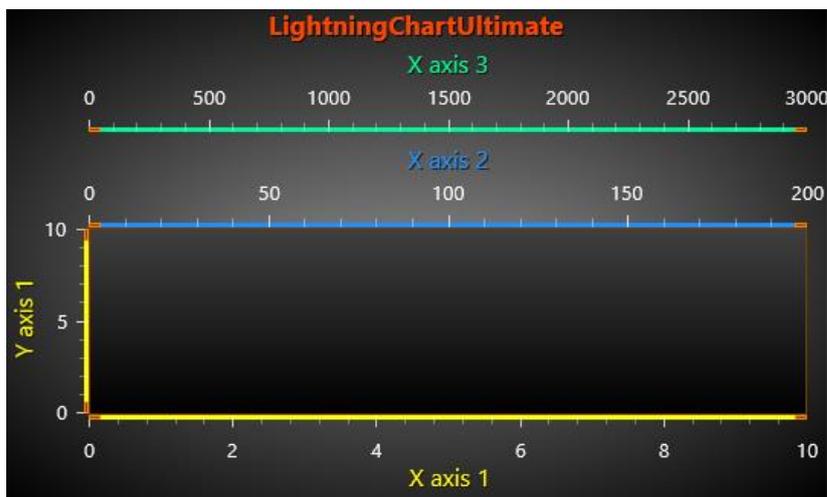


Figure 5-8. XAxisAutoPlacement = Explicit. The axis appears on the side you select explicitly. XAxis1 has ExplicitAutoPlacementSide property set to Bottom, whereas XAxis2 and XAxis3 to Top.

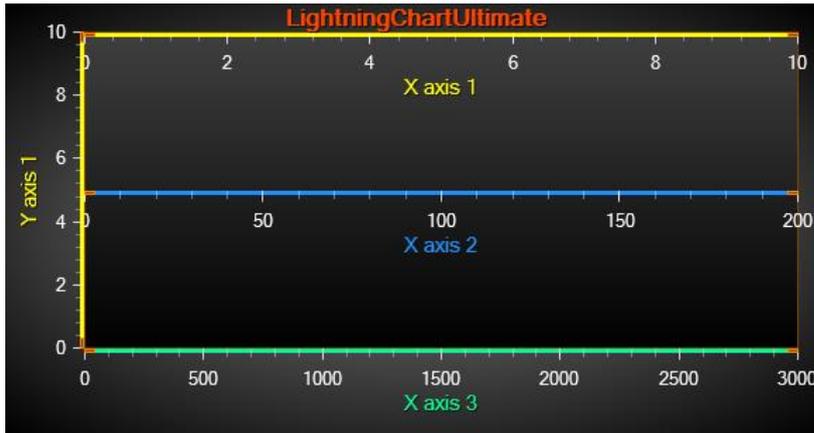


Figure 5-9. XAxisAutoPlacement = Off. Automatic axis placement is disabled, and Position and alignment properties of each axis applies separately. First axis Position = 0, Second axis Position = 50 and Third axis position = 100.

### 5.1.2.2 Y axis automatic placement

**YAxisAutoPlacement** controls how the Y axes are placed vertically.

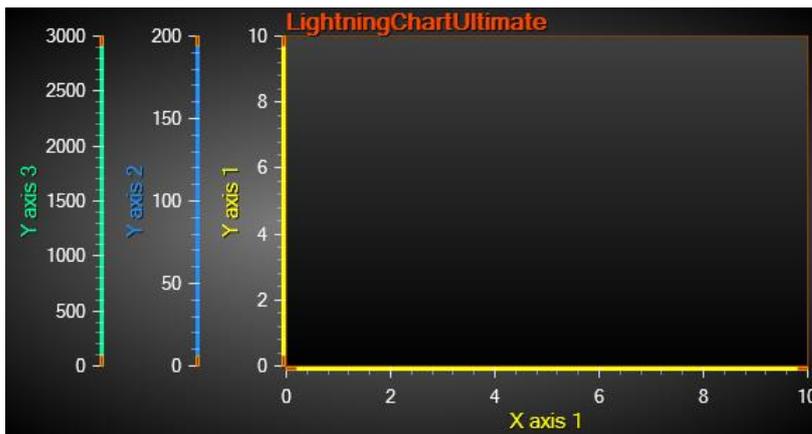


Figure 5-10. YAxisAutoPlacement = AllLeft. Three Y axes added, all are positioned to the left side of the graph.

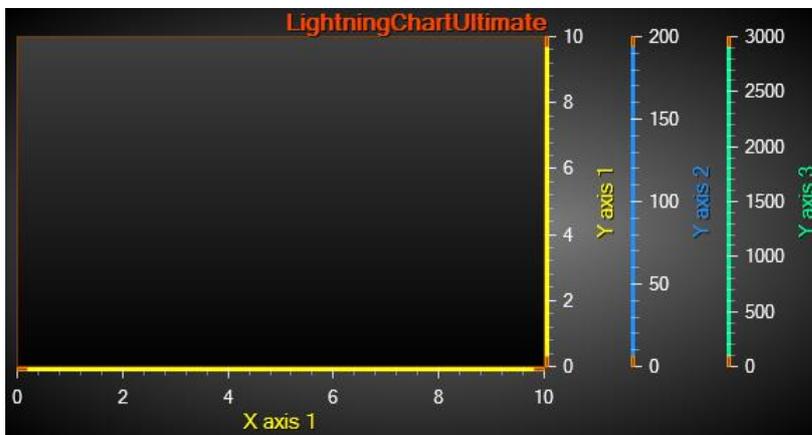


Figure 5-11. YAxisAutoPlacement = AllRight. All Y axes are positioned to the right side of the graph.

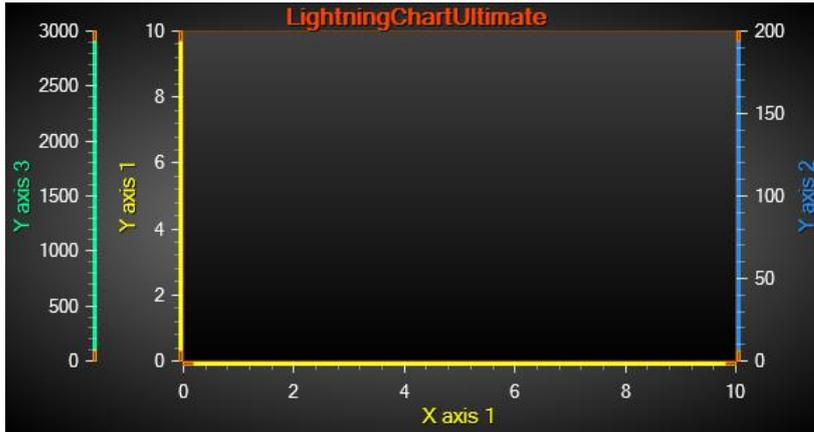


Figure 5-12. YAxisAutoPlacement = LeftThenRight. Axes are distributed to left and right side of the graph every other axis to the opposite side, starting from left side.

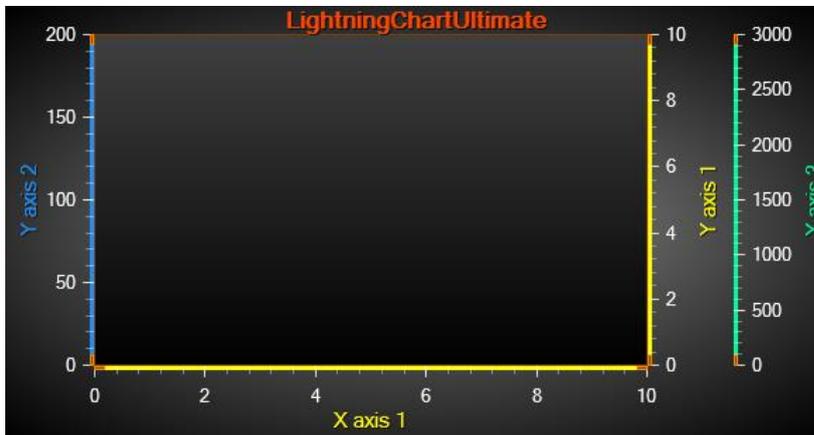


Figure 5-13. YAxisAutoPlacement = RightThenLeft. Axes are distributed to left and right side of the graph, every other axis to the opposite side, starting from right side.

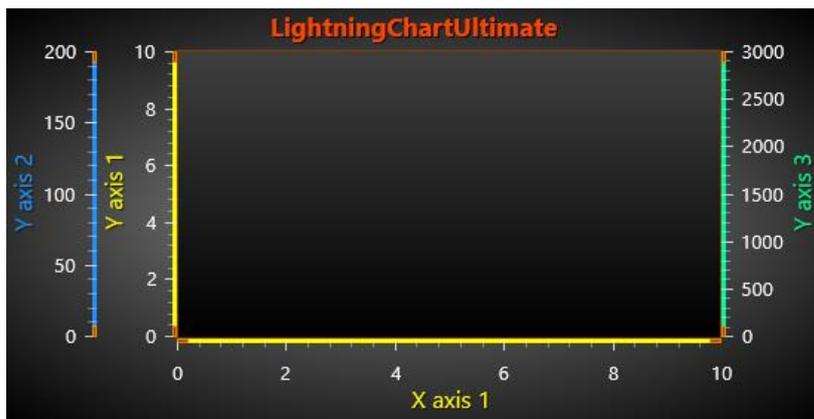


Figure 5-14. YAxisAutoPlacement = Explicit. The axis appears on the side you select explicitly. YAxis1 and YAxis2 have ExplicitAutoPlacementSide property set to Left, and YAxis3 to Right.

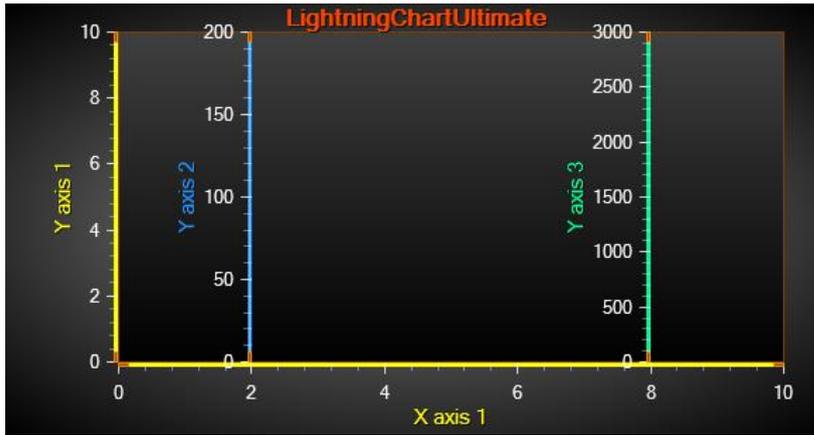


Figure 5-15. YAxisAutoPlacement = Off. Automatic axis placement is disabled, and Position and alignment properties of each axis applies separately. First axis Position = 0, Second axis Position = 20 and Third axis position = 80.

### 5.1.3 Graph segments and Y axes placement in them

If there's several Y axes defined, they can be vertically aligned in 3 different ways: **Layered**, **Stacked** and **Segmented**. This can be selected by **ViewXY.AxisLayout.YAxesLayout** property.

#### 5.1.3.1 Layered

In **Layered** view, all the Y axes start from top of the graph, and stretch to bottom of the graph. The axes and the series bound to them share the same vertical space.

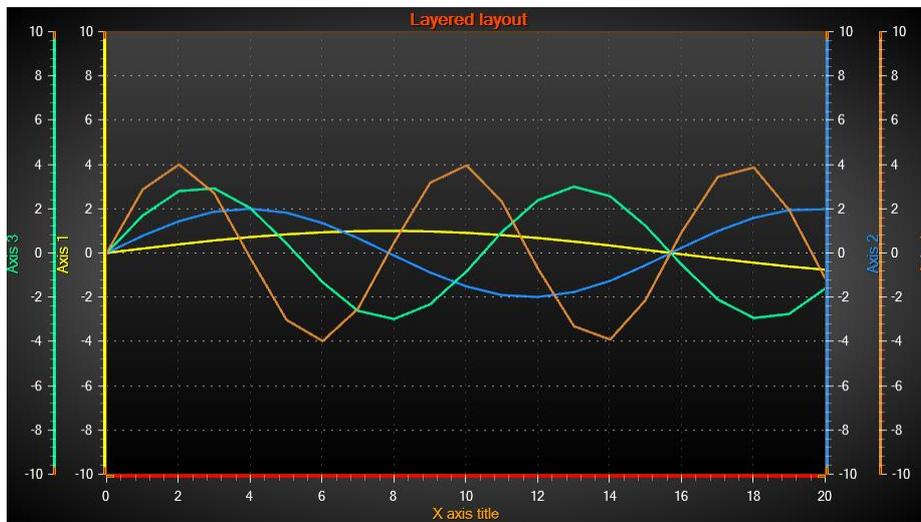


Figure 5-16. An example view of 4 Y axes in YAxesLayout = Layered.

#### 5.1.3.2 Stacked

In **Stacked** view each Y axis get an own vertical space. All Y axes have equal height.

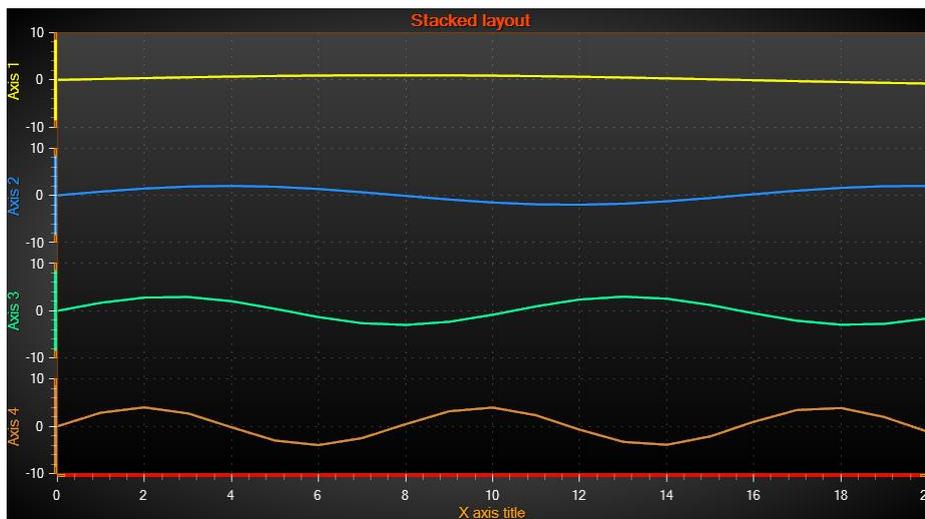


Figure 5-17. An example view of 4 Y axes in YAxesLayout = Stacked.

### 5.1.3.3 Segmented

In Segmented view the vertical space is divided between **Segments**. Each segment can contain several Y axes. Each segment relational height can be set, and all the Y axes within a segment get the segment's height.

The **Segments** must be created in **AxisLayout.Segments** collection. A segment has only one property, **Height**. It is a relational size versus other segments. It is not defined in screen pixels, as they need to rescale with the chart's size.

The Y axes can be assigned with a segment by setting **yAxis.SegmentIndex** property. The **SegmentIndex** the index in the **AxisLayout.Segments** collection.

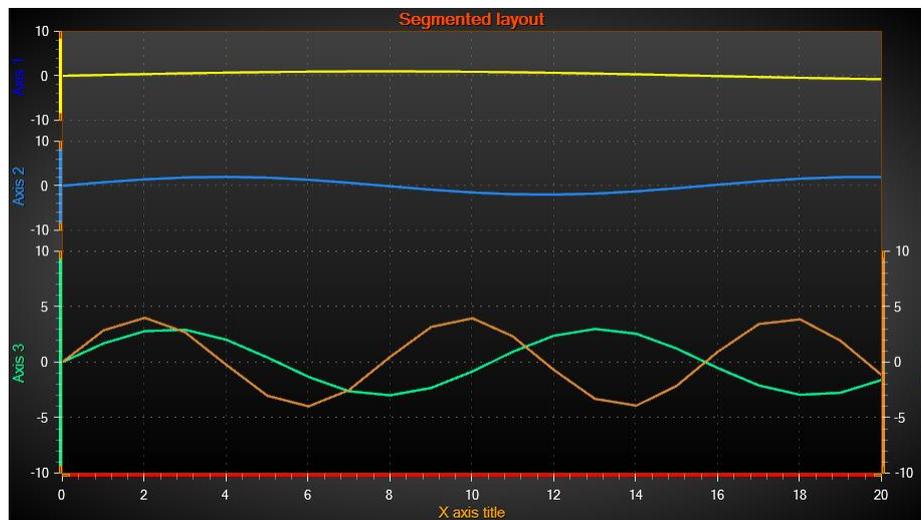


Figure 5-18. An example view of 4 Y axes in **YAxesLayout = Segmented**. First two segments have **Height = 1**, and last segment has **Height of 2.5**. **Axis1.SegmentIndex = 0**, **Axis2.SegmentIndex = 1**, **Axis3 and Axis4.SegmentIndex = 3**.

When a **Stacked** or **Segmented** view is selected, you can adjust the vertical space between graph segments by using **ViewXY.AxisLayout.SegmentsGap** property. If there's a myriad of Y axes defined, you should enable **ViewXY.AxisLayout.AutoShrinkSegmentsGap** property, for automatically decrease the gaps. By doing so, every Y axis gets at least some vertical space to be drawn. Use **ViewXY.GetGraphSegmentInfo()** method to find out where the graph segment borders are, if you need to have segment specific user interface logic

### 5.1.4 Axis grid strips

The axis grid (division) intervals can be shown over the graph background, as fills. By setting **ViewXY.AxisLayout.AxisGridStrips = X**, an X axis is used to set the strips. By setting **AxisGridStrips = Y**, Y axis is used to set the strips. **AxisGridStrips = Both** sets the strips by both X and Y axis. Setting it to **None**, no grid strips are used at all.

**XGridStripAxisIndex** sets the X axis that is to be used for strips. Only one X axis can be set.

**YGridStripAxisIndexLayered** sets the Y axis to be used for strips, when **YAxisLayout = Layered**. When **YAxisLayout = Stacked**, all Y axes have their own strips.

The strip colors can be adjusted in **GridStripColor** property of X or Y axis object.

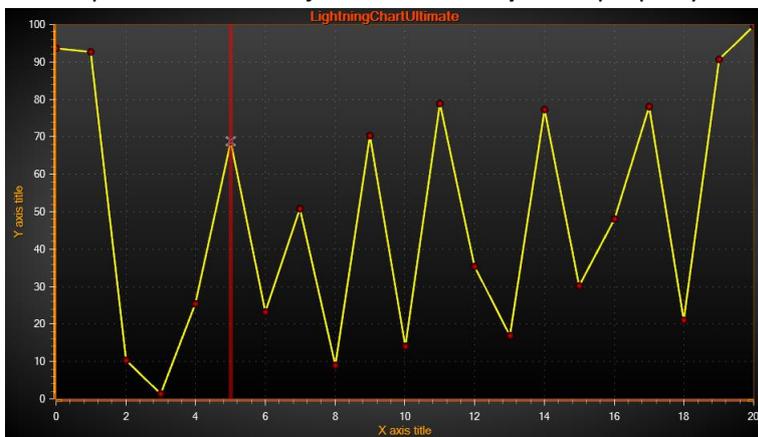


Figure 5-19. AxisGridStrips = None.

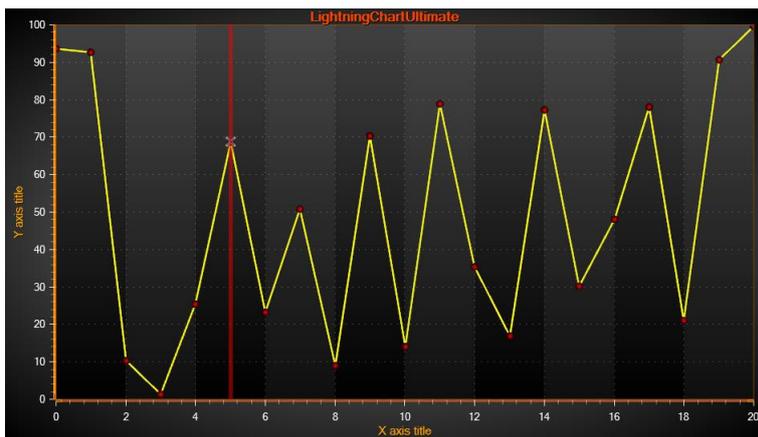


Figure 5-20. AxisGridStrips = X.

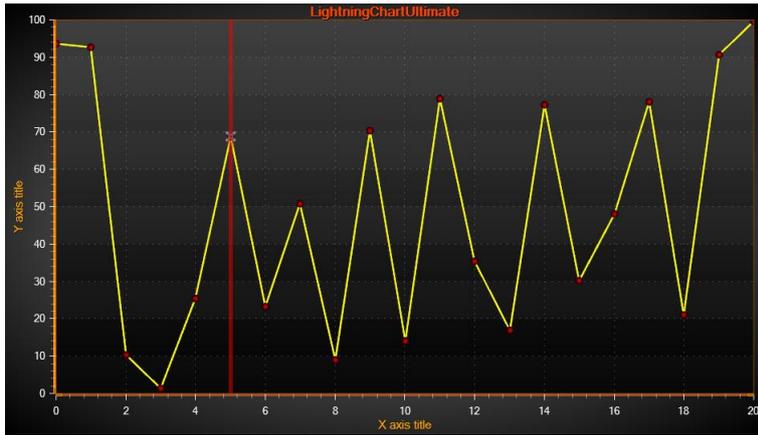


Figure 5-21. AxisGridStrips = Y.

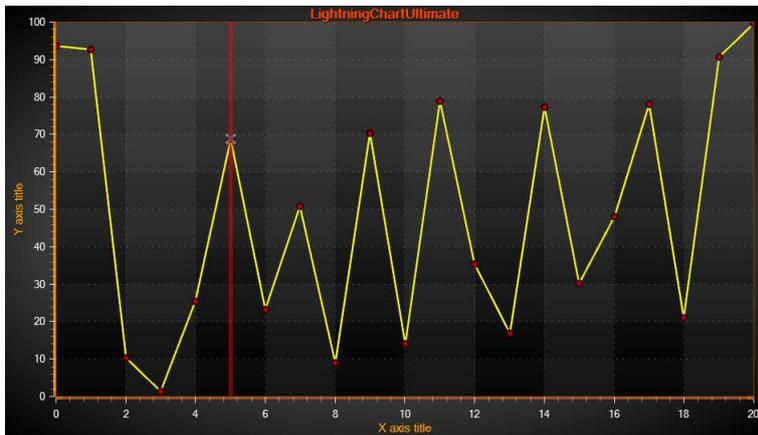


Figure 5-22. AxisGridStrips = Both.

### 5.1.5 Other AxisLayout options

**AutoAdjustAxisGap** sets the space between two adjacent axis areas in pixels, when **XAxisAutoPlacement** or **YAxisAutoPlacement** is enabled.

By enabling **XAxisTitleAutoPlacement** (or **YAxisTitleAutoPlacement**), the axis title distance is automatically calculated based on value labels length, alignment options of axes and tick lines. If **XAxisTitleAutoPlacement** (or **YAxisTitleAutoPlacement**) is disabled, **Title.DistanceToAxis** of axis object property sets the distance to axis line instead.

## 5.2 Y axes

You can define unlimited count of Y axes. Add the Y axes by using **YAxes** collection property.

### 5.2.1 AxisY class properties

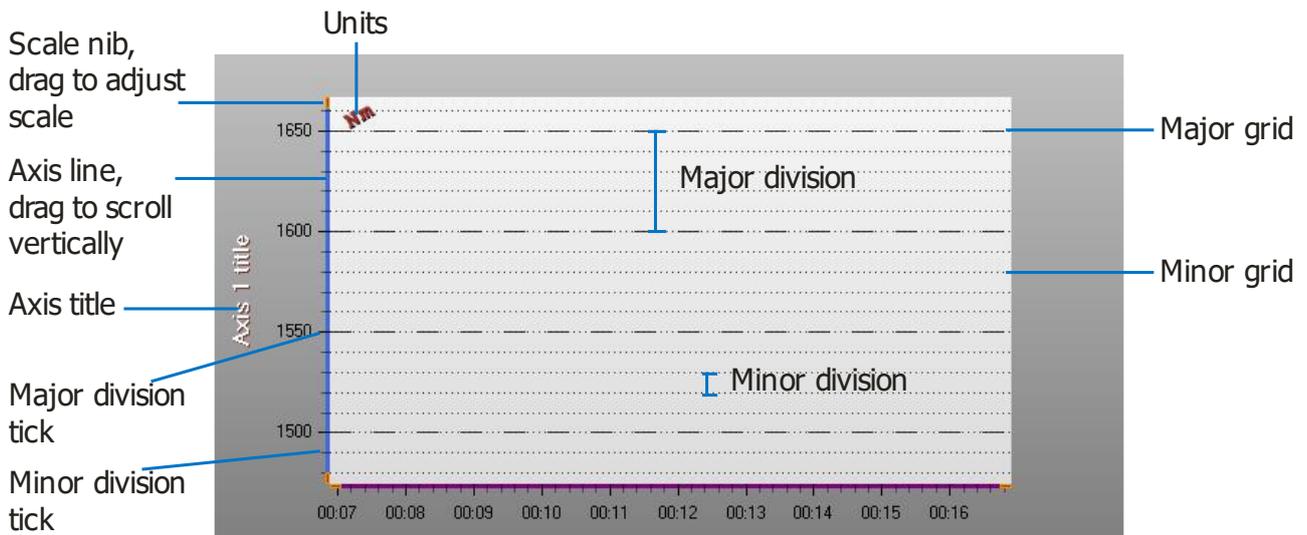


Figure 5-23. Y axis, divisions and grid.

### 5.2.2 Tick value labels formatting

The **AutoFormatLabels** can be left enabled, if you want the count of decimals or time format representation to be calculated automatically suitable for visible range. To set the number formatting manually, disable **AutoFormatLabels** and use **LabelsNumbersFormat** property. Normal `Double.ToString()` method parameter string applies. To set the time formatting manually, disable **AutoFormatLabels** and use **LabelsTimeFormat** property. `DateTime.ToString()` method parameters string applies, but there is also an improvement. `DateTime.ToString()` accepts only three second fractions (.fff) with `DateTime.ToString()`, but **LabelsTimeFormat** supports any count of second fractions (e.g. “.ffffff”) allowing good zoomed views. **LabelsNumberFormat** property is generally used, when **AutoFormatLabels** is enabled, to format the decimal presentation of the numbers.

### 5.2.3 Value type

The **ValueType** property can be set to different settings:

#### **Number**

Regular numeric format, for integer and decimal presentation. When **AutoFormatLabels** is disabled, **LabelsNumbersFormat** applies.

#### **Time**

Time format, for time of day presentation. When **AutoFormatLabels** is disabled, **LabelsTimeFormat** applies.

#### **DateTime**

Date presentation, with optional time of day. When **AutoFormatLabels** is disabled, **LabelsTimeFormat** applies here too, like with **Time** type.

**NOTE!** For best accuracy, it is advised to set **DateOriginYear**, **DateOriginMonth** and **DateOriginDay** just below the dates you will be showing in the chart. Use **DateTimeToAxisValue** method to obtain axis value from a .NET **DateTime** object, to be used in series data.

#### **MapCoordsDegrees**

Geographical map coordinate presentation in degrees decimals.

Example:  $40.446195^{\circ} -79.948862^{\circ}$

#### **MapCoordsDegNESW**

Geographical map coordinate presentation in degrees decimals, with N, E, S, W indication.

Example:  $40.446195N 79.948862W$

#### **MapCoordsDegMinSecNESW**

Geographical map coordinate presentation in degrees, arc minutes, arc seconds, with N, E, S, W indication.

Example:  $40^{\circ}2'13"N 9^{\circ}58'2"W$

#### **MapCoordsDegPadMinSecNESW**

Geographical map coordinate presentation in degrees, arc minutes, arc seconds, with N, E, S, W indication. The arc minute and second values are padded with zeros, if they are < 10. It is great way to present coordinates in Y axis, as the numbers are aligned.

Example:  $40^{\circ}02'13"N 9^{\circ}58'02"W$

## 5.2.4 Range setting

Set the value range by giving values to **Minimum** and **Maximum** properties. **Minimum** must be less than **Maximum**. If you try to set **Minimum** > **Maximum**, or vice versa, internal limiter will limit the values near the other value. To set both values simultaneously, use **SetRange(...)** method. If you pass minimum > maximum in SetRange, it automatically flips this values so that minimum < maximum.

The value range of Y axis can be scrolled directly by dragging the axis with mouse, **MouseScrolling** enabled. **Minimum** or **Maximum** can be modified by dragging the scale nib area (end of an axis) up or down, **MouseScaling** enabled.

## 5.2.5 Restoring range

Axis has properties **RangeRevertEnabled**, **RangeRevertMaximum** and **RangeRevertMinimum**. They can be used to revert axis ranges to specific values when mouse zooming is applied from right to left. See 5.21.5 for details.

## 5.2.6 Divisions

The major divisions spacing can be done automatically by leaving the **AutoDivSpacing** enabled. The spacing is calculated as user-friendly as possible, depending on labels font size and **AutoDivSeparationPercent** properties. Increase the value to obtain less major divisions. Minor divisions are calculated between major divisions by using **MinorDivCount** property value.

By setting **AutoDivSpacing** disabled, you can control the division spacing manually with **MajorDiv** property, if you want to control spacing by magnitude. Or use **MajorDivCount** property to control spacing by division count. **KeepDivCountOnRangeChange** property can be used to force maintaining the divisions count same when range is changed, since **MajorDivCount** and **MajorDiv** depend on each other.

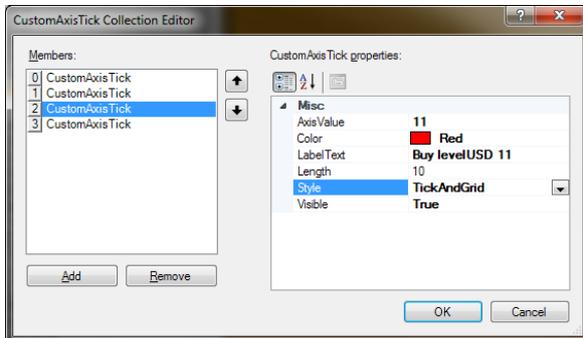
Major division tick style can be set from **MajorDivTickStyle** property. Edit the ticks and labels orientation by using **MajorDivTickStyle.Alignment** property. The value labels are drawn next to major division ticks. Edit the minor division properties with **MinorDivTickStyle** property, respectively.

## 5.2.7 Grid

Horizontal grid lines are drawn on vertical positions of division ticks. Major grid for major division ticks, minor grid for minor division ticks. Use **MajorGrid** and **MinorGrid** properties to edit the appearance.

## 5.2.8 Custom ticks

You can manually set the axis tick positions. Define the positions in the **CustomTicks** list property, and set **CustomTicksEnabled** true.



**CustomAxisTicks** class represents one tick, grid or both. Use **Style** to select between Grid, Tick or TickAndGrid respectively. Set the color of tick or grid in **Color** property. Set tick length in **Length** property.

The grid line pattern follows the setting of **MajorGrid.Pattern** and **PatternScale** properties of axis.

CustomAxisTick has the **AxisValue** and **LabelText** property. When using custom ticks, disable **AutoFormatLabels** to show the custom label texts.

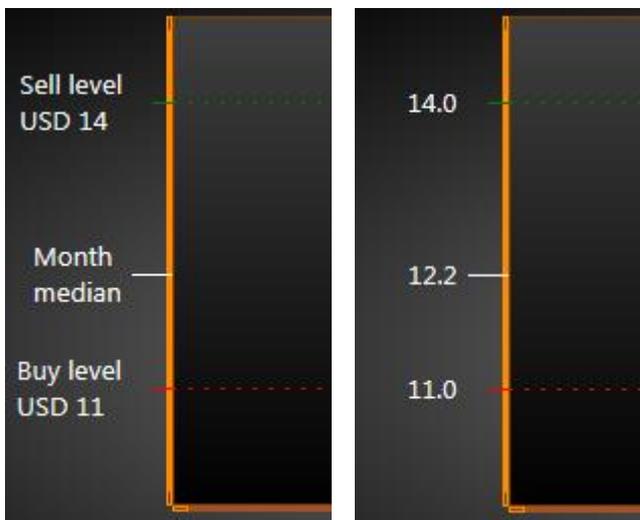


Figure 5-24. Custom ticks on Y axis. On the Left, axis.AutoFormatLabels = false. On the right, AutoFormatLabels = True.

Remember to call **InvalidateCustomTicks()** after setting new ones in code.

Minor ticks or grid are not shown when **CustomAxisTicksEnabled** is true. So set arbitrary minor ticks or grids, just add **CustomAxisTicks** in the **CustomTicks** collection with different colors or line lengths.

## 5.2.9 Reversed X and Y axis

X and Y axis can be shown reversed, so that minimum value is above/after than the maximum value. Handy feature when you want to visually negate polarity of the series data assigned to the Y axis, for example.

## 5.2.10 Logarithmic axes

Set **ScaleType** to **Logarithmic** to use a logarithmic presentation. Set the logarithm base value with **LogBase** property. The chart can show also logarithmic values between 0...1. Use **LogZeroClamp** to set the minimum value in the axis. To use typical minimum value of a log axis, set 1. To use values below zero, set a proper small positive value, like 1.0E-20, suitable for your data. To use special formatting for tick labels, set **LogLabelsType**.

### 5.2.10.1 Exponential presentation for 10 base

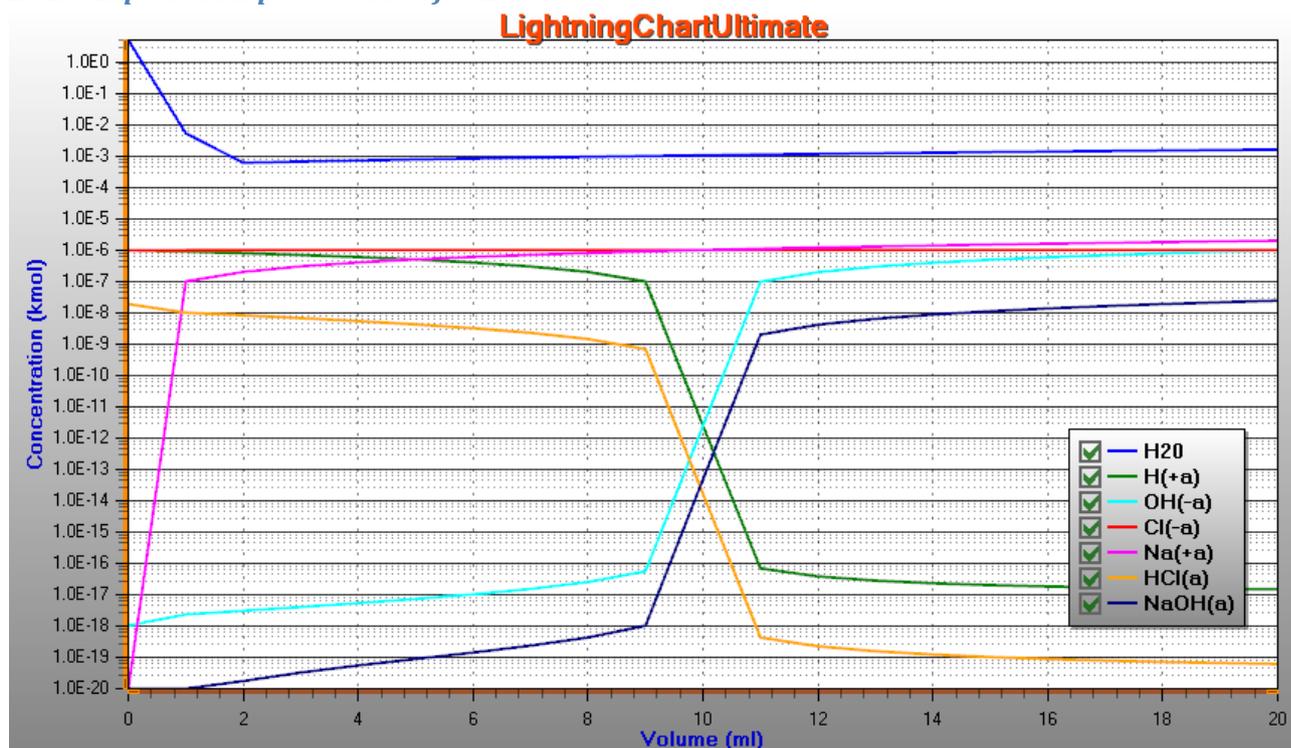


Figure 5-25. Logarithmic Y axis with values near zero. LogZeroClamp is set to 1.0E-20. LogBase is set to 10, LogLabelsType is set to Log10Exponential, to show the values in 1.0E presentation.

### 5.2.10.2 Natural logarithm

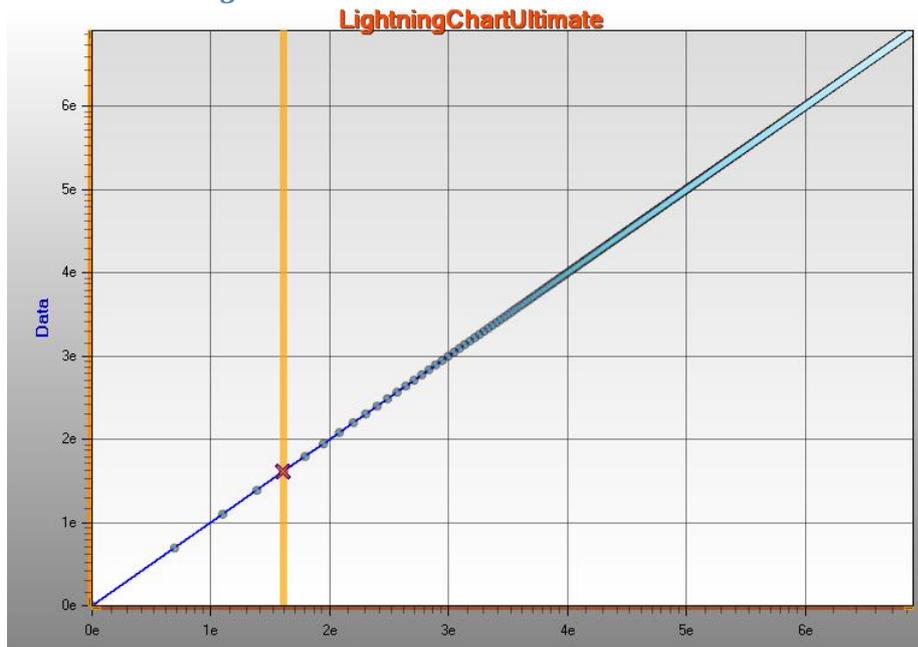


Figure 5-26. Natural logarithm view. LogBase is set to Math.E LogLabelsType is set to LogE\_MultiplesOfNeper.

### 5.2.11 Converting between axis values and screen coordinates

Axes have methods to convert axis values (data points values) to screen coordinates and screen coordinates to axis values. Use **ValueToCoord** method to convert axis value to screen coordinate, and **CoordToValue** to convert screen coordinate to axis value. Pass **UseDIP** = False, if you prefer pixels, not Device independent pixels (DIPs).

## 5.2.12 MiniScale

### Mini scale

A miniature X and Y axis substitute. In some applications this kind of scale presentation is preferred for quick visual overview of data magnitude or there's no space for actual axes. Mini scales are not visible by default. **MiniScale** is a sub-property of Y axis class. When logarithmic axes are used, mini scales can't be used. The X dimension of a **MiniScale** is always bound to first X axis (**XAxes[0]**). Set the visible units by modifying **Units.Text** property of X and Y axis.



Figure 5-27. MiniScale in the bottom-right corner of the graph.

## 5.3 X axis

X axis divisions and grid settings are equal to Y axes settings. The axis can be scrolled by mouse and the minimum and maximum value can be set by dragging the scale nibs.

### 5.3.1 Real-time monitoring scrolling

When making a real-time monitoring solution, the X axis must be scrolled suitable for the current monitoring position, which usually is the time stamp of latest signal point. Set the latest time stamp to **ScrollPosition** property, after you have set the new signal points to series. LightningChart has several scrolling modes, selected by **ScrollMode** property.

#### 5.3.1.1 None

No scrolling is applied when setting a value to **ScrollPosition**. This is probably the selection you want to use when using the chart in other applications than real-time monitoring.

### 5.3.1.2 Stepping

X axis is collected full and then the x axis with all series data is shifted left by Stepping interval. Then X axis is collected full again etc. **SteppingInterval** property is defined as value range.

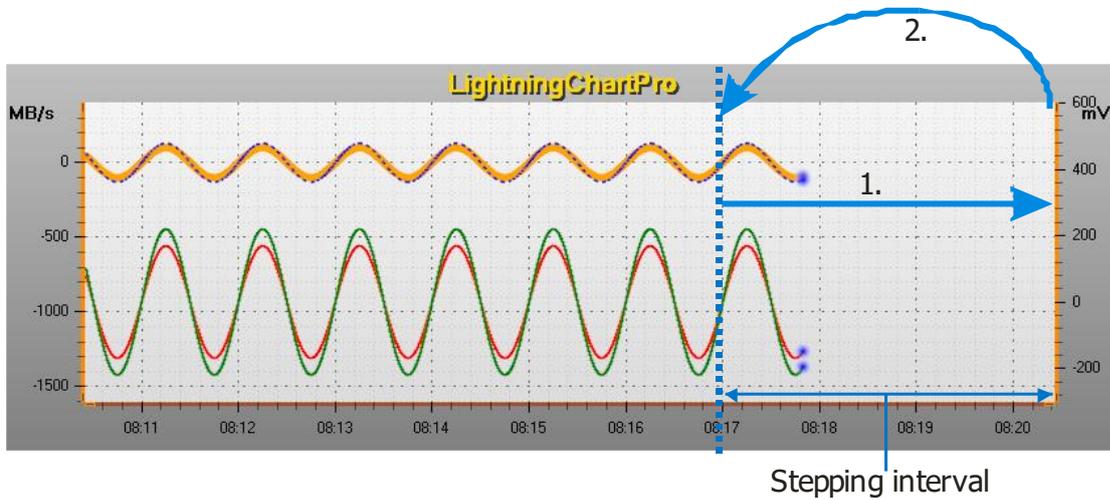


Figure 5-28. X axis scroll mode: stepping

### 5.3.1.3 Scrolling

The X axis is kept stationary until scrolling gap has been reached. Then the X axis with all series is shifted left. If you want the scrolling to take effect when the scroll position reaches end of X axis, set **ScrollingGap** to 0. **ScrollingGap** property is defined as percents of graph width.

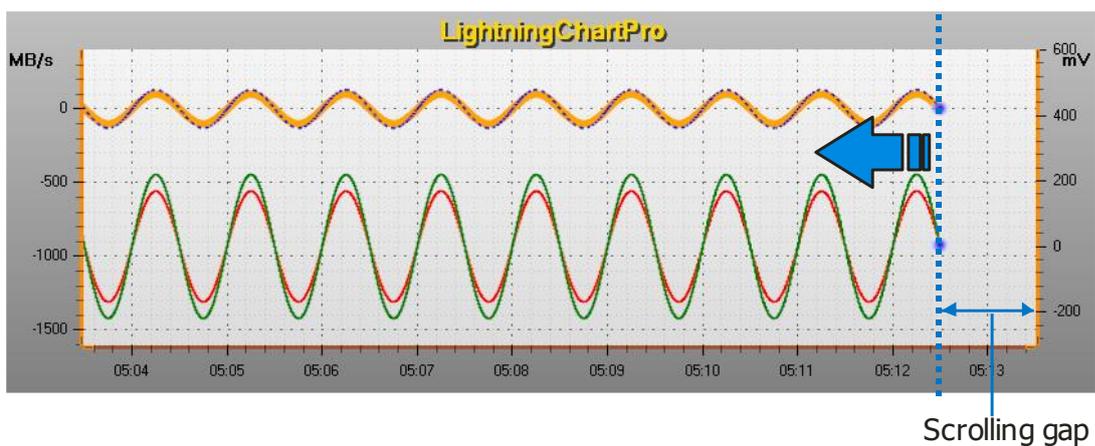


Figure 5-29. X axis scroll mode: scrolling

### Waveform stability during scrolling

LightningChart supports incremental rendering data construction of real-time signal, when using series.**AddPoints()**, or **AddValues()** or **AddSamples()**. In short, it means that rendering data is calculated only from the new part of data and combined with the existing rendering data.

**PointLineSeries**, **SampleDataSeries**, **AreaSeries** and **HighLowSeries** have specific property for **ScrollMode = Scrolling**, which effects the visual stability of scrolled series or maintaining the waveform quality. The property is called **ScrollingStabilizing**.

When it's **enabled**, floating point coordinates are rounded to nearest integer coordinate, which results into a visually stable, non-fluctuating waveform. In most cases, this is the best approach. It may however distort the phase info slightly, when it rounds the coordinates.

When **ScrollingStabilizing** is disabled, the rendering data uses floating point coordinates which appear as slightly fluctuating waveform when GPU decides the pixel coordinate. This gives better visual quality especially when displaying sine data where there's a transition going up and down nearly every other pixel.

To use incremental rendering data construction, add new points as follows

```
chart.BeginUpdate();
series.AddPoints(array, false);
xAxis.ScrollPosition = latestXValue;
chart.EndUpdate();
```

Full refresh of rendering data can be made any time with **InvalidateData()** call of series.

```
chart.BeginUpdate();
series.AddPoints(array, false);
series.InvalidateData();
xAxis.ScrollPosition = latestXValue;
chart.EndUpdate();
```

	Performance	Stability	Phase
<b>series.AddPoints()</b> , <b>ScrollStabilizing</b> <b>disabled</b>	Perfect	Impaired	Good
<b>series.AddPoints()</b> , <b>ScrollStabilizing</b> <b>enabled</b>	Perfect	Best	Slightly impaired
<b>series.AddPoints()</b> , <b>InvalidateData()</b>	Impaired	Impaired	Perfect

### 5.3.1.4 Sweeping

Sweeping mode gives probably the most user-friendly real-time monitoring view. Sweeping uses two X axes. First one is collected full and then a sweeping gap appears with second X axis. The second X axis is swept over the first one. Both X axes show their own value labels. **SweepingGap** property is defined as percents of graph width.

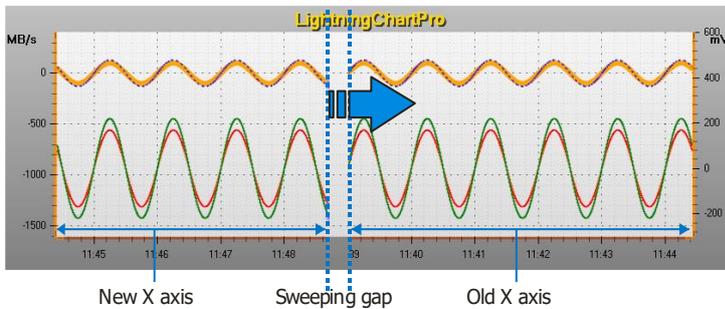


Figure 5-30. X axis scroll mode: sweeping

### 5.3.1.5 Triggering

The X axis position is determined by a series value exceeding or falling below a trigger level. Set the triggering options in **Triggering** property. You must set one series as a triggering series. Accepted triggering series types are **PointLineSeries** and **SampleDataSeries**. Set the triggering Y level with **Triggering.TriggerLevel**. Use **Triggering.TriggeringXPosition** to order where the level triggered point is to be drawn horizontally, as percents of graph width. Remember to set triggering active by enabling **Triggering.TriggeringActive** property.



Figure 5-31. X axis scroll mode: Triggered with static X grid.

When using a triggered X-axis scroll position, usually it is not suitable to show the regular X axis with values and grid, because they jump from place to another according to the incoming series data.

- Approach 1: Use static X grid. Hide the regular X axis objects by setting ***XAxis.Visible = false*** (or ***LabelsVisible = false***, ***MajorGrid.Visible = false*** and ***MinorGrid.Visible = false***). Then, show the static X grid by setting ***Triggering.StaticMajorXGridOptions*** and ***Triggering.StaticMinorXGridOptions***.
- Approach 2: Create another X axis, with preferred scale, and set it to ViewXY collection. Don't assign the second XAxis for the series.

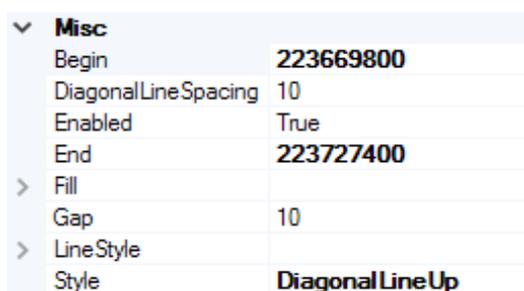
For scale indication, you can use Y axis **MiniScale** or define an **Annotation** (see section 5.19) object to show range like "200 ms/div".

### 5.3.2 Scale breaks

Starting from version 8, X axes support *Scale breaks*. Scale breaks allow excluding specific X ranges, e.g. inactive trading hours/dates or machinery off-production hours. All the series that have been assigned to the specified X axis, are clipped, including axis and labels themselves.

There are limitations of when ScaleBreaks can be used: **ScrollMode** must be set to **'None'** and **ScaleType** to **'Linear'**.

Insert the **ScaleBreak** objects in **ScaleBreaks** collection of X axis.



▼ <b>Misc</b>	
Begin	223669800
DiagonalLineSpacing	10
Enabled	True
End	223727400
> Fill	
Gap	10
> LineStyle	
Style	DiagonalLineUp

Figure 5-32. ScaleBreak properties.

Specify the range of the break with **Begin** and **End**. They are given as axis values, not DateTimes. Use **axis.DateTimeToAxisValue** method to convert them if using DateTimes.

Gap width can be adjusted with **Gap**, also 0 is accepted if no gap should be visible. Gap appearance can be configured with **Style**.

- With **Style = 'Fill'**, adjust the fill with **Fill** property.
- With **Style = 'DiagonalLineUp'** or **'DiagonalLineDown'**, adjust the appearance with **DiagonalLineSpacing** and **LineStyle** properties.

By setting **Enabled = False**, the break is not effective.

**PointLineSeries**, **AreaSeries** and **HighLowSeries** have **ContinuousOverScaleBreak** property. By enabling it, a connecting line will be rendered over the gap.



Figure 5-33. Original trading data, Monday to Friday, 10 AM – 6 PM. Scale breaks haven't been applied. Majority of the time range doesn't have data as stock exchange has been closed. Hard to see the essential info. PointLineSeries jumping from Close-to-Close values.

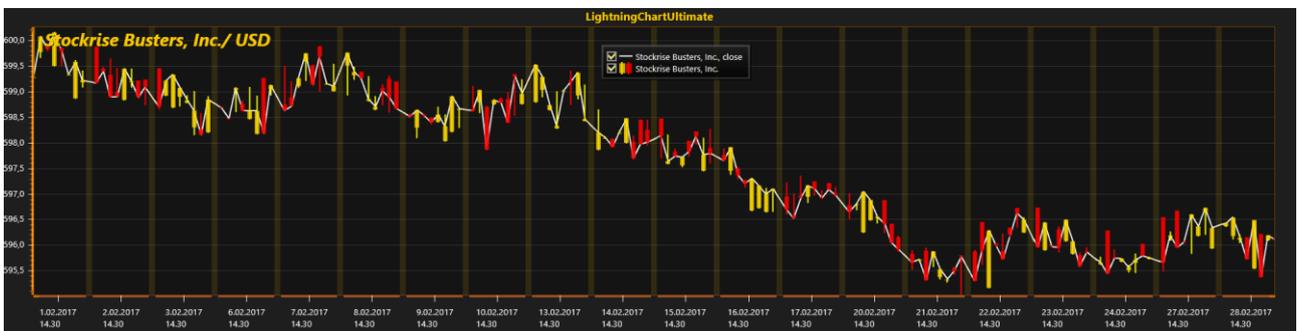


Figure 5-34. Scale breaks applied, during non-active trading hours. More screen space is available for essential data. Style = Fill, Gap = 10. PointLineSeries jumping from Close-to-Close values, PointLineSeries.ContinuousOverScaleBreak = True.



Figure 5-35. Scale breaks applied, during non-active trading hours. Style = DiagonalLinesUp, Gap = 20. PointLineSeries.ContinuousOverScaleBreak = True.

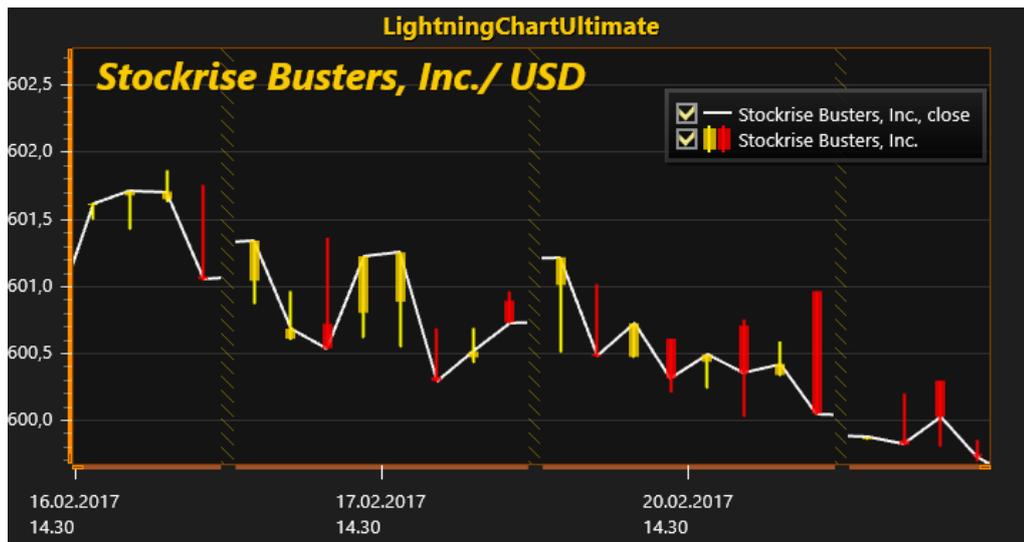


Figure 5-36. `PointLineSeries.ContinuousOverScaleBreak = False`. The lines are not connected from previous point to next point over the gap. Instead, they continue to their original direction if no scale break would have been defined.

## 5.4 ViewXY series, general

ViewXY's series allow data visualization in different ways and formats. All series are bound to axis value ranges. The series must be bound to one Y axis. Series have `AssignXAxisIndex` and `AssignYAxisIndex` property for assigning the X and Y axis. In code, you can assign the X and Y axis with series constructor parameter, alternatively.

## 5.5 Point line series

### PointLineSeries - for variable interval progressing data

FAST TO RENDER

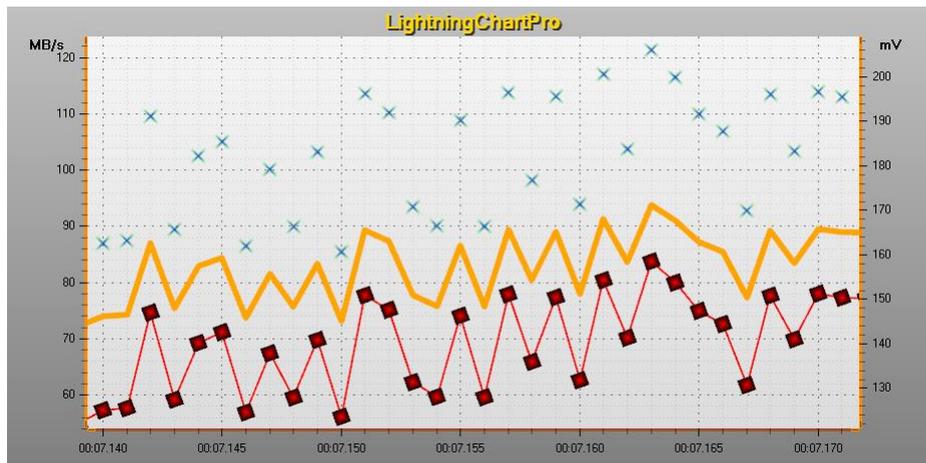
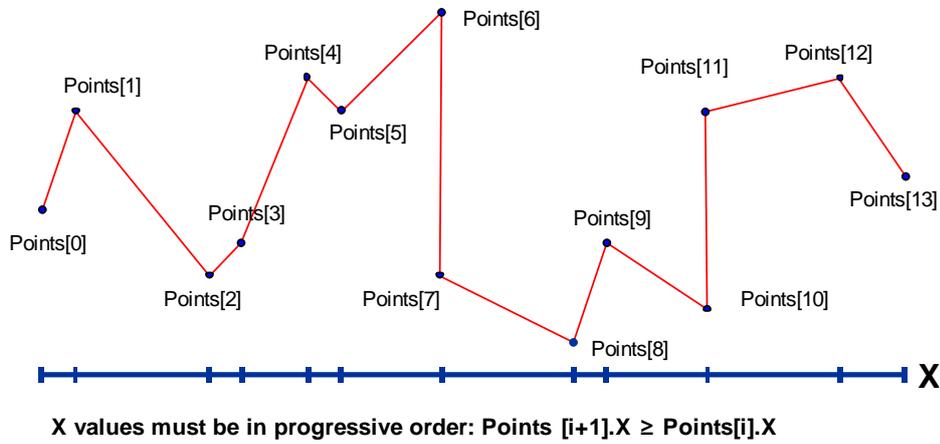


Figure 5-37 Some point line series.

A **PointLineSeries** can present a simple line, points (scatter) or both as a point line. Add the series to chart by adding **PointLineSeries** objects to **PointLineSeries** list.

### 5.5.1 Line style

Define the line style with **LineStyle** property. If you don't want the line to be visible, set **LineVisible** = false.

## 5.5.2 Points style

To show the points, set **PointsVisible** = true. Set the point style by setting **PointStyle** properties. Select the shape from many pre-defined styles from **PointStyle.Shape**. One of the shape styles is **Bitmap**, which allows drawing any bitmap image in the point location. Define the bitmap image with **BitmapImage** property. To set some transparency on the bitmap, use **BitmapAlphaLevel** property. Change the bitmap color tone by defining **BitmapImageTintColor** other color than white. In pre-defined point styles, like **Circle**, **Triangle**, **Cross** etc you can define the drawing colors and filling style. Note that all colors or fills are not applicable for all shape styles. Point width and height can be set and the points can be rotated as well.

## 5.5.3 Coloring points individually

Starting from v.7.2, the **PointLineSeries**, **FreeformPointLineSeries**, **AreaSeries** and **HighLowSeries** have Color field in the data point structures.

To enable individual point coloring, set **IndividualPointColoring** to **Color1**, **Color2**, **Color3** or **BorderColor** setting. To disable individual point coloring, set **IndividualPointColoring** = **Off**. The color settings correspond that color in **PointStyle** property.

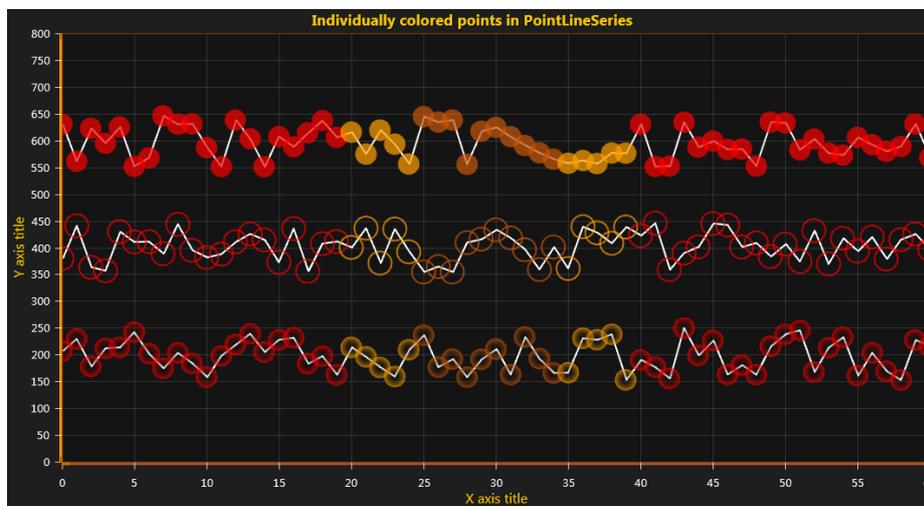


Figure 5-38 In top, **IndividualPointColoring** = **Color1** (solid colored point). In the middle, **IndividualPointColoring** = **BorderColor**. In the bottom, **IndividualPointColoring** = **Color2** (having gradient coloring with **Color1** = transparent).

## 5.5.4 Adding points

You must add the series points in code. Use **AddPoints(SeriesPoint[], bool invalidate)** method to add points to the end of existing points.

```
chart.ViewXY.PointLineSeries[0].AddPoints(pointsArray); //Add points to the  
end
```

To set whole series data at once, and overwrite old points, assign the new point array directly:

```
chart.ViewXY.PointLineSeries[0].Points = pointsArray; //Assign the points  
array
```

**Note! The PointLineSeries points X values must be in ascending order. If you need them to be otherwise ordered, use FreeformPointLineSeries instead.**

For example, definition Points[0].X = 0, Points[1].X = 5, Points[2].X = 5, Points[3].X = 6 is valid.

But Points[0].X = 2, Points[1].X = 1, Points[2].X = 6, Points[3].X = 7 is **not** a **valid** value array for **PointLineSeries**.

### 5.5.5 Adding points, alternative way

You can add points also in X and Y values arrays, which turns to be more convenient way in many applications.

```
chart.ViewXY.PointLineSeries[0].AddPoints(xValuesArray, yValuesArray, false);
```

To set whole series data at once, and overwrite old points, assign the X and Y values arrays directly (applicable in WinForms and WPF Non-bindable APIs).

```
chart.ViewXY.PointLineSeries[0].SetValues(xValuesArray, yValuesArray);
```

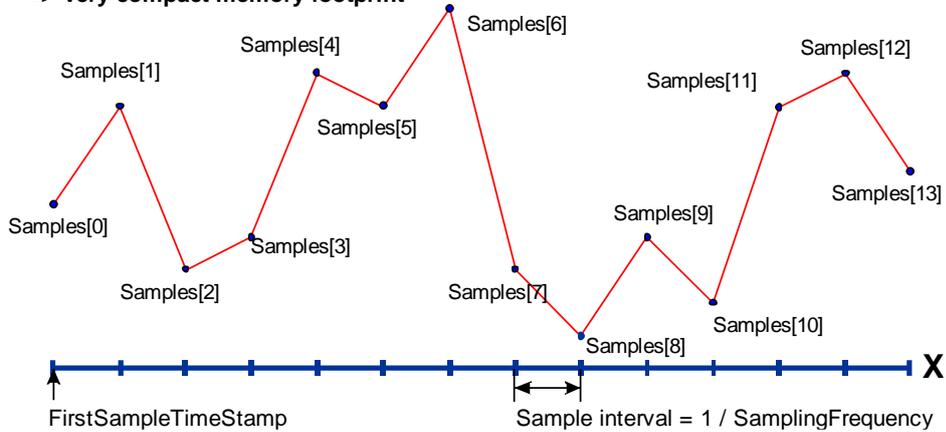
## 5.6 Sample data series

### SampleDataSeries - for fixed interval progressing data

**VERY FAST TO RENDER**

Just Y values are stored in **SamplesSingle** or **SamplesDouble** array

=> Very compact memory footprint



Add the series to chart by adding **SampleDataSeries** objects to **SampleDataSeries** list.

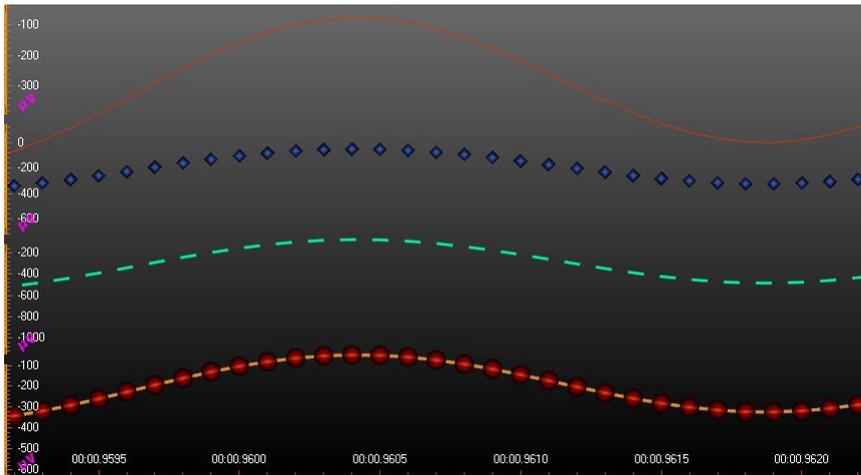


Figure 5-39. Some sample data series.

**SampleDataSeries** is the line series you should use for presenting sampled signal data (discrete signal data). This is generally used in real-time DSP applications. Visually, it is similar to **PointLineSeries**, so all line and point formatting options apply. **SampleDataSeries** has a fixed sample interval, so there's no need to reserve memory to store point X values.

**NOTE!** **SampleDataSeries** does not resample or down-sample the given data. All given data values are retained in the **SamplesSingle** or **SamplesDouble** arrays. **LightningChart** does not reduce the quality of the data, lose peaks or accuracy of the data.

## 5.6.1 Y precision

The **SampleDataSeries** supports single and double precision sample Y values. Using single precision values is recommended when keeping the memory reserving as low as possible. Select the sample format with **SampleFormat** property.

Use the series **SamplingFrequency** (1 / sample interval) to set the fixed sample interval. To set the X value (time stamp) where the samples begin, set **FirstSampleTimeStamp** property.

## 5.6.2 Adding points

You must add the samples in code. Use **AddSamples** method to add samples to the end of existing samples.

```
chart.ViewXY.SampleDataSeries[0].AddSamples(samplesArray, false); //Add samples to the end
```

To set whole series data at once, and overwrite old samples, assign the new samples array directly:

If **SampleFormat** is **SingleFloat**

```
chart.ViewXY.SampleDataSeries[0].SamplesSingle = samplesSingleArray;
```

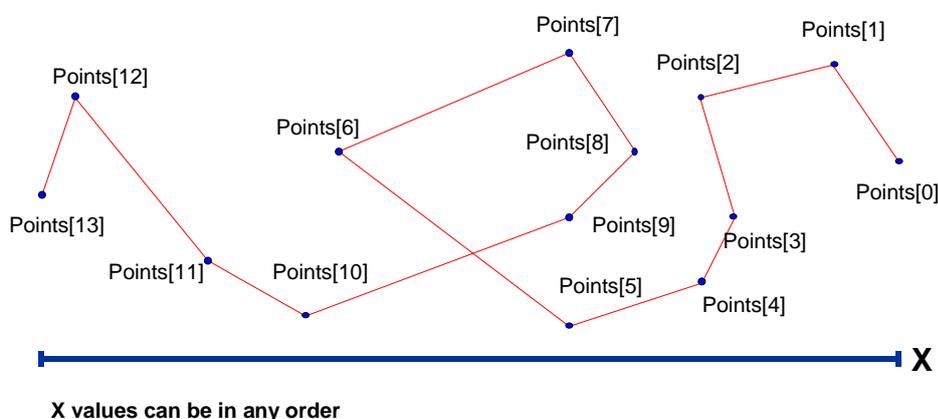
Or if **SampleFormat** is **DoubleFloat**

```
chart.ViewXY.SampleDataSeries[0].SamplesDouble = samplesDoubleArray;
```

## 5.7 Freeform point line series

### FreeformPointLineSeries - for arbitrary data

HEAVY TO RENDER WHEN POINT COUNT IS VERY HIGH



A **FreeformPointLineSeries** can present a simple line, points (scatter) or both as a point line. Add the series to chart by adding **FreeformPointLineSeries** objects to **FreeformPointLineSeries** list. Freeform point line series allows drawing line point to any direction from previous point. All line and point formatting options from **PointLineSeries** apply.

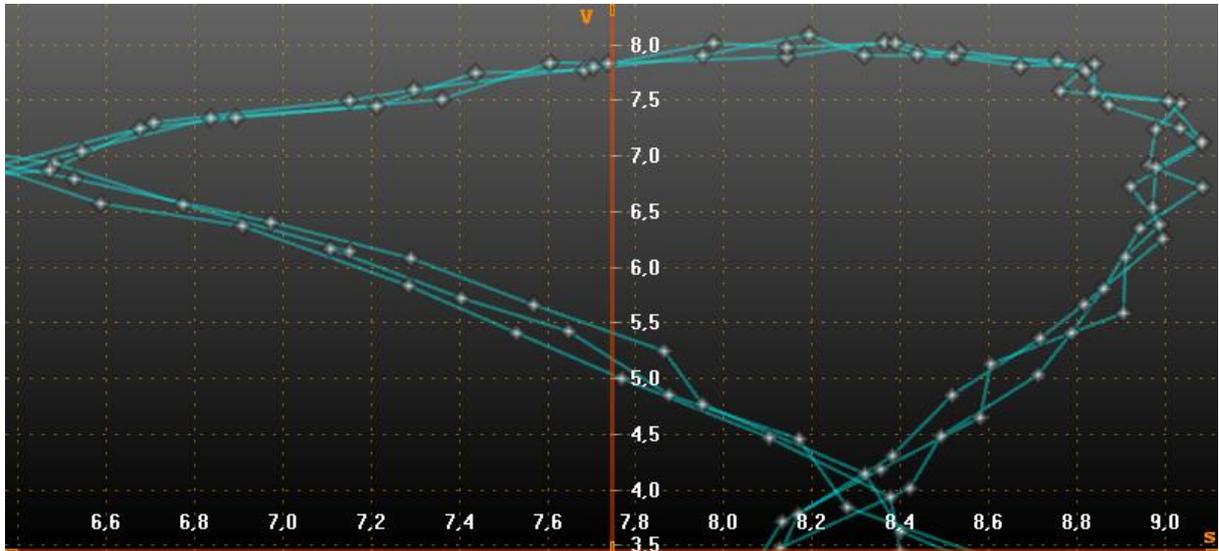


Figure 5-40. A freeform point line series

Freeform point line series line points are not automatically destroyed even if **DropOldSeriesData** is enabled and the points are scrolled out of current view. For automatically destroy old series points in real-time monitoring solution, use point count limiter. Set **PointCountLimitEnabled** = true and set the limit to **PointCountLimit** property. If limiter is enabled, the **Points** array behaves as a ring buffer after the point count limit has been reached. You can always find the oldest point from **Points** array by retrieving value from **OldestPointIndex**. If you need to read the existing data out of point count limited buffer, use the following method:

- If **OldestPointIndex** is 0, read from **Points[0]** till **Points[PointCount-1]**.
- If **OldestPointIndex** > 0, first read from **Points[OldestPointIndex]** till **Points[PointCountLimit-1]**. Then, read from **Points[0]** till **Points[OldestPointIndex-1]**.

To directly retrieve the last series point, call **GetLastPoint()** method.

## 5.8 Advanced line coloring of line series

The line color can be changed based on data values, or other external logic.

### 5.8.1 Y-value based coloring of line and fill with value-range palette

By enabling *UsePalette* property of *SampleDataSeries*, *PointLineSeries* or *FreeformPointLineSeries*, the coloring of line is applied by the *ValueRangePalette* property. *ValueRangePalette* contains Y values and color pairs. *ValueRangePalette.Type* sets the *Gradient* or *Uniform* steps palette.

The palette coloring can be set for Y axis line too. Enable *UsePalette* property of Y axis, and assign the preferred series in *PaletteSeries* property.

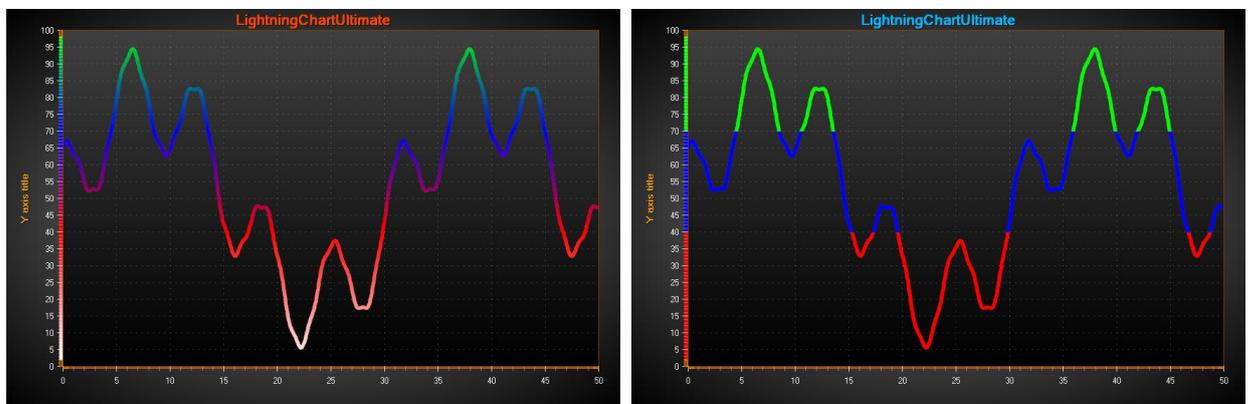


Figure 5-41. On the left, a Gradient palette is used to color the line based on Y values. On the right, a Uniform palette is used. UsePalette is enabled for Y axis too.



Figure 5-42. Gradient palette coloring for bipolar signal data. UsePalette for Y axis is disabled.

## 5.8.2 Custom shaping and coloring with CustomLinePointColoringAndShaping event

Custom coloring and coordinate adjustment can be made with **CustomLinePointColoringAndShaping** event, which is called just before entering the rendering stage of the chart.

Note, the custom coloring is available when **LineStyle.Pattern = Solid**. It also has serious limitations in vector file exports.

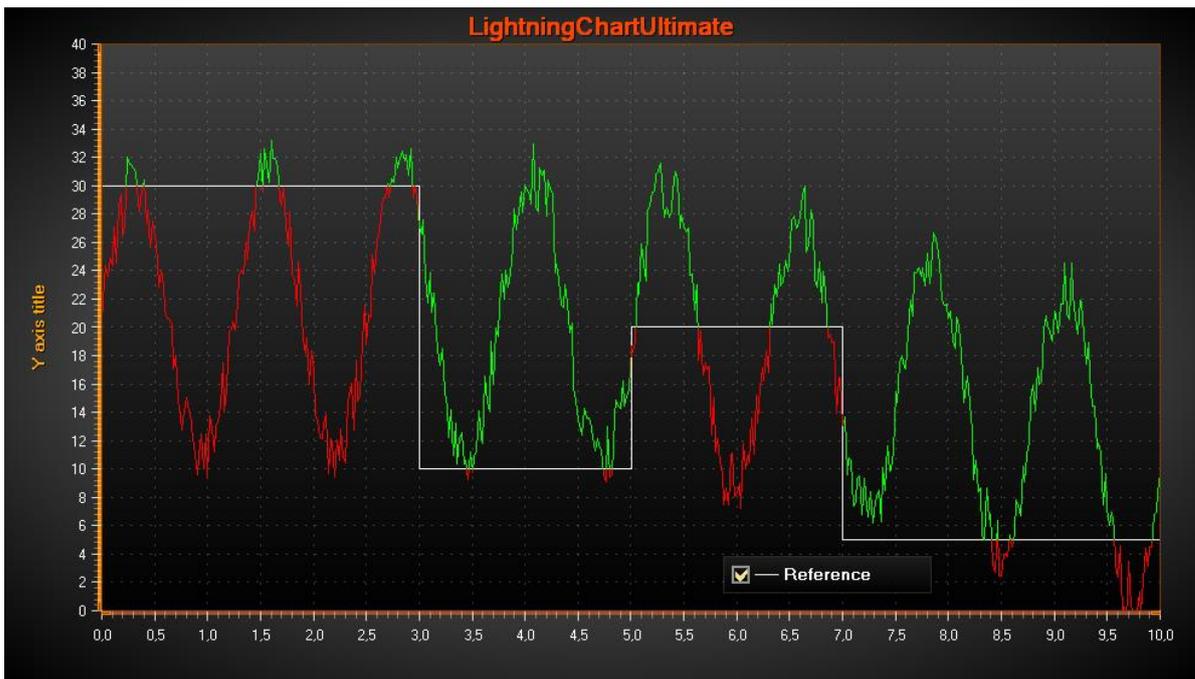


Figure 5-43. CustomLinePointColorAndShaping event handler used to transform color by the specific changing reference level.

The event arguments have the following info:

- **CanModifyColors:** Colors modification is available.
- **Colors:** Prefilled colors array with LineStyle color. If **CanModifyColors** is true, you can modify either by assigning new values to prefilled colors, or by creating new colors array. If **CanModifyColors** is false, don't fill them.
- **CanModifyCoords:** Coordinates modification is available.
- **Coords:** Pre-filled screen coordinates array. If **CanModifyCoords** is true, you can modify either by assigning new values to prefilled coordinates, or by creating new coordinates array. The new array length doesn't have to be equal to prefilled one. **Ensure the length of the Coords and Colors array are equal when exiting the event handler.** If **CanModifyCoords** is false, don't fill them.
- **HasDataPointIndices:** Only applicable in **FreeformPointLineSeries**.
- **DataPointIndices:** Data point indices that were included in the coordinates and colors arrays. Note: chart will skip subsequent points in line construction, if their X and Y values or coordinates

are equal. Using `DataPointIndices` info, you can e.g. pick a color for line point, from data point's `PointColor` field or external color array.

- **SweepPageIndex**: If `XAxis.ScrollMode = 'Sweeping'`, tells the page index (0 or 1).

## 5.9 High-low series

High-low series presents data as filled area between high and low values. Add the series to chart by adding `HighLowSeries` objects into `HighLowSeries` list.

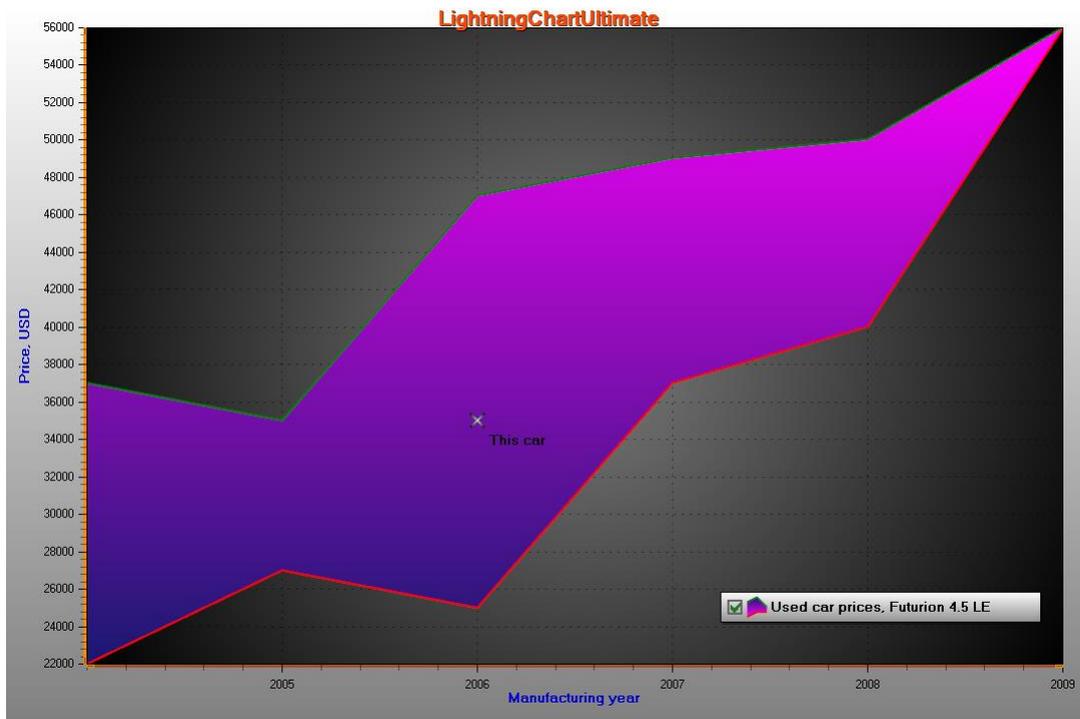


Figure 5-44. A high-low series with a marker over it.

### 5.9.1 Fill, line and point styles

The fill can be set with `Fill` property and its sub-properties. Define the line style with `LineStyleHigh` and `LineStyleLow` properties. If you don't want the high line to be visible, set `LineVisibleHigh = false`, and `LineVisibleLow = false`, respectively. Define the point style with `PointStyleHigh` and `PointStyleLow` properties. If you don't want to show the points, set `PointsVisibleHigh = false`, `PointsVisibleLow = false`. See sections 5.5.1 and 5.5.2 for line and point style details.

When you input data so that High value is less than Low value, reversed fill is applied in that part. Edit the reversed fill with `ReverseFill` property.

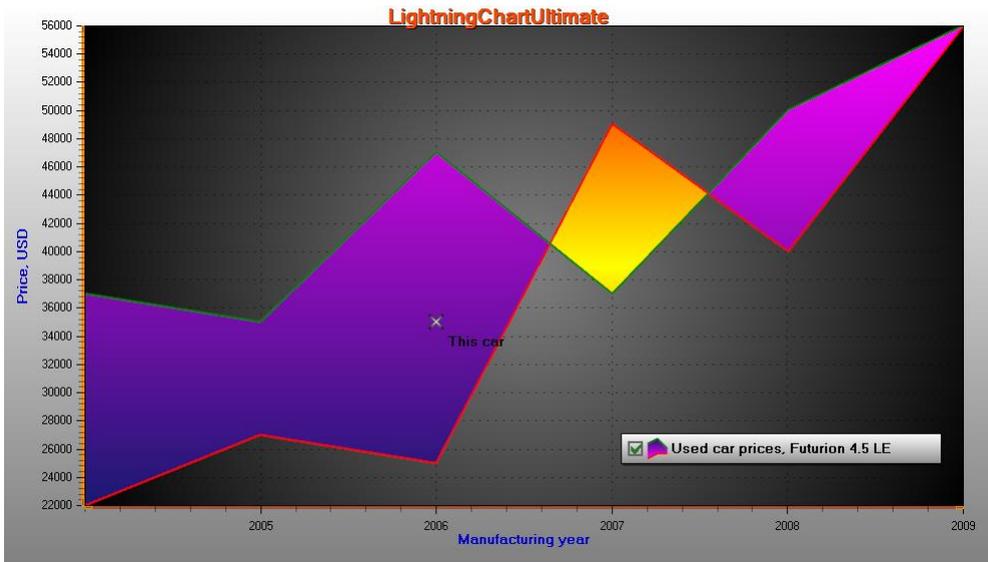


Figure 5-45. Fourth data item is given reversed: high value is < low value.

### 5.9.2 Limits

By enabling *UseLimits*, series shows different solid coloring above the *exceed limit* and below *deceed limit*. The regular *Fill* and *ReverseFill* apply then only for the range between the limits.

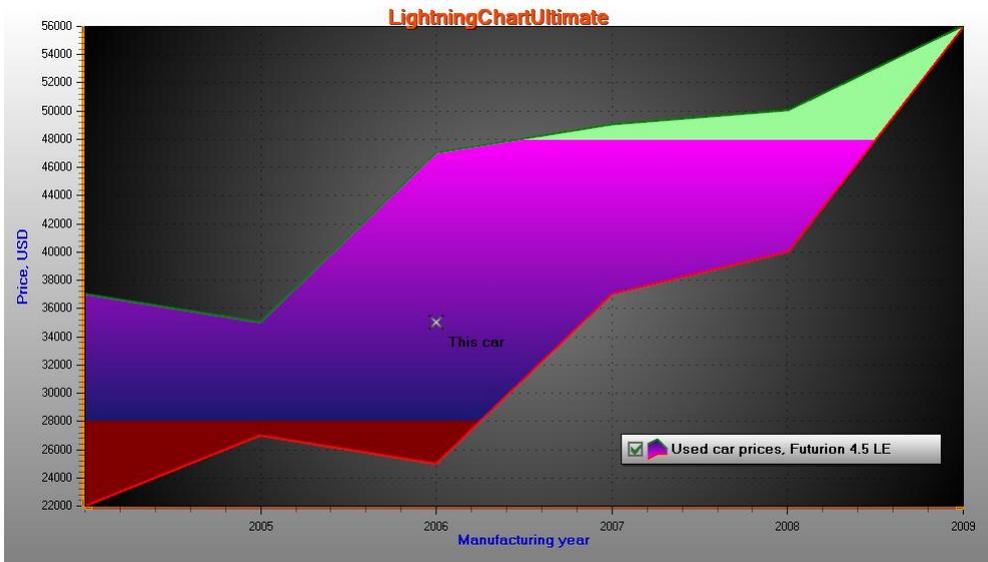


Figure 5-46. UseLimits = true, ExceedLimit = 48000 and DeceedLimit = 28000.

### 5.9.3 Coloring by value-range palette

By enabling *UsePalette*, the fill uses *ValueRangePalette* steps. *Uniform* and *Gradient* coloring are both supported.

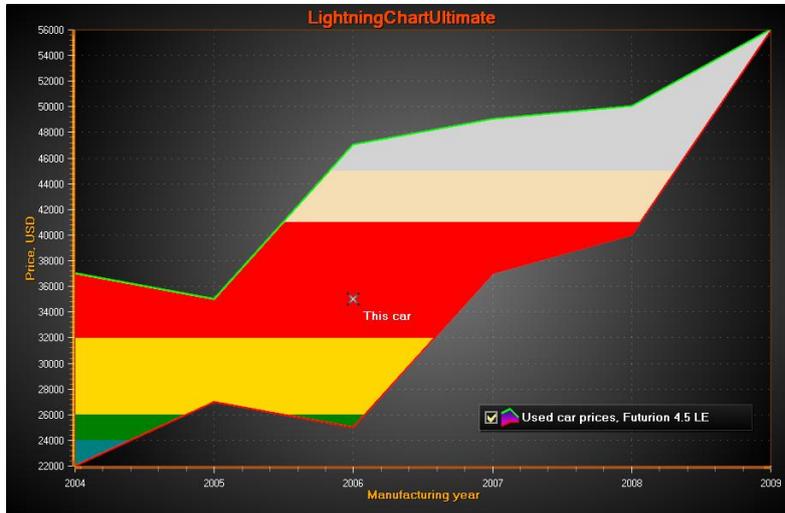


Figure 5-47. UsePalette = True, several steps defined in ValueRangePalette. Uniform coloring.

### 5.9.4 Adding data

You must add the data values in code. The data must be given in ascending order by X values, **Points[i+1].X ≥ Points[i].X**.

Use *AddValues(HighLowSeriesPoint[], bool invalidate)* method to add data values to the end of existing values array.

```
HighLowSeriesPoint[]dataArray = new HighLowSeriesPoint[6];
dataArray [0] = new HighLowSeriesPoint(2004, 37000, 22000);
dataArray [1] = new HighLowSeriesPoint(2005, 35000, 27000);
dataArray [2] = new HighLowSeriesPoint(2006, 47000, 25000);
dataArray [3] = new HighLowSeriesPoint(2007, 37000, 49000);
dataArray [4] = new HighLowSeriesPoint(2008, 40000, 50000);
dataArray [5] = new HighLowSeriesPoint(2009, 56000, 56000);

//Add data to the end
chart.ViewXY.HighLowSeries[0].AddValues(dataArray, true);
```

To set whole series data at once, and overwrite old data, assign the new data array directly:

```
//Assign the data into points array
chart.ViewXY.HighLowSeries[0].Points = dataArray;
```

## 5.10 Area series

Area series presents data as filled area between base level and values. Add the series to chart by adding **AreaSeries** objects into **AreaSeries** list. Area series is quite similar to **HighLowSeries** described in section 5.9, but simpler.

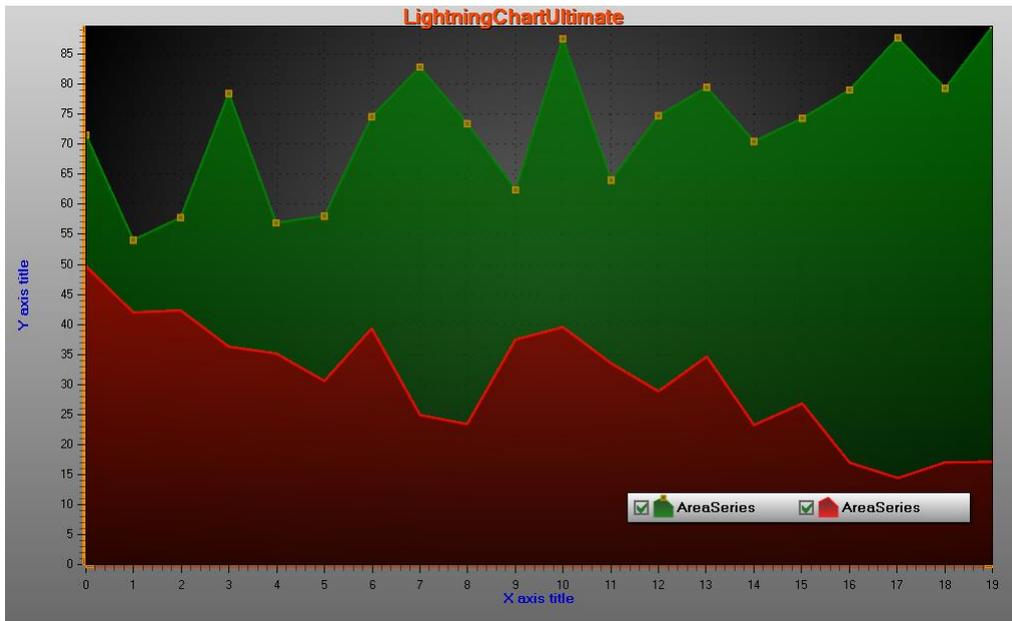


Figure 5-48. Two area series. BaseValue = 0 in both. Points are visible here only in first series.

Set base level with **BaseValue** property. Set the preferred fill style with **Fill** property. Line style can be set with **LineStyle** property. Point style can be set with **PointStyle** property. Exceed and deceed limits can be used like in **HighLowSeries**.

### 5.10.1 Adding data

You must add the data values in code. The data must be given in ascending order by X values, **Points[i+1].X ≥ Points[i].X**.

Use **AddValues(HighLowSeriesPoint[], bool invalidate)** method to add data values to the end of existing values array.

```
AreaSeriesPoint[] dataArray = new AreaSeriesPoint[6];
dataArray [0] = new AreaSeriesPoint (2004, 37000);
dataArray [1] = new AreaSeriesPoint (2005, 35000);
dataArray [2] = new AreaSeriesPoint (2006, 47000);
dataArray [3] = new AreaSeriesPoint (2007, 37000);
dataArray [4] = new AreaSeriesPoint (2008, 40000);
dataArray [5] = new AreaSeriesPoint (2009, 56000);

//Add data to the end
chart.ViewXY.AreaSeries[0].AddValues(dataArray, true);
```

To set whole series data at once, and overwrite old data, assign the new data array directly:

```
//Assign the data into points array
chart.ViewXY.AreaSeries[0].Points = dataArray;
```

## 5.11 Bars

Bar series allows displaying data in horizontal or vertical bars.

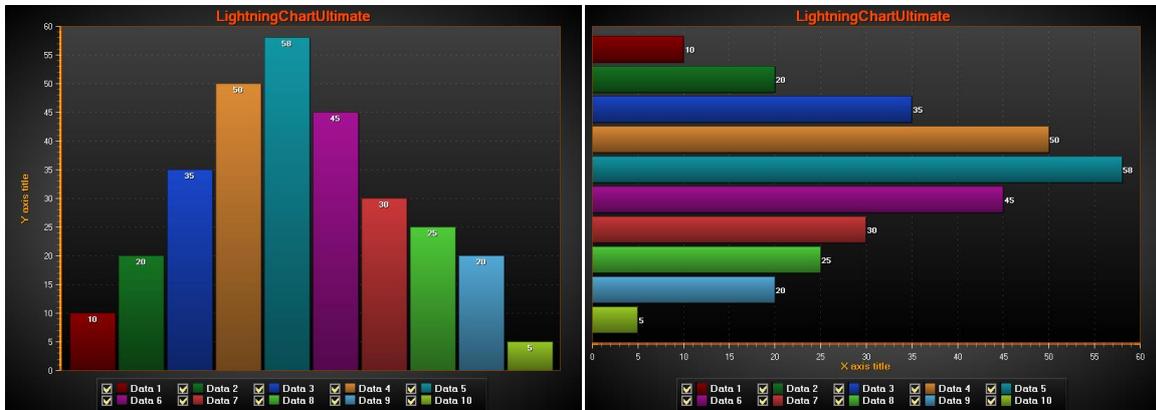


Figure 5-49. Bars series, vertical and horizontal.

Use **Values** array property to store the values of a bar series. Add values with **AddValue(...)** method. Update existing value by given value index with **SetValue(...)** method. The values are of type **BarSeriesValue**, which has the following fields:

- **Value** The bar length.
- **Location** Location of bar in the X axis (vertical presentation) or Y axis (horizontal presentation).
- **Text** The text that appears in the bar.

Use **LabelStyle** property of a bar series to control how the text bar value label appears on the chart. The label value text is set by **AddValue(...)** or **SetValue(...)** method parameter. Various fill styles can be used by setting **Fill** property and its sub-properties.

Use **BarViewOptions** property of the chart to control how the bars are displayed. **BarView.Options.Orientation** to select between **Horizontal** and **Vertical** bar orientation.

**BarViewOptions.Grouping** allows grouping the bars by value indices, by indices using width fitting and by Location values. Grouping tells to bring values from different bar series visually together. If you don't want any grouping, use **BarViewOptions.Grouping.ByLocation** and set different **Location** field for every **BarSeriesValue** object. Use width fitting properties to adjust the spaces between columns and aside them. When no width fitting is used, **BarThickness** property of the bar series rules the bar width.

The groups can be stacked by setting **BarViewOptions.Stacking** to **Stack** or **StackStretchToSum**. When using **StackStretchToSum**, define the target sum by setting **StackSum** property. It is 100 by default to represent 100 %.

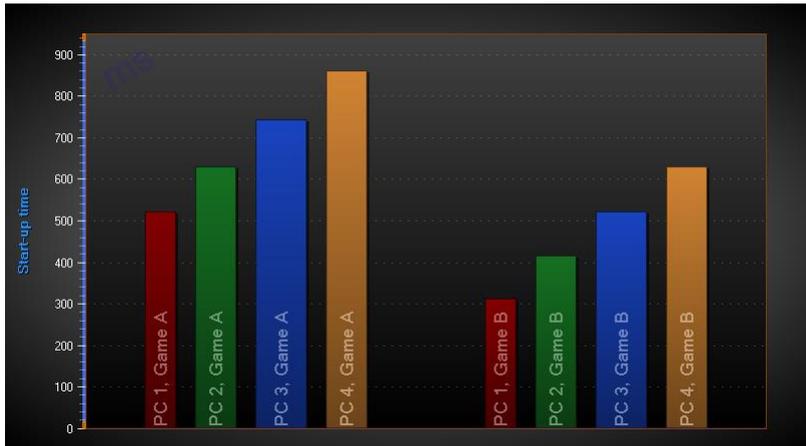


Figure 5-50. Bars series Grouping = ByIndex, Stacking = None.

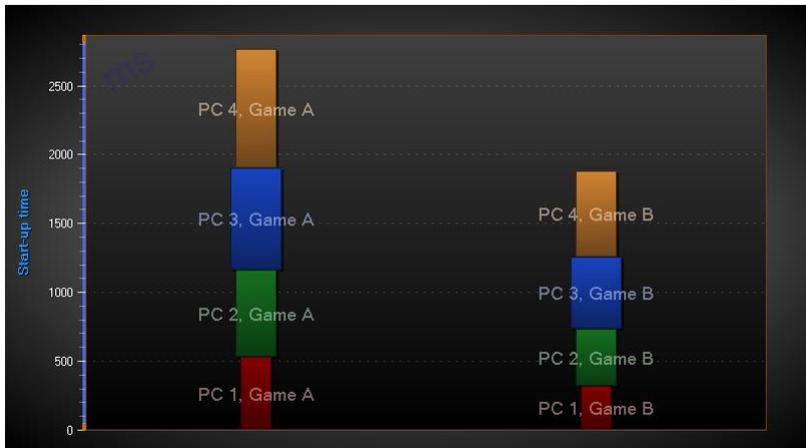


Figure 5-51. Bars series Grouping = ByIndex, Stacking = Stack.

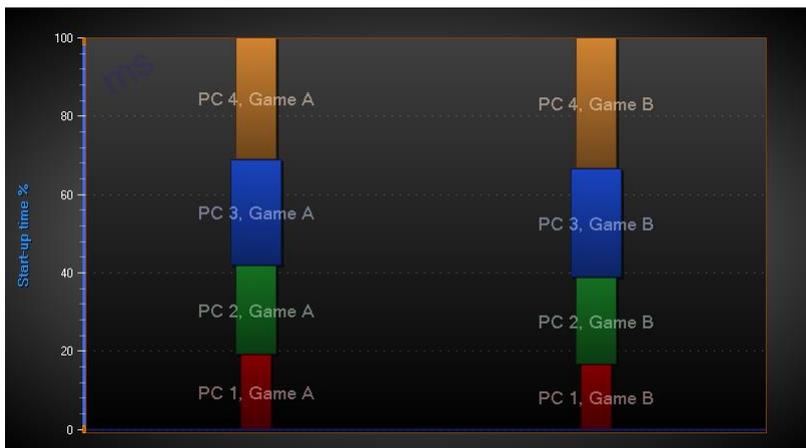


Figure 5-52. Bars series Grouping = ByIndex, Stacking = StackStretchToSum. StackSum = 100.

**BaseLevel** property in **BarSeries** is series minimum value for all values and sets bar start position. In **Stacked** view, it will increase (if positive) or decrease (if negative) the size of the bar. If **StackedToSum**, the bar size is relative and calculated like **Stacked**.

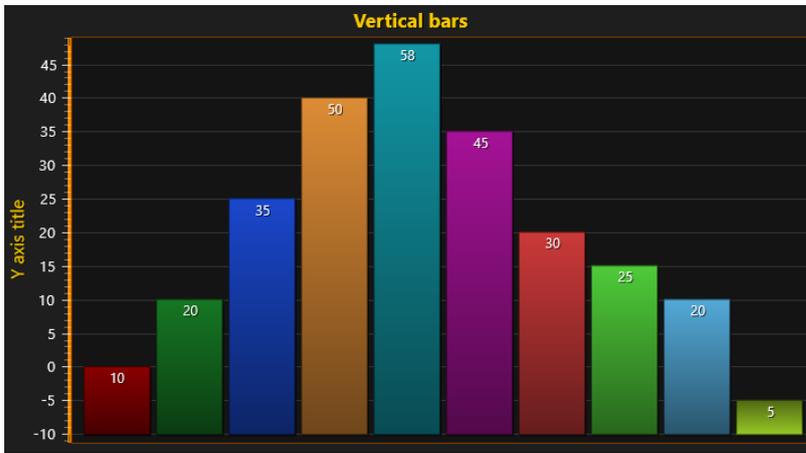


Figure 5-53. BaseLevel set to -10. Bar values are 10, 20, 35, 50, 58, 45, 30, 25, 20, 5.

## 5.12 Stock series

Stock series allow stock exchange data visualization in candle-stick or stock bars format. You can add several Stock series in the same chart, by adding several *StockSeries* objects in *StockSeries* list property. Select the style with Style property. The options are: *Bars*, *CandleStick* and *OptimizedCandleStick*.



Figure 5-54. One StockSeries with Style = CandleStick. Light blue line is a PointLineSeries in the background, going through all Close values.



Figure 5-55. One StockSeries with Style = Bars. Line series are used for showing linear regression fit, and offset of that line (2 \* standard deviation). A band is used for selecting a date range for line fit.

Set the coloring and filling options with **ColorStickDown**, **ColorStickUp**, **FillDown** and **FillUp** properties. Adjust the stick width in pixels with **StickWidth** property. Adjust the total data item width with **ItemWidth** property.

For maximum rendering performance, use **Bars** style, with **StickWidth** = 1.

For maximum rendering performance with candle sticks, use **OptimizedCandleStick** style. **OptimizedCandleSticks** are significantly faster. **OptimizedCandleStick** has only limited set of fill effects available - it supports **Solid** and left-to-right direction **Linear** fill. Bitmap, Radial, RadialStretched and Cylindrical fills are not supported. **OptimizedCandleSticks** don't support borders of the candles - **FillBorder** is not applicable.

**StockSeries** can be set render before the line series, by setting **Behind** = True.

### 5.12.1 Setting data to StockSeries

Create data array and set the array items. Each item has fields:

<b>Date</b>	DateTime value (year, month, day)
<b>Open</b>	The opening value of the day
<b>Close</b>	Close value of the day
<b>Low</b>	Lowest value during day
<b>High</b>	Highest value during day
<b>Transaction</b>	The total trading sum
<b>Volume</b>	Count of shares traded

You should keep the data always in ascending order by Date value (oldest date first).

```
// Create data array
StockSeriesData[] data = new StockSeriesData[] {
    new StockSeriesData(2010,09,01, 24.35, 24.76, 24.81, 23.82,
        269210, 6610451.55),
    new StockSeriesData(2010,09,02, 24.85, 24.66, 24.85,
        24.53, 216395, 5356858.225),
    new StockSeriesData(2010,09,03, 24.80, 24.84, 25.07,
        24.60, 164583, 4084950.06),
    new StockSeriesData(2010,09,06, 24.85, 25.01, 25.12,
        24.84, 118367, 2950889.31)
};
// Assign the data array to series
chart.ViewXY.StockSeries[0].DataPoints = data;
```

## 5.12.2 Setting X axis to date display

```
chart.ViewXY.XAxes[0].ValueType = AxisValueType.DateTime;
chart.ViewXY.XAxes[0].LabelsAngle = 90;
chart.ViewXY.XAxes[0].LabelsTimeFormat =
    System.Globalization.CultureInfo.CurrentCulture.DateTimeFormat
        .ShortDatePattern;
chart.ViewXY.XAxes[0].MajorDiv = 24 * 60 * 60; //major div is one day in seconds
chart.ViewXY.XAxes[0].AutoFormatLabels = false;

//Set datetime origin
chart.ViewXY.XAxes[0].DateOriginYear = data[0].Date.Year;
chart.ViewXY.XAxes[0].DateOriginMonth = data[0].Date.Month;
chart.ViewXY.XAxes[0].DateOriginDay = data[0].Date.Day;
```

Set the X axis range suitable for data:

```
// x-axis stretched half a day at both ends. Use first and last date value
chart.ViewXY.XAxes[0].SetRange(
    chart.ViewXY.XAxes[0].DateTimeToAxisValue(data[0].Date) - 12 * 60 * 60,
    chart.ViewXY.XAxes[0].DateTimeToAxisValue(data[data.Length - 1].Date) + 12
        * 60 * 60);
```

## 5.12.3 Custom formatting of appearance

The **StockSeries** has **CustomStockDataAppearance** event handler, which can be used to format appearance of series data items individually, overriding the generic fill and color styles applied with properties. In the event handler, you can give your own width and colors for specific points.



Figure 5-56. CustomStockDataAppearance used to highlight specific data items with greater width and brighter, gradient color.

## 5.12.4 Applying Scale breaks

To cut off non-trading hours and days, see 5.3.2.

## 5.13 PolygonSeries

**PolygonSeries** renders a fill and borderline, by given border path.

Set the filling preferences in Fill property. Use Border property to set the border line style.

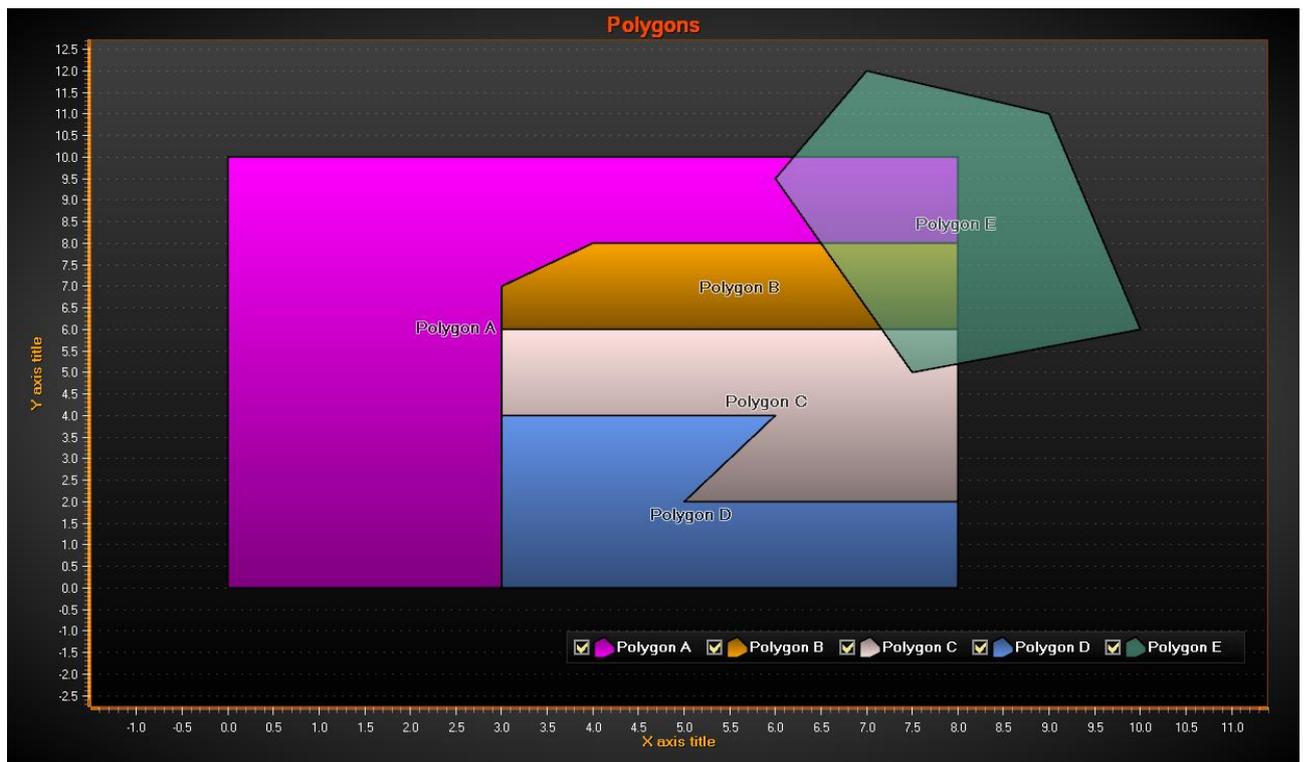


Figure 5-57. Several polygons.

### 5.13.1 Setting data to a Polygon

Set the path points in **Points** property. **PolygonSeries** has an automatic path closing feature. If you don't connect last point to first point, the chart will do that automatically.

This example shows how to assign the points of the previous picture's transparent teal polygon's path:

```
polygon.Points = new PointDouble2D[] {  
    new PointDouble2D(7,12),  
    new PointDouble2D(6,9.5),  
    new PointDouble2D(7.5,5),  
    new PointDouble2D(10,6),  
    new PointDouble2D(9,11)};
```

### 5.13.2 Enabling complex / intersecting fills

Set **IntersectionsAllowed = True** to enable polygon path to intersect itself. Without enabling this property and intersecting the path, the fill will appear all garbled up. By default, the property is **False** for performance reasons, as intersecting detecting and rendering in intersecting cases are heavy.

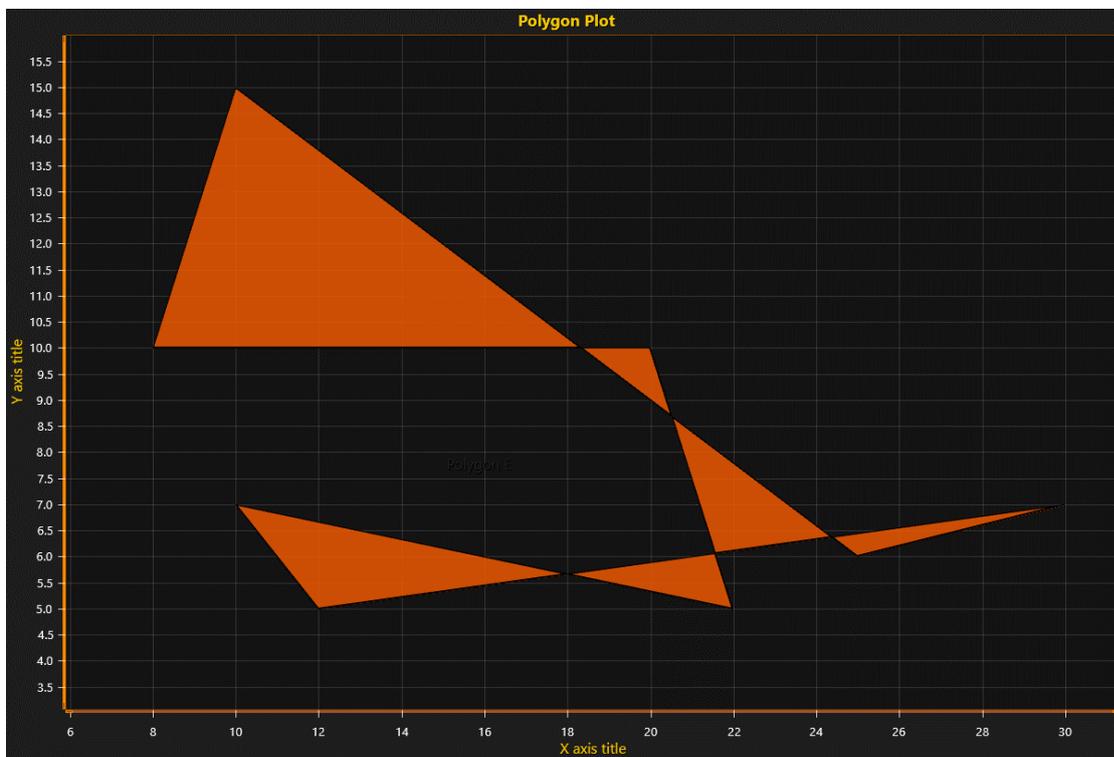


Figure 5-58. Polygon with intersecting path, with IntersectionsAllowed = True.

## 5.14 LineCollections

A **LineCollection** is a collection of line segments. Each line segment is a line going from point A to B. One **LineCollection** can contain thousands of line segments. **LineCollection** is extremely efficient in rendering of thousands of distinct line segments, in contrast to **PointLineSeries**, **FreeformPointLineSeries** or **SampleDataSeries**. **PointLineSeries**, **FreeformPointLineSeries** or **SampleDataSeries** are more efficient in rendering continuous polylines of millions of points.

Use **LineStyle** property to control the line color, style and width. Set the line segments in **Lines** property.

Add the **LineCollection** object in **ViewXY.LineCollections** list property.

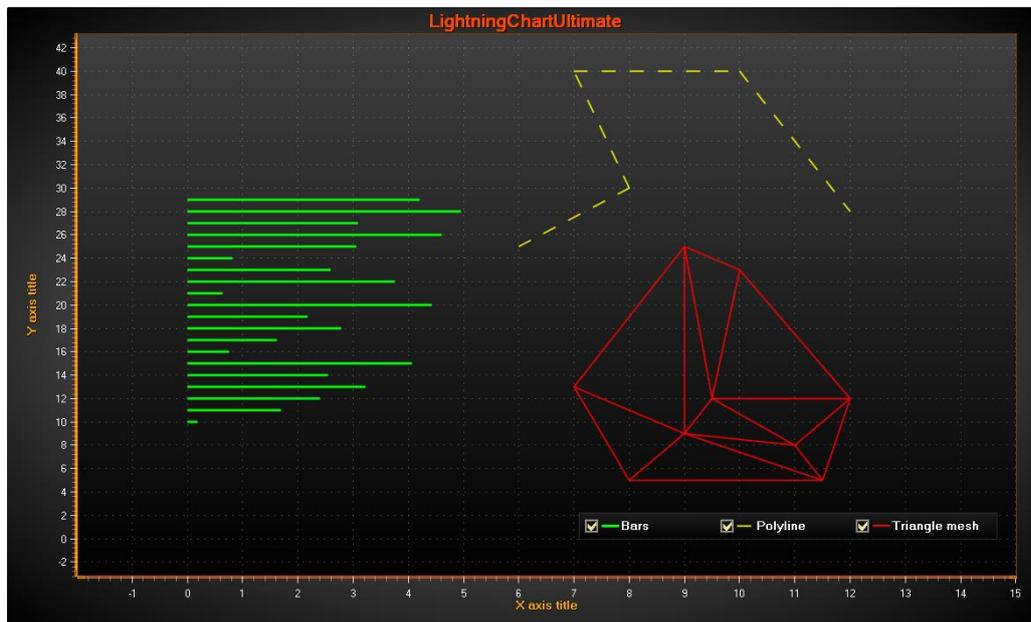


Figure 5-59. Three LineCollections in use. Green acts as very rapidly rendering bars, yellow as a polyline, and red as an arbitrary triangle wireframe mesh.

### 5.14.1 Setting data to a LineCollection

SegmentLine structure consists of four fields:

<b>AX</b>	Start point, X
<b>AY</b>	Start point, Y
<b>BX</b>	End point, X
<b>BY</b>	End point, Y

Add the SegmentLines array to Lines property as follows:

```
lineCollection.Lines = new SegmentLine[] {  
    new SegmentLine(6,25,8,30),  
    new SegmentLine(8,30,7,40),  
    new SegmentLine(7,40,10,40),  
    new SegmentLine(10,40,12,28) };
```

## 5.15 IntensityGridSeries

**IntensityGridSeries** allows visualizing M x N array of nodes, colored by assigned value-range palette. The colors between nodes are interpolated. **IntensityGridSeries** is an evenly-spaced, rectangular series, in X and Y dimension. This series allows rendering contour lines, contour line labels, and wireframe as well.

Misc	
AssignableXAxes	String[] Array
AssignableYAxes	String[] Array
AssignXAxisIndex	0
AssignYAxisIndex	0
ContourLineLabels	
ContourLineStyle	
ContourLineType	<b>ColorLine</b>
Data	<b>IntensityPoint[,] Array</b>
DisableDragToAnotherAxis	True
FastContourZoneRange	1
Fill	Paletted
FullInterpolation	False
IncludeInAutoFit	True
InitialValue	0
LegendBoxUnits	Units
LegendBoxValuesFormat	0.0
LegendBoxValueType	Number
LimitYToStackSegment	<b>False</b>
MouseHighlight	<b>None</b>
MouseInteraction	<b>False</b>
Optimization	DynamicData
PixelRendering	False
RangeMaxX	<b>90</b>
RangeMaxY	<b>90</b>
RangeMinX	<b>10</b>
RangeMinY	<b>10</b>
ShowInLegendBox	True
ShowNodes	False
SizeX	<b>200</b>
SizeY	<b>200</b>
Stencil	
Title	<b>Series Title: Data surface</b>
ToneColor	Black
TraceMouseCell	<b>True</b>
ValueRangePalette	
Visible	True
WireframeLineStyle	
WireframeType	None

Figure 5-60. IntensityGridSeries properties.

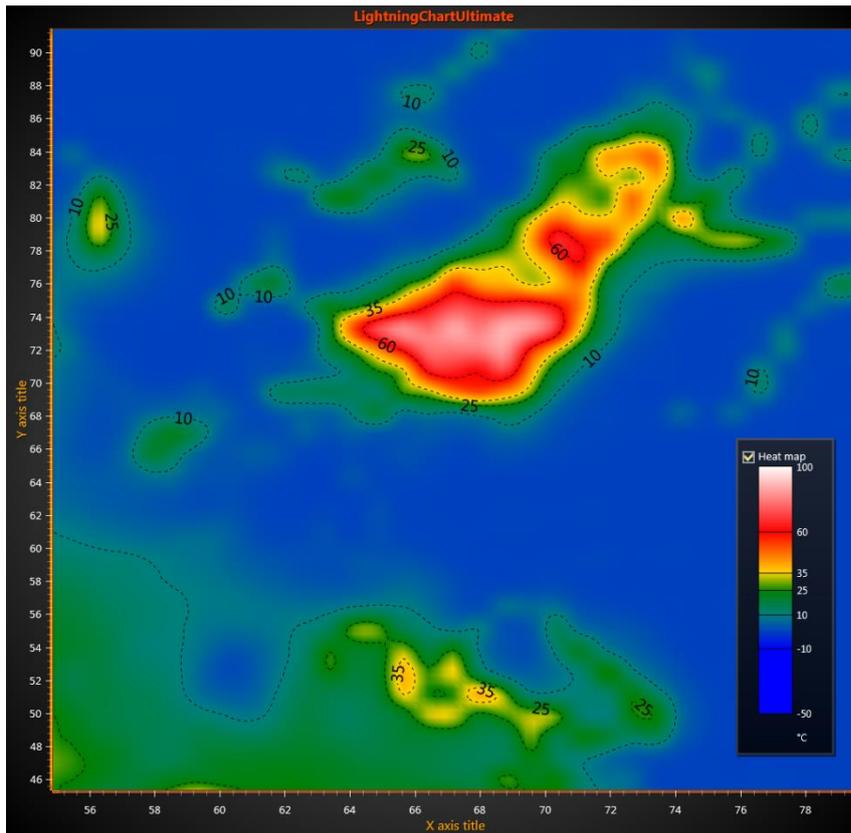


Figure 5-61. IntensityGrid series showing a heat map presentation. Legend box shows the value-range color palette.

The data is stored in Data property, as two-dimensional array. Each array item is of type *IntensityPoint*. Store the data value of each node in *Value* field of *IntensityPoint* structure, which tells what color should be used from the *ValueRangePalette*.

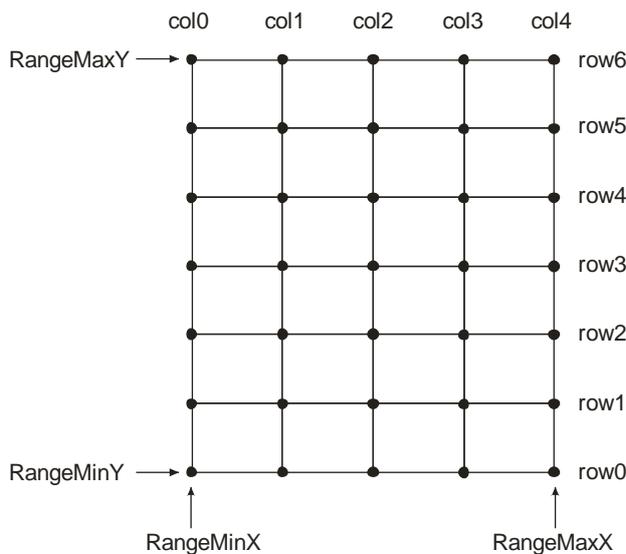


Figure 5-62. IntensityGridSeries nodes. SizeX = 5, SizeY = 7.

Node distances are automatically calculated as

$$\text{node distance } X = \frac{\text{RangeMaxX} - \text{RangeMinX}}{\text{SizeX} - 1}$$

$$\text{node distance } Y = \frac{\text{RangeMaxY} - \text{RangeMinY}}{\text{SizeY} - 1}$$

### 5.15.1 Setting intensity grid data

- Set X range by using **RangeMinX** and **RangeMaxX** properties, to order the minimum and maximum value in assigned X axis.
- Set Y range by using **RangeMinY** and **RangeMaxY** properties, to order the minimum and maximum value in assigned Y axis.
- Set **SizeX** and **SizeY** properties to give the grid a size as columns and rows.
- Set **Value** for each node:

#### Method, with Data array index

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++)  
    {  
        intensityValue = //some height value.  
        gridSeries.Data[iNodeX, iNodeY].Value = intensityValue;  
    }  
}  
gridSeries.InvalidateData(); //Notify new values are ready and to refresh
```

#### Alternative method, usage of SetDataValue

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++)  
    {  
        intensityValue = //some height value  
        gridSeries.SetDataValue(nodeIndexX, nodeIndexY,  
            0, //X value is irrelevant in grid  
            0, //Y value is irrelevant in grid  
            intensityValue,  
            Color.Green); //Source point colors are not used in this  
            example, so use any color here  
    }  
}  
gridSeries.InvalidateData(); //Notify new values are ready and to refresh
```

### Setting Values only to existing grid

When the geometry of IntensityMesh doesn't change or SizeX or SizeY for IntensityGrid series doesn't change, and data is changing rapidly, it is most advantageous to use **SetValuesData** method. It accepts Double[][] format values. With this kind of data array, scrolling or re-ordering rows or columns is quick. Especially this combined to **PixelRendering** property (see 5.15.4) is a very effective approach for high-resolution scrolling spectrogram visualization. Note that when **PixelRendering** is **disabled** with external data array set by SetValuesData, **Data** property can't be null.

### Setting Colors only to existing grid

When the geometry of IntensityMesh doesn't change or SizeX or SizeY for IntensityGrid series doesn't change, and data is changing rapidly, it is most advantageous to use **SetColorsData** method. It accepts int[][] format values, i.e. ARGB values that GPU accepts directly. With this kind of data array, scrolling or re-ordering rows or columns is quick. Especially this combined to **PixelRendering** property (see 5.15.4) is a very effective approach for high-resolution scrolling spectrogram visualization. Note that when **PixelRendering** is **disabled** with external data array set by SetColorsData, **Data** property can't be null.

## 5.15.2 Creating intensity grid data from bitmap file

You can create a surface from a bitmap image. Use **SetHeightDataFromBitmap** method to achieve that. The series **Data** array property gets the size of the bitmap size (if no anti-aliasing or resampling is used). For each bitmap image pixel, Red, Green and Blue values are summed. The greater the sum, the greater will be the data value for that node. Black and dark colors get lower values and bright and white colors get higher values.

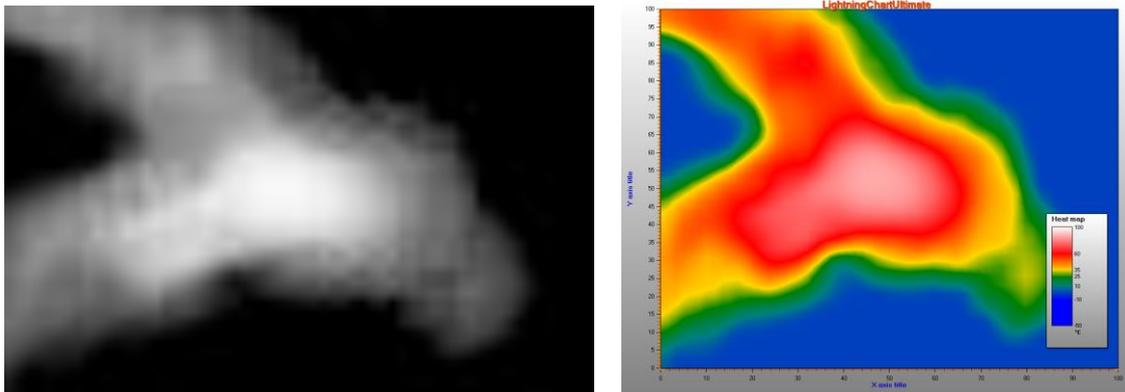


Figure 5-63. Source bitmap and calculated intensity values data. Dark values stay low and bright values get higher values.

### 5.15.3 Fill styles

Use **Fill** property to select the filling style. The following options are available

- **None:** By using this, no filling is applied. This is the selection you may want to use with wireframe mesh or plain contour lines.
- **FromSurfacePoints:** The colors of the Data property nodes are used.
- **Toned:** ToneColor applies
- **Paletted:** see section 5.15.4.

Enable **FullInterpolation** property to use enhanced interpolation method in the fill. Note that it will cause more CPU and GPU overhead. By using full interpolation, the fill quality is better, but can be seen only when the data array size is quite small.

### 5.15.4 Rendering as pixel map

By enabling **PixelRendering** property, the nodes are rendered as pixels, or rectangles. This is a very high-performance rendering style e.g. for real-time high-resolution thermal imaging applications. Note that when this rendering mode is selected, many other options are disabled, such as contour lines, wireframe and interpolation. If logarithmic axes are used, the logarithmic transformation is only applied to corners of the series, the pixels in the bitmap remain evenly spaced and no logarithmic transformation is applied to them.

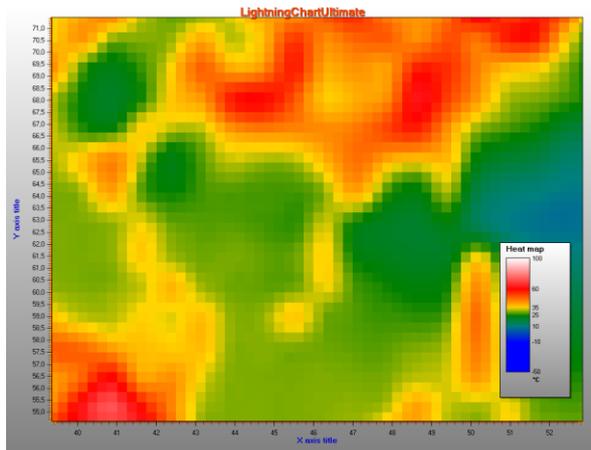


Figure 5-64. PixelRendering = true.

### 5.15.5 ValueRangePalette

With **ValueRangePalette** property, you can define color steps for value coloring. **ValueRangePalette** can be used for

- **Fill** (see section 5.15.3)
- **Wireframe** (see section 5.15.6)
- **Contour lines** (see section 0)

For contour palette, you can define several steps. Each step has a height value and the corresponding color.

**Note: 20 steps are precompiled and load fast.** With higher step counts, several seconds delay is to be expected when initializing the chart.

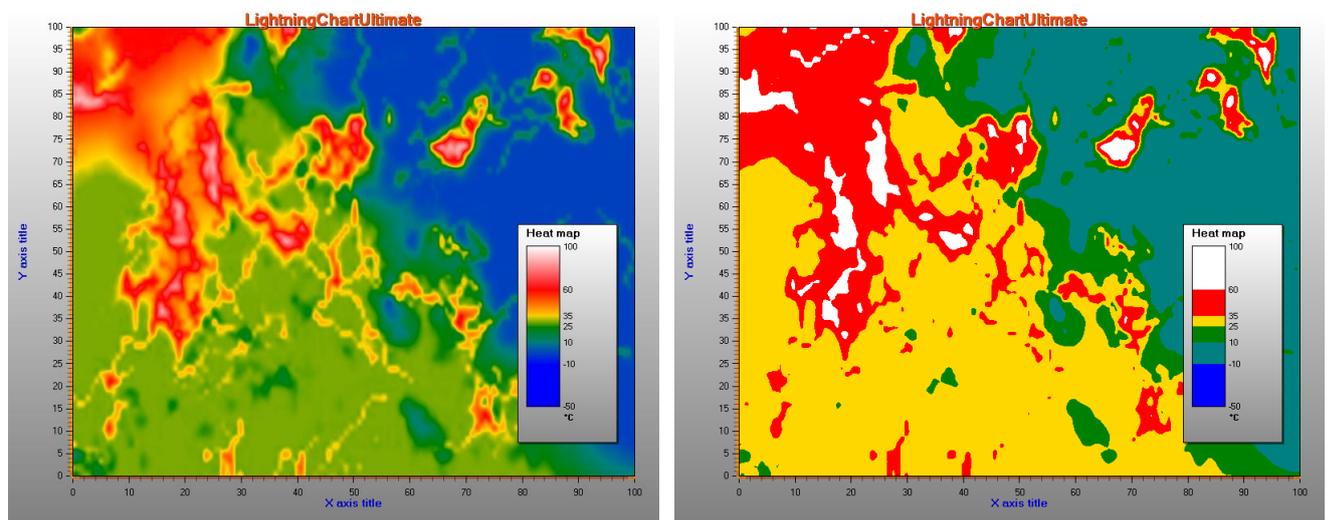


Figure 5-65. On the left, IntensityGridSeries Fill is set to Paletted and palette Type is set to Gradient. On the right, palette Type is set to Uniform.

The palette is defined with **MinValue**, **Type** and **Steps** properties. For **Type**, there's two choices: **Uniform** and **Gradient**. The contour palette of the previous figures (pay attention the legend boxes):

- **MinValue**: -50
- **Type**: Uniform
- **Steps**:
  - Steps[0]: MaxValue:-10, Color: Blue
  - Steps[1]: MaxValue:10, Color: Teal
  - Steps[2]: MaxValue:25, Color: Green
  - Steps[3]: MaxValue:35, Color: Yellow
  - Steps[4]: MaxValue:60, Color: Red
  - Steps[5]: MaxValue:100, Color: White

The values below first step value are colored with first step's color.

### 5.15.6 Wireframe

Use **WireframeType** to select the wireframe style. The options are:

- **None**: no wireframe
- **Wireframe**: a solid color wireframe. Use **WireframeLineStyle.Color** to set the color
- **WireframePaletted**: the wireframe coloring follows **ValueRangePalette** (see section 5.15.4)
- **WireframeSourcePointColored**: the wireframe coloring follows the color of grid nodes
- **Dots**: solid color dots are drawn in the grid node positions
- **DotsPaletted**: dots are drawn in the grid node positions, and colored by **ValueRangePalette**
- **DotsSourcePointColored**: dots are drawn in the grid node positions, coloring follows the color of grid nodes

The wireframe line style (color, width, pattern) can be edited by using **WireframeLineStyle**.

**Note!** Palette colored wireframe lines and dots are available only when **WireframeLineStyle.Width = 1** and **WireframeLineStyle.Pattern = Solid**.

### 5.15.7 Contour lines

Contour lines can be used combined with fill and wireframe. By setting **ContourLineType** property, contour lines can be drawn with different styles:

- **None**: no contour lines are shown
- **FastColorZones**: The lines are drawn as thin zones on palette step end. Very powerful rendering, suits very well for continuously updated or animated surface. Steep value changes are shown as thin line, as gently sloping height differences are shown with thick zone. All lines use same color defined with **ContourLineStyle.Color** property. The zone width can be set by **FastContourZoneRange** property. The value is in Y axis range.
- **FastPalettedZones**: Like **FastColorZones**, but line coloring follows **ValueRangePalette** options (see section 5.15.4)
- **ColorLine**: Like **FastColorZones**, but the contour lines are made with actual lines. Rendering takes longer and is not recommended for continuously updated or animated surface. The line width can be adjusted with **ContourLineStyle.Width** property.
- **PalettedLine**: Like **ColorLine**, but line coloring follows **ValueRangePalette** options.

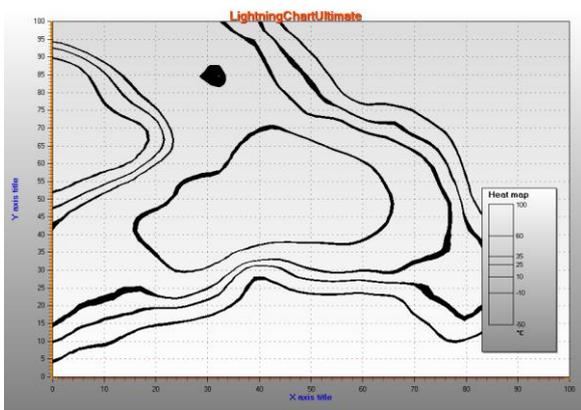


Figure 5-67. ContourLineType = FastColorZones.



Figure 5-66. ContourLineType = FastPalettedZones.

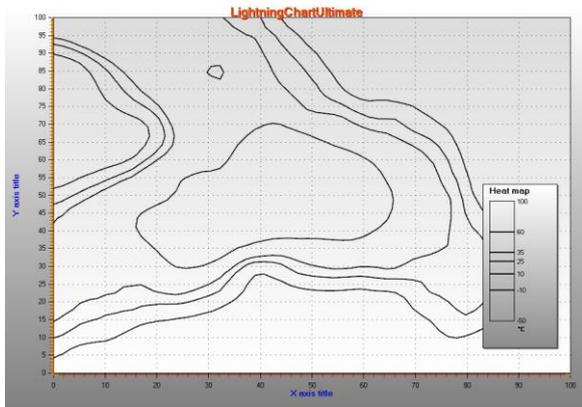


Figure 5-68. ContourLineType = ColorLine



Figure 5-69. ContourLineType = PalettedLine

### 5.15.8 Contour line labels

When contour lines are shown, numeric values can be shown within the line paths.

ContourLineLabels	
Color	Black
Font	Segoe UI, 15pt, style=Bold
LabelsNumberFormat	0.0
Visible	True

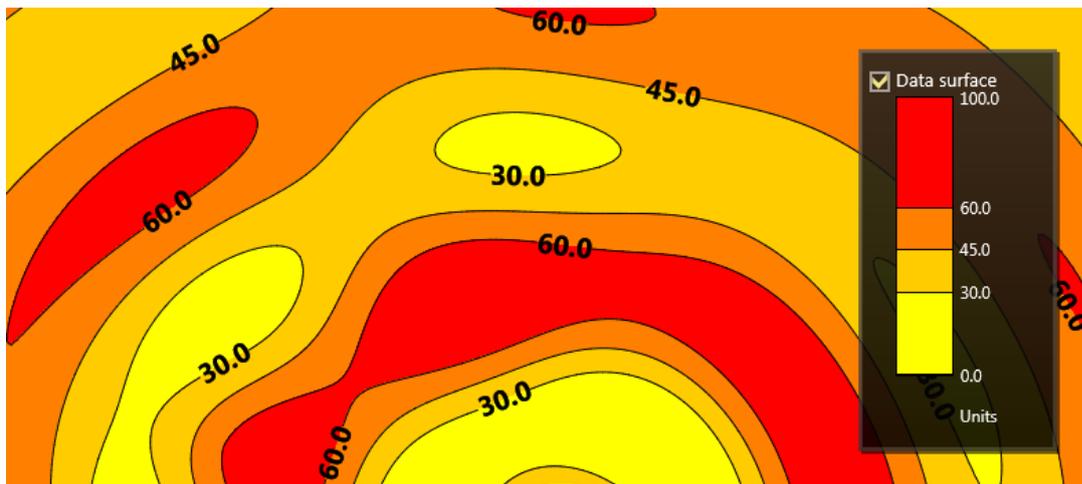


Figure 5-70. ContourLineLabels.Visible = true.

Use *LabelsNumberFormat* to set the number of decimals, or other custom formatting string.

## 5.16 IntensityMeshSeries

*IntensityMeshSeries* is almost similar to *IntensityGridSeries*. The biggest difference is that series nodes can be positioned arbitrarily in X-Y space. The series does not have to be rectangular. Wireframe lines can be set visible with *WireframeType* property and nodes can be shown by setting *ShowNodes* true.

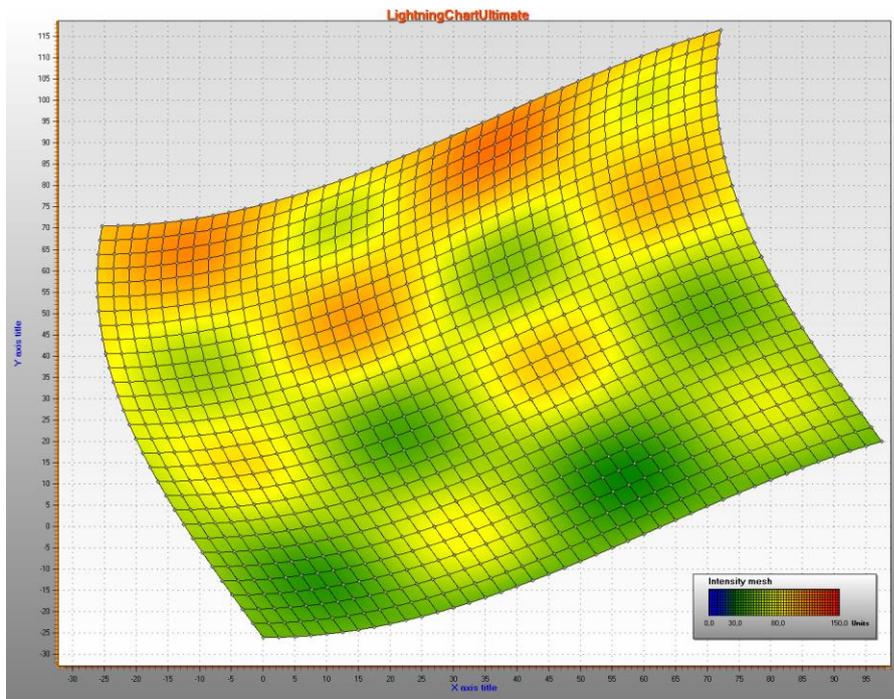


Figure 5-71. IntensityMeshSeries with freely positioned X and Y values for each node. WireframeType = Wireframe and ShowNodes = true.

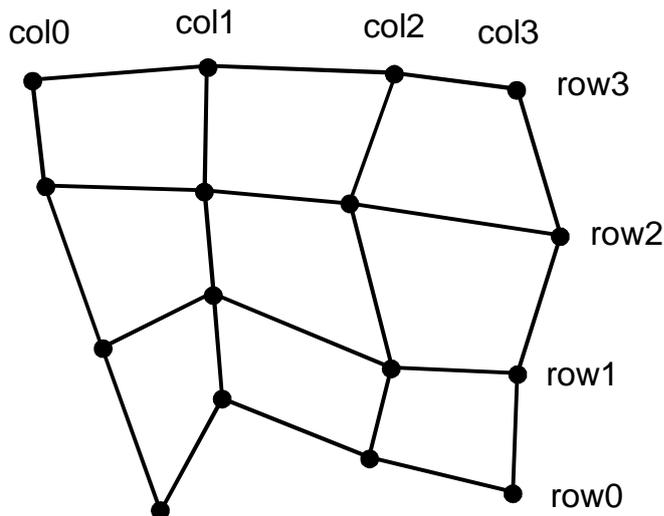


Figure 5-72. Intensity mesh nodes. SizeX = 4, SizeY = 4.

### 5.16.1 Setting intensity mesh data, when geometry changes

Follow these instructions, when the **X**, **Y** and **Value** fields are updated in the same time.

- Set **SizeX** and **SizeY** properties to give the mesh a size as columns and rows.
- Set X, Y and Y values for all nodes:

#### Method, with Data array index

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++)  
    {  
        meshSeries.Data[nodeIndexX, nodeIndexY].X = xValue;  
        meshSeries.Data[nodeIndexX, nodeIndexY].Y = yValue;  
        meshSeries.Data[nodeIndexX, nodeIndexY].Value = value;  
    }  
}  
meshSeries.InvalidateData(); //Notify new values are ready to refresh
```

#### Alternative method, usage of SetDataValue

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++)  
    {  
        meshSeries.SetDataValue(nodeIndexX, nodeIndexY,  
            xValue,  
            yValue,  
            value,  
            Color.Green); //Source point colors are not used in this  
                           example, so use any color here  
    }  
}  
meshSeries.InvalidateData(); //Notify new values are ready to refresh
```

## 5.16.2 Setting intensity mesh data, when geometry does not change

Follow these instructions, when only the **Value** fields of **Data** array **IntensityPoint** structures are updated. This is the performance optimized way for updating data for example in thermal imaging or environmental data monitoring solutions, where **X** and **Y** values of each node stay at same location.

### 5.16.2.1 Creating the series and geometry

- Set **Optimization** to **DynamicValuesData**
- Set **SizeX** and **SizeY** properties to give the mesh a size as columns and rows.
- Set X, Y and Y values for all nodes:

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++)  
    {  
        meshSeries.Data[nodeIndexX, nodeIndexY].X = xValue;  
        meshSeries.Data[nodeIndexX, nodeIndexY].Y = yValue;  
        meshSeries.Data[nodeIndexX, nodeIndexY].Value = value;  
    }  
}  
meshSeries.InvalidateData(); //Rebuild geometry from nodes and repaint
```

### 5.16.2.2 Updating the values periodically

- Set only values for all nodes:

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++)  
    {  
        meshSeries.Data[nodeIndexX, nodeIndexY].Value = value;  
    }  
}  
meshSeries.InvalidateValuesDataOnly(); //Only data values are updated
```

## 5.17 Bands

Actually, bands can be considered as series, or not. All user interface actions are like with other series, but one band series contains only one band. Band is a vertical or horizontal area, which reaches from a margin across to another. A band can be bound to a Y axis or X axis, use **Binding** property to control that. If the band is bound to Y axis, you must also set the **AssignYAxisIndex** property. If the series is bound to X axis, you can ignore **AssignYAxisIndex** property, or set it as unassigned (-1).

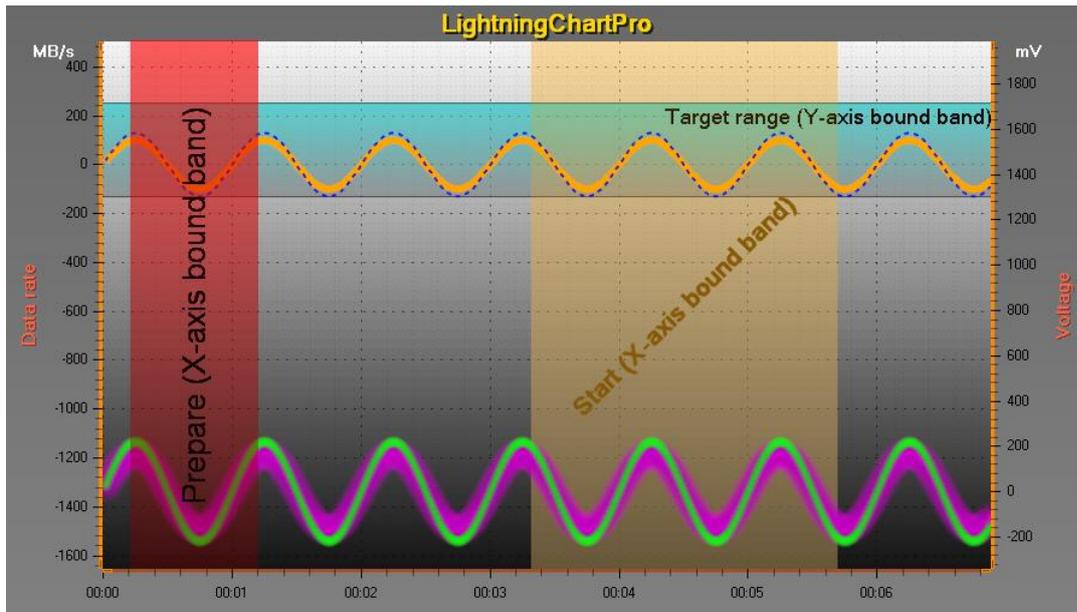


Figure 5-73. A couple of bands with line series

If you want the band to be behind the line and bar series, set **Behind** property true. Band edges are set by **ValueBegin** and **ValueEnd** properties, which are values in axis bound. Band can be dragged to another location with mouse. Resize the band by dragging it from the edge, which updates the dragged edge value, **ValueBegin** or **ValueEnd**.

## 5.18 Constant lines

Like bands, constant lines can be considered as series, or not. Constant lines are bound to Y axis, and it represents one horizontal line, ranging from graph left edge to right edge. Set the level in **Value** property. Constant lines can be vertically moved by dragging with mouse. By setting **Behind** property true, the constant line is drawn behind line and bar series, otherwise it is drawn on front of them.

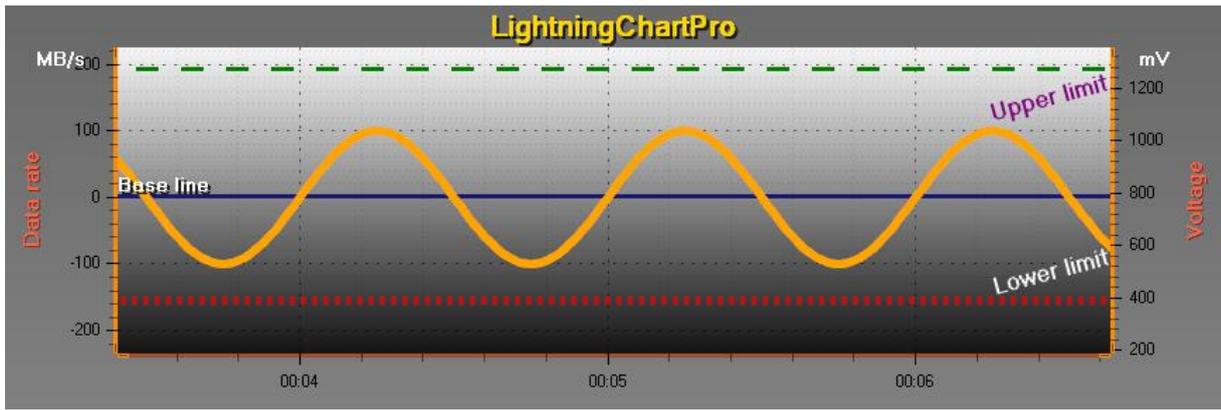


Figure 5-74. Some constant lines around a sine line series.

## 5.19 Annotations

Annotations let you display mouse-interactive text labels or graphics anywhere in the chart area. The annotations can be moved around by mouse, resized, rotated, their target and location can be changed etc. Alternatively, they can be controlled by code. The annotations are great also when custom graphics must be rendered on the screen, as they can be rendered in different styles and shapes. Create **AnnotationXY** objects in **ViewXY.Annotations** collection.

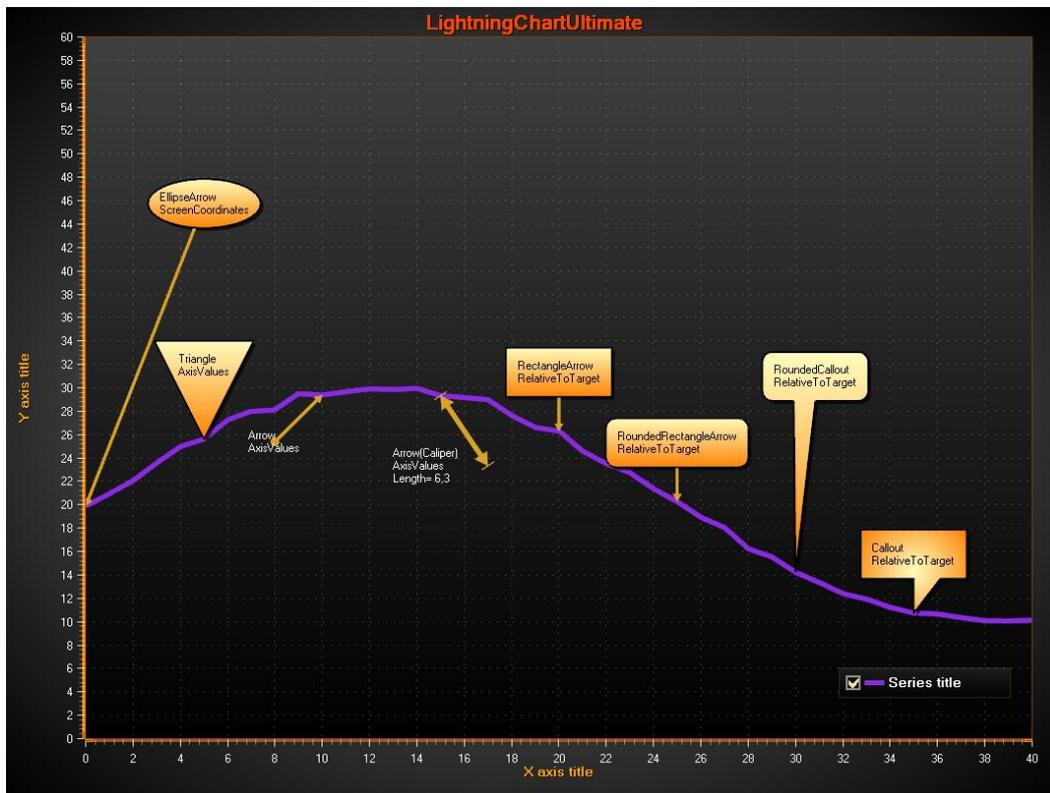


Figure 5-75. AnnotationXY objects with various styles, placed around a line series. Use *Style* property to select the shape.

By clicking the annotation, it goes into mouse-interactive edit state, where you can adjust where the annotation is located, where the arrow points to, resize it, and rotate it.

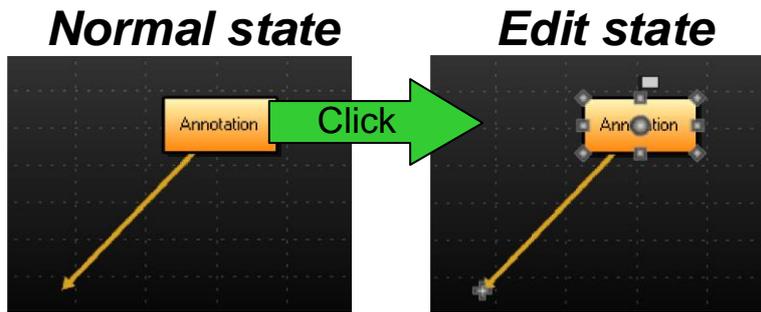


Figure 5-76. Click the annotation to enter the editing state. Click another annotation or graph background to leave the edit state.

### 5.19.1 Controlling target and location

**Target** is the ending point of the arrow, the point that the arrow or callout tip points to. Target can be set in axis values or in screen coordinates. Use **TargetCoordinateSystem** to select between **AxisValues** or **ScreenCoordinates**. When **AxisValues** is selected, **TargetAxisValues** property sets where the arrow line points to (end of the arrow line). Use **TargetScreenCoords** to set it in screen coordinates instead.

**Location** is the starting point of the arrow, and it can be set by screen coordinates, axis values, or as relative offset from Target. Use **LocationCoordinateSystem** to select, and **LocationScreenCoords**, **LocationAxisValues** or **LocationRelativeOffset** to control the location by the selected method. **Location** is also the center point of text area rotation.

**Anchor** property controls how the text area is placed at **Location**. By setting **Anchor.X** = 0.5 and **Anchor.Y** = 0.5, the beginning of the arrow is in the middle. When setting **Anchor.X** 0.1 and **Anchor.Y** = 0.25, arrow start is near the upper left corner as the following figure illustrates:

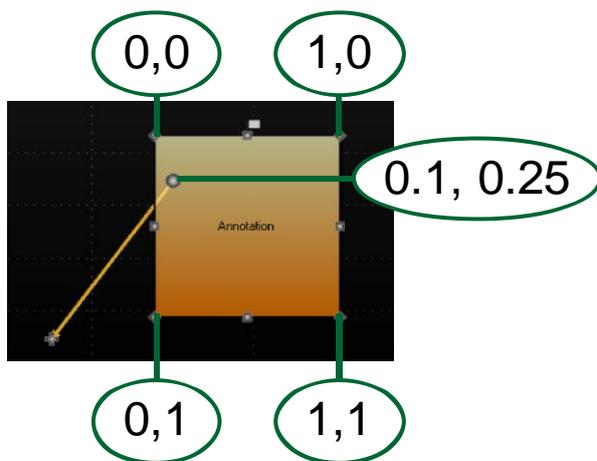


Figure 5-77. Anchor values explained. Current **Anchor.X** = 0.1 and **Anchor.Y** = 0.25. When the anchor values are between 0...1, the arrow start point is inside the text area.

### 5.19.2 Using mouse to move, rotate and resize

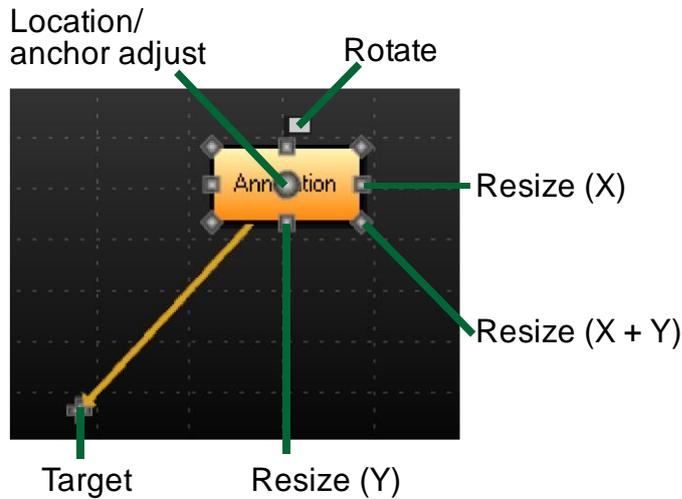


Figure 5-78. Annotation mouse-interactive nodes.

Drag from **Target** to move the end of the arrow. Drag from text area to set new **Location**. By dragging from round Location/anchor node, you can adjust the **Anchor** and **Location** at same time, keeping the text box in the same place.

By holding **Shift** key down while dragging from X or Y resize node, a symmetrical operation is used, both sides are adjusted at same time. By holding **Shift** key down while dragging from corner resize node (X+Y), it makes an aspect ratio keeping resize. In rotate operation, **Shift** key snaps the rotate angle to nearest multiple of 15 degrees.

### 5.19.3 Adjusting appearance

Select the annotation shape by setting **Style** property. The options are: **Rectangle**, **RectangleArrow**, **RoundedRectangle**, **RoundedRectangleArrow**, **Arrow**, **Callout**, **RoundedCallout**, **Ellipse**, **EllipseArrow**, **Triangle** and **TriangleArrow**.

With styles with arrow, use **ArrowLineStyle**, **ArrowStyleBegin** and **ArrowStyleEnd** to control the arrow looks. As arrow end styles, there are options: **None**, **Square**, **Arrow**, **Circle** and **Caliper**.

Use **Fill** to modify the fill of the annotation. From **NibStyle**, you can change the appearance of the editing state mouse-interactive nodes. **TextStyle** controls the font settings and text alignment inside the text area. **BorderLineStyle** and **CornerRoundRadius** control the border line appearance.

### 5.19.4 Size settings

**Sizing** property controls how the annotation text box is to be sized:

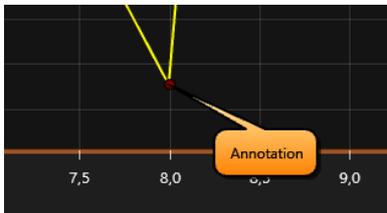
- **Automatic** adjusts the size by the contents, and leaves **AutoSizePadding** space to the borders.
- **AxisValuesBoundaries** allows the size of the annotation to be set by axis values. Use **AxisValuesBoundaries.XMin**, **XMax**, **YMin** and **YMax** for defining them.
- **ScreenCoordinates** enables you to set size by the screen coordinates. Use **SizeScreenCoords.Height** and **Width**.

### 5.19.5 Keeping text area visible

When **KeepVisible** is enabled, the annotation text area is forced inside the graph. The annotation won't move outside the graph when moving it by mouse or code. When you pan the graph view or adjust axes, the annotations are repositioned to show inside the graph.

### 5.19.6 Displaying annotation over axes

By setting **RenderBehindAxes = True**, annotation is shown over axes. All clipping and Z ordering features are not feasible in that case.



### 5.19.7 Clipping inside graph

When **ClipInsideGraph** is enabled, the annotation is clipped inside the graph. When it's disabled, the annotation is rendered also in the margins area of the chart.

By enabling **ClipWhenSweeping**, the annotation doesn't show up in the sweeping gap area.

Ensure **RenderBehindAxis** is set to **True** for **ClipInsideGraph** property to apply.

### 5.19.8 Controlling the Z order

By setting leaving **Behind** to its default value, **False**, the annotation appears on top of series. By setting it **True**, it is rendered before these series.

The annotations appear in the order they exist in Annotations list, while keeping the **Behind** filter as a master controller. Annotations Z order can be changed quickly by using **ChangeOrder** method of annotation, for example in a mouse event handler. The options for order change are:

- **BringToFront** brings the annotation to topmost
- **SendToBack** sends to back
- **MoveBack** moves one step backwards
- **MoveFront** moves one step forwards

### 5.19.9 LayerGrouping performance optimization

When having hundreds of annotations with visible text, the delay of text rendering starts to play a significant role. By default, the texts are rendered Z ordered, keeping the text firmly within an annotation.

The performance can be improved by setting **LayerGrouping = True**, and the chart will use only two flat annotation text layers. One for annotations with **Behind** set to **True**, and other for annotations with **Behind** set to **False**. It greatly improves performance. In the other hand, the text will be rendered wrong if there's other annotations overlapping others.

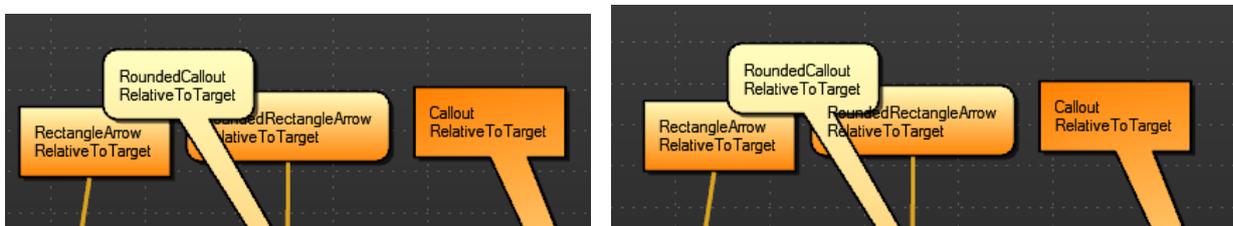


Figure 5-79 On the left **LayerGrouping = False**. On the right, **LayerGrouping = True**, Z order of texts is lost.

When using **Style = Arrow** or by setting the annotation fill not visible, the restriction of Z order typically doesn't show up.

### 5.19.10 Converting between axis values and screen coordinates

In some cases, you may want **Location** or **Target** to be defined in mixed configuration. X in screen coordinates and Y in axis values, or vice versa. Axes have **ValueToCoord** method for converting axis value to screen coordinate, and **CoordToValue** to convert screen coordinate to axis value.

## 5.20 Legend box

Starting from v.8, ViewXY supports multiple legend boxes in the same graph. Insert these legend boxes in **ViewXY.LegendBoxes** collection.

Misc	
AlignmentInSegmentGap	Near
AlignmentInVerticalMargin	Center
AllowMouseResize	True
AutoSize	True
BorderColor	<input type="color" value="#402555"/> 40, 255, 255, 255
BorderWidth	1
Categorization	None
CategoryColor	<input type="color" value="White"/> White
CategoryFont	<b>Segoe UI, 10pt, style=Bold</b>
CheckBoxColor	<input type="color" value="#140255"/> 140, 255, 255, 255
CheckMarkColor	<input type="color" value="Khaki"/> Khaki
Fill	
Height	<b>30</b>
HighlightSeriesOnTitle	True
HighlightSeries TitleColor	<input type="color" value="Yellow"/> Yellow
IntensityScales	
Layout	Horizontal
MouseHighlight	Simple
MouseInteraction	True
MoveByMouse	True
MoveFromSeries Title	True
Offset	
Position	<b>Segment BottomRight</b>
ScrollBarVisibility	Both
SegmentIndex	<b>0</b>
Series TitleColor	<input type="color" value="White"/> White
Series TitleFont	<b>Segoe UI, 10pt</b>
Shadow	
ShowCheckboxes	True
ShowIcons	True
UnitsColor	<input type="color" value="White"/> White
UnitsFont	<b>Segoe UI, 9pt</b>
UseSeries Titles Colors	False
ValueLabelColor	<input type="color" value="White"/> White
ValueLabelFont	<b>Segoe UI, 9pt</b>
Visible	True
Width	<b>303</b>

Figure 5-80. Extensive LegendBoxXY property tree.

### 5.20.1 Hiding / showing a series from legend box

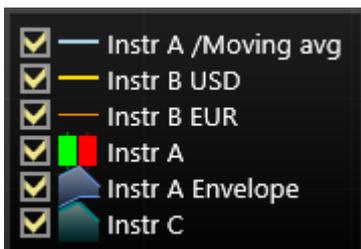


Figure 5-81. Legend box shows series titles and icons. You can hide a series by deselecting the series checkbox.

### 5.20.2 Preventing a series from listing itself in the legend box

If you don't want to list a specific series in the legend box, you can set `series.ShowInLegendBox = False`, for that specific series.

### 5.20.3 Selecting in which legend box to show a specific series

Use `series.LegendBoxIndex` to select the preferred legend box. A series can appear only in one legend box. Default index is 0 for all series.

### 5.20.4 Hiding check boxes

To hide the check boxes, set `ShowCheckBoxes = False`.

### 5.20.5 Hiding icons

To hide the icons, set `ShowIcons = False`.

### 5.20.6 Hiding intensity series palette scales

To hide the palette scale, set `IntensityScales.Visible = False`.



Figure 5-82. LegendBox.IntensityScales.Visible = 'False' in the picture in the bottom.

## 5.20.7 Controlling positions

The legend boxes can be aligned left/top/right/bottom side of the graph segments, or margins. Control the position with **Position** property. Position options are: **TopCenter**, **TopLeft**, **TopRight**, **LeftCenter**, **RightCenter**, **BottomLeft**, **BottomCenter**, **BottomRight**, **Manual**.

And for segment-based controlling: **SegmentTopLeft**, **SegmentTopCenter**, **SegmentTopRight**, **SegmentBottomLeft**, **SegmentBottomCenter**, **SegmentBottomRight**, **SegmentLeftMarginCenter**, **SegmentRightMarginCenter**.

**Offset** property shifts the position by given amount *from the position determined by Position* property.

## 5.20.8 Allocating space for legend boxes between graph segments

When setting **ViewXY.AutoSpaceLegendBoxes = True**, additional space between segments will be allocated to fit the legend boxes in them.

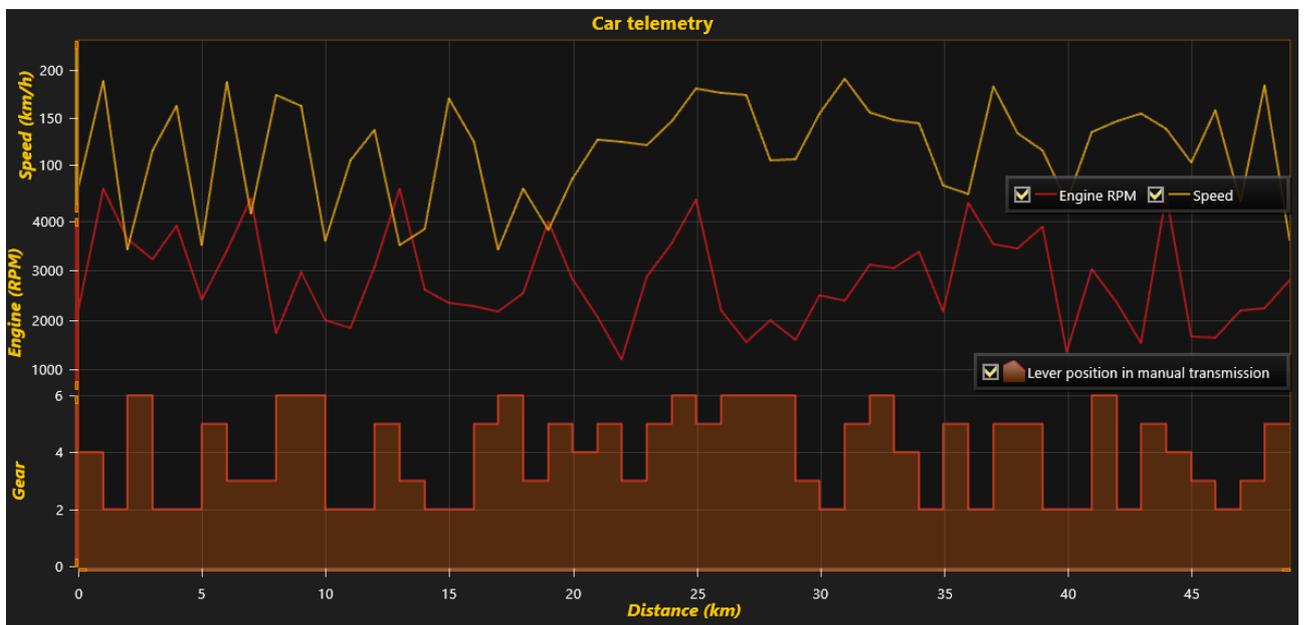


Figure 5-83. Position = SegmentBottomRight. AutoSpaceLegendBoxes = False.

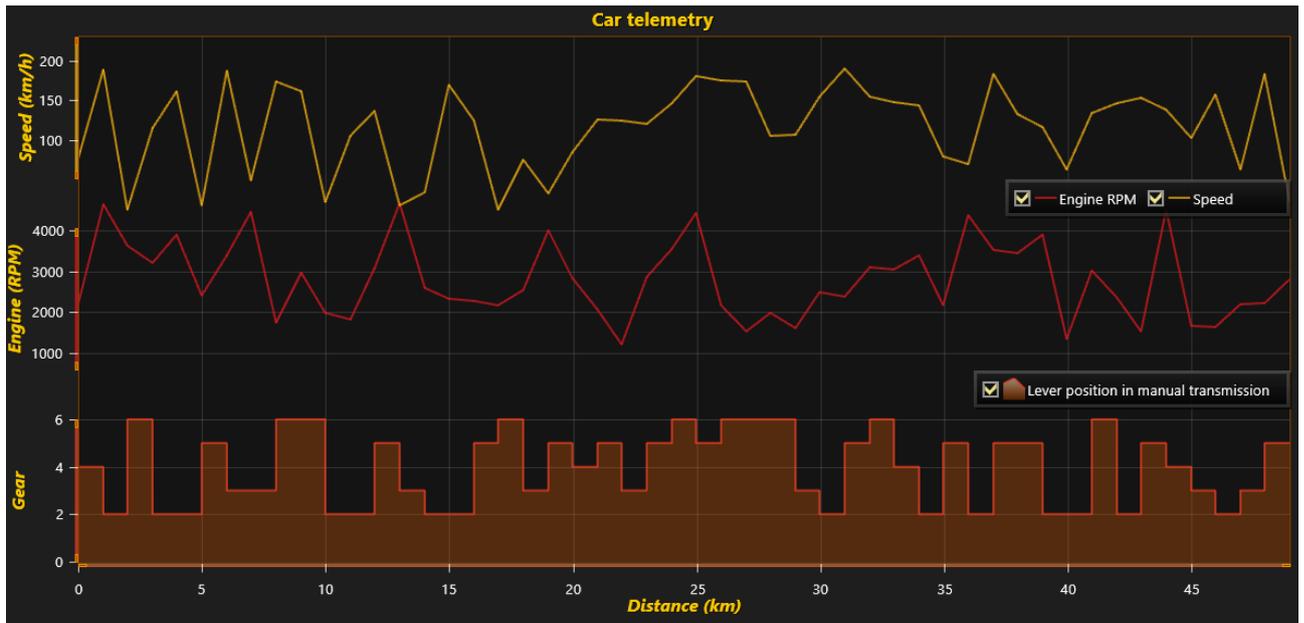


Figure 5-84. Position = SegmentBottomRight. AutoSpaceLegendBoxes = True.

Note that also *ViewXY.AxisLayout.SegmentsGap* is allocated between segments.

### 5.20.9 Alignment of legend boxes in segment gap

To align legend box vertically near the specified segment, set *AlignmentInSegmentGap = Near*. To align it vertically to center of the gap between segments, set *AlignmentInSegmentGap = Center*.

### 5.20.10 Selecting in which graph segment to show a legend box

Use *SegmentIndex* to control in which segment to show the legend box. It applies only to segment-based *Position* options.

### 5.20.11 Horizontal alignment of several legend boxes sharing the same margin

*AlignmentInVerticalMargin* property has *Left/Center/Right* options. The property controls horizontal positioning of legend boxes set to same vertical margin.

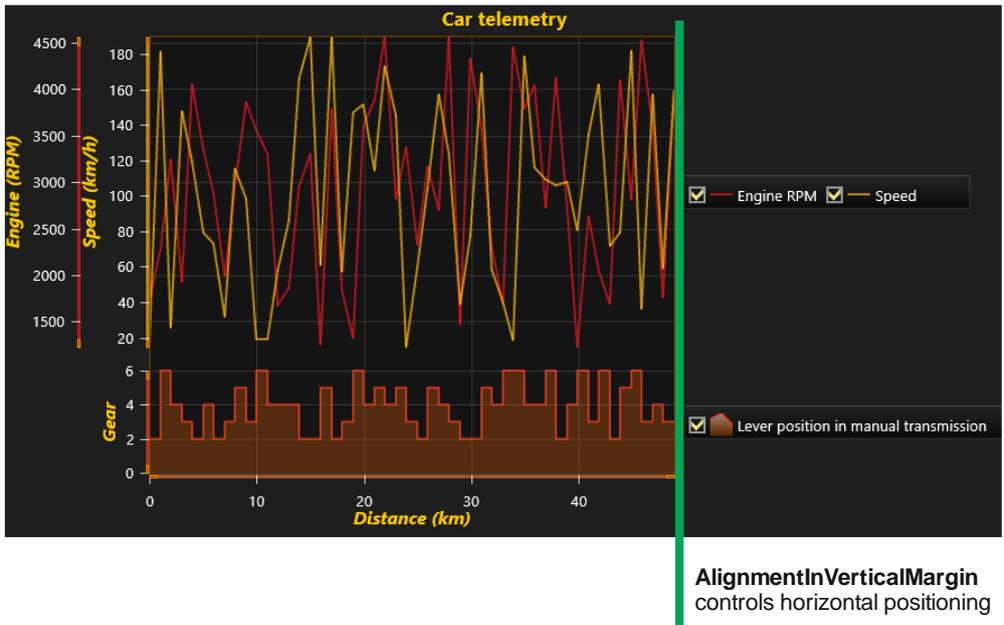
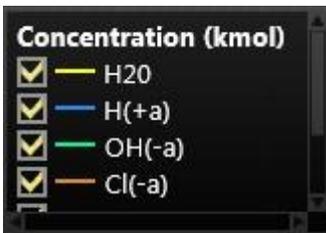


Figure 5-85. AlignmentInVerticalMargin = Left set for both Legend boxes.

## 5.20.12 Resizing and moving legend boxes

The legend boxes support resizing and scroll bars. Grab from the edge to resize it.



Note that when moving or resizing legend box, its **Position** is set to **Manual**, and **Offset** property is updated to reflect the new position.

Automatic legend box alignment is disabled until setting **Position** back to option other than '**Manual**'.

## 5.21 Zooming and panning

Use **ZoomPanOptions** properties to control the zooming and panning settings.

ZoomPanOptions	
AltEnabled	True
<ul style="list-style-type: none"> <li>AspectRatioOptions           <ul style="list-style-type: none"> <li>AspectRatio</li> <li>ManualAspectRatioWH</li> <li>XAxisIndex</li> <li>YAxisIndex</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>Off</li> <li><b>2</b></li> <li>0</li> <li>0</li> </ul>
<ul style="list-style-type: none"> <li>AutoYFit           <ul style="list-style-type: none"> <li>Enabled</li> <li>MarginPercents</li> <li>TargetAllYAxes</li> <li>Thorough</li> <li>UpdateInterval</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>False</li> <li>5</li> <li><b>False</b></li> <li>True</li> <li>100</li> </ul>
AxisMouseWheelAction	Pan
CtrlEnabled	True
IgnoreZerosInLogFit	False
LeftMouseButtonAction	Zoom
MousePanThreshold	5
MouseWheelZooming	HorizontalAndVertical
MultiTouchPanEnabled	True
MultiTouchSensitivity	2
MultiTouchZoomDirection	Rails
MultiTouchZoomEnabled	True
PanDirection	Both
RectangleZoomAboutOrigin	False
RectangleZoomDirection	Both
<ul style="list-style-type: none"> <li>RectangleZoomingThreshold           <ul style="list-style-type: none"> <li>X</li> <li>Y</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li><b>4</b></li> <li><b>4</b></li> </ul>
RectangleZoomLimitInsideGraph	False
RectangleZoomUnitsLinkYAxes	False
RightMouseButtonAction	Pan
RightToLeftZoomAction	FitView
ShiftEnabled	True
ViewFitYMarginPixels	0
ZoomFactor	2
ZoomOutRectFill	
ZoomOutRectLine	
ZoomRectFill	
ZoomRectLine	

Figure 5-86. ZoomPanOptions properties and sub-properties.

Zooming and panning can be performed by left or right mouse button, they are configurable. Zooming can be also performed with mouse wheel.

### 5.21.1 Zooming with touch screen

Set two fingers on the chart, and pinch your fingers closer to zoom out, or away to zoom in.

The chart tries to detect if you are trying to do a horizontal or vertical zooming, or both at same time. This feature is called 'zooming with rails', which can be controlled by **MultiTouchZoomDirection** (*Free/XAxis/YAxis/Rails*).

By pinching/spreading fingers above an X or Y axis or labels of it, the zooming applies to that specific axis only.

Zooming with touch can be disabled by setting **MultiTouchZoomingEnabled** = false.

### 5.21.2 Panning with touch screen

Set two fingers on the screen and slide them at same pace to pan the view.

Some systems support panning with inertia, so you can "throw" your fingers of the screen, and the view keeps panning and finally slows down until stopped.

By setting a finger above an X or Y axis or labels of it, and sliding the finger, the panning applies to that specific axis only.

### 5.21.3 Left mouse button action

Set **LeftMouseButtonAction** to **Zoom**, to enable zooming with left mouse button. Set it to Pan to enable panning. To disable zoom and pan from left mouse button, set it to **None**.

### 5.21.4 Right mouse button action

Set **RightMouseButtonAction** to **Zoom**, to enable zooming with right mouse button. Set it to Pan to enable panning. To disable zoom and pan from right mouse button, set it to **None**.

### 5.21.5 RightToLeftZoomAction

**RightToLeftZoomAction** applies when **LeftMouseButtonAction** or **RightMouseButtonAction** is set to **Zoom**. RightToLeftZoomAction specifies what happens when mouse zooming is made from right to left (mouse X button down-coordinate > button up-coordinate).

The following selections are available:

**ZoomToFit**: Fits all Y axes and X axis so that all series data belonging to them is shown. By using **ViewFitYMarginPixels** with greater value than 0, the axes are scaled so that given space in pixels is reserved empty of data, in both Y axis minimum and maximum end.

**RectangleZoomIn**: Zooms in with rectangle, similarly than by zooming from left to right.

**ZoomOut**: Zooms out, by using **ZoomFactor**.

**RevertAxisRanges**: Sets axis values to specific values, in other words, restores the original view after it has been zoomed or axis ranges have been otherwise modified. In each axis, there's **RangeRevertEnabled** property, which controls if the axis range should be reverted. If it's enabled, **RangeRevertMinimum** and **RangeRevertMaximum** properties are applied to the axis when dragging mouse from right to the left, and the mouse button is released. See section 5.2.5.

**PopFromZoomStack**: Sets same axis ranges that were used when zooming in last time, i.e. goes back to previous zoom level.

### 5.21.6 Zooming with mouse button

#### 5.21.6.1 Zoom in/out by clicking

Use **ZoomFactor** property to control the how much closer/farther the zoom is applied. If you want to apply negative zoom effect, set value as inversed value (1/factor). The zoom is applied using mouse cursor position as a zoom center point.

##### X dimensional zoom:

With chart control focused, press Shift key down. Zoom X cursor appears. Click configured mouse button to zoom in, and the other button to zoom out.

##### Y dimensional zoom:

With chart control focused, press Ctrl key down. Zoom Y cursor appears. Click configured mouse button to zoom in, and the other button to zoom out. When using a stacked **YAxisLayout**, zooming applies to all graph segments (Y axes). By pressing Ctrl and Alt keys down, the Y dimensional zoom is applied only to graph segment the mouse was clicked over.

You can press both Shift and Ctrl(+Alt) keys down simultaneously, for applying zoom to both X and Y dimensions.

#### 5.21.6.2 Zoom in with rectangle

With configured mouse button, drag a rectangle around the area to be zoomed, from upper left corner to bottom right corner. Both X and Y dimensions effect. The dimensions are selected by **RectangleZoomDirection** property.

The zoom rectangle border and fill style can be modified by using **ZoomRectFill** and **ZoomRectLine** properties.

#### 5.21.6.3 Configuring zoom out rectangle

When RightToLeftZoomAction is set to **ZoomToFit**, **ZoomOut**, **RevertAxisRanges** or **PopFromZoomStack**, the zoom out rectangle appears when zooming. Configure its fill by **ZoomOutRecFill** and line style by **ZoomOutRectLine**.

#### 5.21.7 Zooming with mouse wheel

By enabling **MouseWheelZooming**, you can zoom in by scrolling the mouse wheel upwards, and zoom out by scrolling it downwards. The zoom center is the position of mouse cursor. Use **ZoomFactor** to adjust the mouse wheel zoom strength. By keeping Shift key pressed, the zoom is applied only for X dimension. By keeping Ctrl key pressed, the zoom applies only for Y dimension.

Note that zooming is not available when **ScrollMode** is set to **Sweeping**.

#### 5.21.8 Zooming and panning with mouse wheel, over axis

Use AxisMouseWheelAction to configure what happens when mouse wheel is applied over an axis.

**None**: Mouse wheel does nothing

**Zoom**: Zoom only the axis the mouse is over

**Pan**: Pan only the axis the mouse is over

***ZoomAll***: Zooms all X axes if mouse is over an X axis, or all Y axes if mouse is over a Y axis. **Note**, applies to other axes only when *YAxisLayout* = *Layered*.

***PanAll***: Pans all X axes if mouse is over an X axis, or all Y axes if mouse is over a Y axis. **Note**, applies to other axes only when *YAxisLayout* = *Layered*.

### 5.21.9 Panning with mouse button

Configure *LeftMouseButtonAction* or *RightMouseButtonAction* to *Pan* for panning to work. Drag graph area with configured mouse button pressed. To stop panning, release the button. Pan scrolls both X and Y axes by dragged amount, if *PanDirection* is *Both*. By setting *PanDirection Vertical*, it'll only target Y axes. *PanDirection Horizontal* targets X axes only. Use *MousePanThreshold* to give some tolerance in pixels before the panning starts to effect. It's very handy if you use *ContextMenuStrip* control assigned for the chart control, preventing it to open every time the panning stops.

### 5.21.10 Enabling/disabling Ctrl, Shift and Alt

Zoom operations support these modifier keys, and they are enabled by default, to disable these, set *AltEnabled* = *False*, *CtrlEnabled* = *False* or *ShiftEnabled* = *False*.

### 5.21.11 Zoom in/out with code

Use *ZoomByFactor(...)* method to zoom with a center point and a zoom factor. Use *Zoom(...)* method to zoom with rectangle. *ZoomToFit ()* method fits does "Zoom to fit" operation.

### 5.21.12 Zooming an axis by code

Set values to X or Y axis *Minimum* and *Maximum* properties. Use *SetRange(...)* to set them both at same time.

### 5.21.13 Rectangle zooming about a configurable origin

By enabling *RectangleZoomAboutOrigin*, the rectangle zooming in/out applies symmetrically about *ZoomOrigin* set in X axis and Y axis.



Figure 5-87. `ZoomPanOptions.RectangleZoomAboutOrigin` enabled. `ViewXY.XAxes[0].ZoomOrigin = 50` and `ViewXY.YAxes[0].ZoomOrigin = 50`.

#### 5.21.14 Linking Y axes zoom with same units

By enabling `RectangleZoomLinkYAxes`, all the Y axes having same `Units.Text` string, get the same Y axis range than the axis that is under rectangle zooming.



Figure 5-88. Stacked view with 5 Y axes. When rectangle zoom is applied over a graph segment, the Y axes of it get zoomed, and the new Y axis range is copied to all Y axes having same `Units.Text`.

### 5.21.15 Automatic Y fit

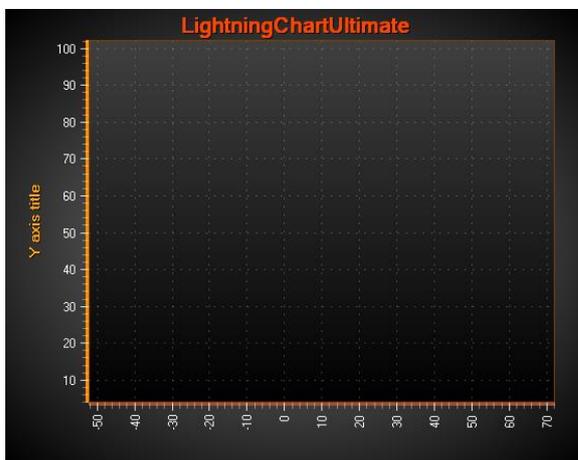
Use **AutoYFit** property to control the automatic Y axis adjustment. Automatic Y fit can be used to adjust the Y axis ranges to show all the data in the chart, in visible X axis range. Automatic Y fit is especially intended for real-time monitoring purposes. The fit is applied in time intervals, use **UpdateInterval** to set the interval in milliseconds. Use **MarginPercents** for getting some extra range for axes. By enabling **Through**, the fitting analysis is made for all data, but may cause some overhead in performance critical systems. By disabling it, only a small piece of latest data is used for fitting analysis, and may cause improper behavior in certain applications.

**Note!** **AxisY** class also **Fit()** methods for fitting in Y dimension.

### 5.21.16 Aspect ratio

**AspectRatioOptions.AspectRatio** controls the X/Y (or longitude / latitude in maps) ratio.

Set it to **Off** to allow X and Y axis range setting individually. By setting the aspect ratio to **Manual**, use the **ManualAspectRatioWH** property to set the preferred ratio. The zooming operations will follow aspect ratio setting.



When **AspectRatio** is other than **Off**, axis scaling nibs are not available.

For maps (see 5.24), **AspectRatio = AutoLatitude** is a very useful option. **AutoLatitude** changes the aspect ratio dynamically, as you view the map in different locations. The aspect ratio is determined by the center point of the view.

### 5.21.17 Excluding specific X or Y axes from zooming and panning operations

- To exclude specific X or Y axes from Zooming operations, set  
*axis.ZoomingEnabled = False*
- To exclude specific X or Y axes from Panning operations, set  
*axis.PanningEnabled = False*

## 5.22 DataBreaking by NaN or other value

DataBreaking	
Enabled	True
Value	NaN

Figure 5-89 DataBreaking options in series that support it.

These series types support data breaking:

- PointLineSeries
- FreeformPointLineSeries
- SampleDataSeries
- AreaSeries
- HighLowSeries
- PointLineSeries3D

LightningChart skips rendering of the data points that match with specified breaking Value. All other values it renders normally.

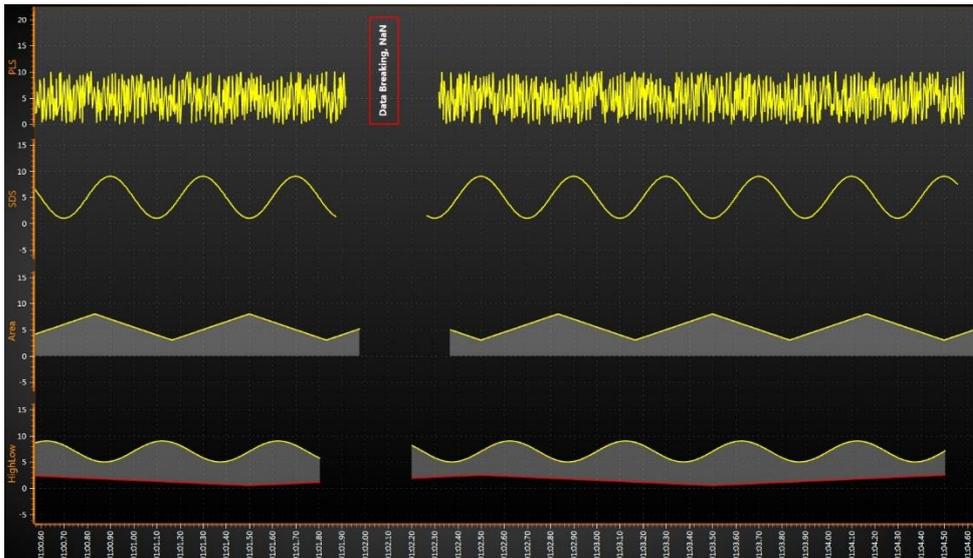
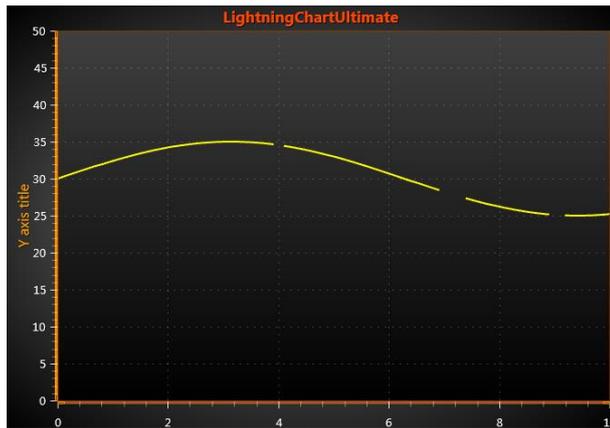


Figure 5-90. DataBreaking in use for PointLineSeries, SampleDataSeries, AreaSeries and HighLowSeries.

**NOTE!** When `DataBreaking.Enabled = True`, it will cause significant extra overhead, and not recommended for solutions needing very high real-time data rates. Consider using `ClipAreas`, see section 5.23.

For example, using NaN to break PointLineSeries data:



Code:

```

int pointCount = 101;
double[] xValues = new double[pointCount];
double[] yValues = new double[pointCount];

for (int point = 0; point < pointCount; point++)
{
    xValues[point] = (double)point * interval;
    yValues[point] = 30.0 + 5.0 *Math.Sin((double)point / 20.0);
}

//Add some NaN values in Y array to mark break points
yValues[40] = double.NaN;
yValues[70] = double.NaN;
yValues[71] = double.NaN;
yValues[72] = double.NaN;
yValues[73] = double.NaN;
yValues[90] = double.NaN;
yValues[91] = double.NaN;

//Add new series with DataBreaking Enabled
PointLineSeries pls = new PointLineSeries(_chart.ViewXY,
    _chart.ViewXY.XAxes[0], _chart.ViewXY.YAxes[0]);
pls.DataBreaking.Enabled = true;

// set data gap defining value (default = NaN)
pls.DataBreaking.Value = double.NaN;

SeriesPoint[] points = new SeriesPoint[pointCount];
for (int point = 0; point < pointCount; point++)
{
    points[point].X = xValues[point];
    points[point].Y = yValues[point];
}

//Assign the data for the point line series
pls.Points = points;

//Add the created point line series into PointLineSeries list
_chart.ViewXY.PointLineSeries.Add(pls);

```



## 5.24 Maps

Use **Maps** property and its sub-properties to show geographic maps. LightningChart maps come in two different categories: **vector maps** and **tile maps**. The maps are shown in so called *equiarectangular* projection.

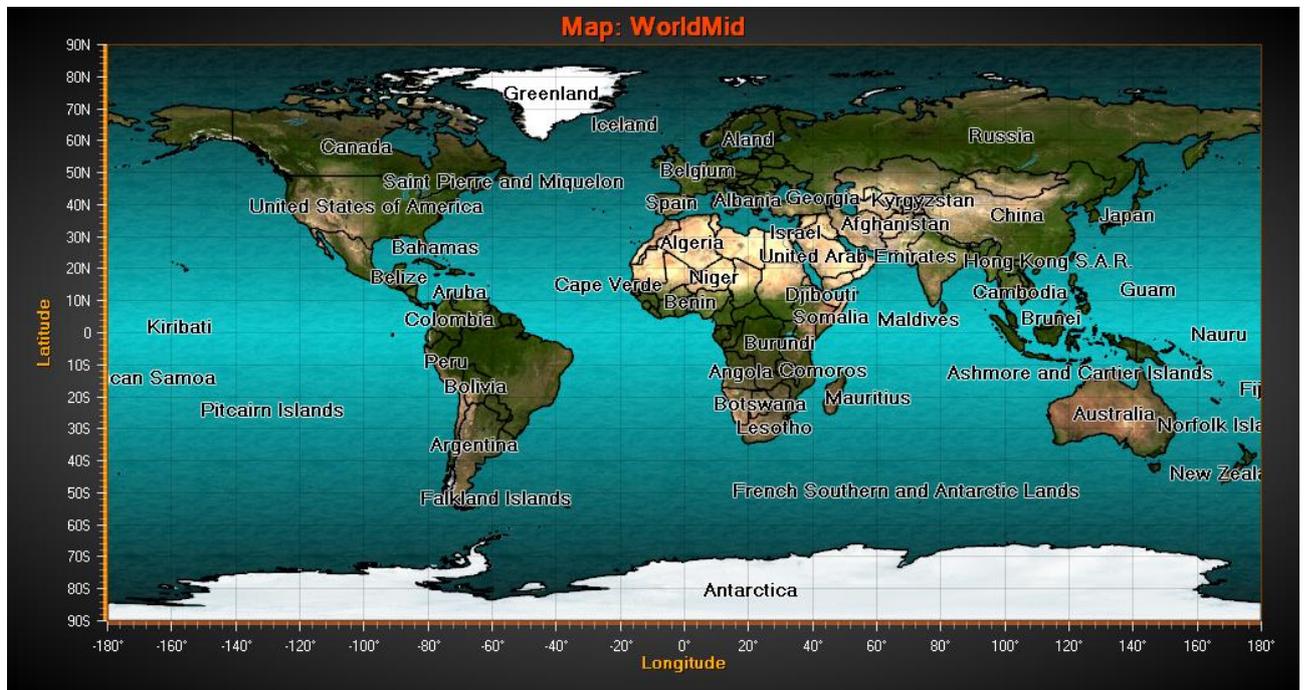


Figure 5-92. Equiarectangular projection of the world. X range is from -180 to 180 degrees (180W to 180E), Y range from -90 to 90 degrees (90S to 90N). Polar areas get greatly stretched in this projection.

This projection allows using LightningChart's series types and other objects that are practically all bound to X and Y axes, same time with the maps.

## 5.25 Vector maps

The geographic vector data is stored in LightningChart map files, with **.md** extension. LightningChart is delivered with set of map files.

The X Axis is used for Longitude, and the Y axis for latitude. See section 5.2.3 for showing map coordinate axes. The map coordinates are decimal degrees, with latitude origin at equator and longitude origin at Greenwich, U.K.

[-] Maps	
Backgrounds	<b>(Collection)</b>
[-] CityOptions	
Description	Map of world in mid resolution.
FileName	<b>WorldMid</b>
[-] LakeOptions	
[-] LandOptions	
[-] Layers	MapLayer[] Array
MouseHighlight	Simple
MouseInteraction	True
MouseOverMapItemLayer	1
Names	(Collection)
Optimization	<b>None</b>
[-] OtherOptions	
OverlapLabels	False
Path	<b>..\..\Maps</b>
RenderIntensitySeriesBeforeLayerIndex	-1
[-] RiverOptions	
[-] RoadOptions	
SimpleHighlightColor	 <b>100, 0, 255, 0</b>
TileCacheFolder	<b>c:\temp\map_cache</b>
TileLayers	<b>(Collection)</b>
Type	<b>WorldMid</b>
XAxisIndex	0
YAxisIndex	0

Figure 5-93. Maps properties and sub-properties. The whole tree is for vector maps, except TileLayers collection and TileCacheFolder, which is for tile maps.

### 5.25.1 Selecting active map

Set the directory names to the **Path** property, where the map files exist. The active map can be selected with **Type** property, for maps delivered with LightningChart. To use your own map file, set the **FileName** property.

If you don't want to use any map, set **Type** to **Off**.

Off  
AustraliaMid  
CanadaUSASatesMid  
EuropeLow  
EuropeMid  
EuropeHigh  
Other  
USALakesRiversMid  
USALakesRiversHigh  
USASatesLakesRiversMid  
USASatesLakesRiversHigh  
USASatesLakesRiversRoadsMid  
WorldLow  
WorldMid  
WorldHigh  
WorldLakesRiversLow  
WorldLakesRiversMid  
NorthAmericaLow  
NorthAmericaMid  
NorthAmericaHigh

**Figure 5-94. Map Type options.** The maps delivered with LightningChart are shown. The type name postfix tells a rough detail level of the map.

The maps of LightningChart are made in very high detail level, in general. For real-time monitoring solutions it is important to select a map giving proper detail and performance level.

### 5.25.2 Aspect ratio

**ViewXY.ZoomPanOptions.AspectRatioOptions.AspectRatio** controls the X/Y (or longitude / latitude) ratio.

Set it to **Off** to allow X and Y axis range setting individually, by stretching the map. **AutoLatitude** changes the aspect ratio dynamically, as you view the map in different locations. The aspect ratio is determined by the center point of the view. By setting the aspect ratio to **Manual**, use the **ManualAspectRatioWH** property to set the preferred ratio.

### 5.25.3 Layers and their appearance settings

Each map file can contain several layers. For example, layers for land regions, lakes, rivers, roads and cities. The layers and their data are accessible from **Layers** array property.

Layers	MapLayer[] Array
[0]	Arction.LightningChartUltimate.Maps.Poir
Color	
Items	City[] Array
Name	<b>Cities</b>
Priority	<b>0</b>
Type	City
Visible	<b>True</b>
[1]	Arction.LightningChartUltimate.Maps.Reg
[2]	Arction.LightningChartUltimate.Maps.Reg
BorderDrawStyle	<b>Options</b>
Items	Region[] Array
Name	<b>Lakes</b>
Priority	<b>2</b>
RegionDrawStyle	<b>Options</b>
Type	Lake
Visible	<b>True</b>
[3]	Arction.LightningChartUltimate.Maps.Line
AutoAdjustLineWidth	<b>True</b>
Items	Line[] Array
LineDrawStyle	<b>Options</b>
LineWidthCoeff	<b>1</b>
Name	<b>Rivers2</b>
Priority	<b>3</b>
Type	River
Visible	<b>True</b>

Figure 5-95. Map layer details opened in Properties editor.

Each layer has a specific type. The layer appearance options can be changed with corresponding options property. Use **LandOptions** for modifying the appearance of land regions, **LakeOptions** for lakes, **RiverOptions** for rivers, **RoadOptions** for roads, **CityOptions** for cities, and **OtherOptions** for unspecified layer types.

LandOptions	Arction.LightningChartUltimate.h
AntialiasFill	<b>False</b>
Fill	Arction.LightningChartUltimate.F
Bitmap	Arction.LightningChartUltimate.B
Color	<b>OldLace</b>
GradientColor	<b>Moccasin</b>
GradientDirection	<b>270</b>
GradientFill	<b>Linear</b>
Style	<b>ColorOnly</b>
FillVisible	<b>True</b>
LabelStyle	Arction.LightningChartUltimate.h
Angle	<b>0</b>
Color	<b>Black</b>
Font	<b>Microsoft Sans Serif; 12pt; style</b>
Shadow	Arction.LightningChartUltimate.T
Visible	<b>True</b>
LineStyle	Arction.LightningChartUltimate.L
AntiAliasing	<b>Normal</b>
Color	<b>DimGray</b>
Pattern	<b>Solid</b>
PatternScale	<b>1</b>
Width	<b>1</b>
LineVisible	<b>True</b>



Figure 5-96. Default LandOptions, and corresponding view from Europe.

LandOptions	Arction.LightningChartUltimate.M
AntialiasFill	False
Fill	Arction.LightningChartUltimate.Fi
Bitmap	Arction.LightningChartUltimate.B
Color	SandyBrown
GradientColor	Black
GradientDirection	270
GradientFill	Radial
Style	ColorOnly
FillVisible	True
LabelStyle	Arction.LightningChartUltimate.M
Angle	45
Color	Black
Font	Arial Black; 12pt; style=Bold, Ital
Shadow	Arction.LightningChartUltimate.T
Visible	True
LineStyle	Arction.LightningChartUltimate.Li
AntiAliasing	Normal
Color	Black
Pattern	Solid
PatternScale	1
Width	3
LineVisible	True

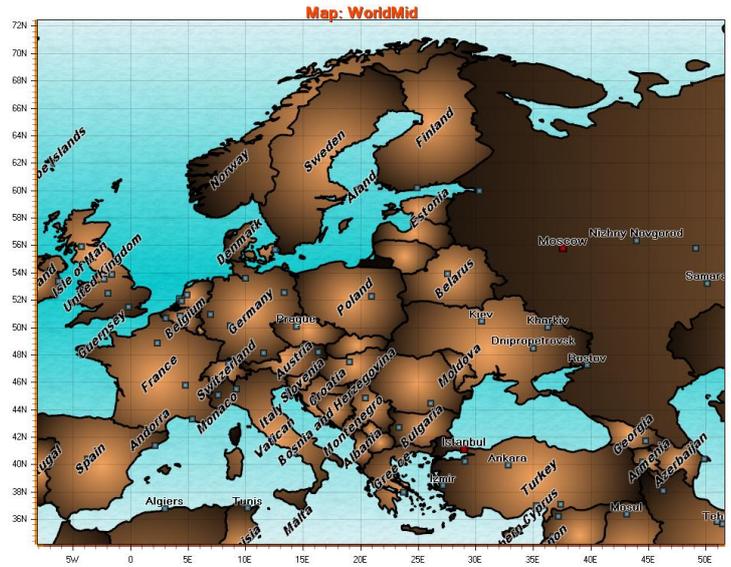


Figure 5-97. Modified LandOptions.

### 5.25.3.1 Setting individual fill and border style for each layer item

Each map element fill or border appearance can be set individually. Change **BorderDrawStyle** and **RegionDrawStyle** properties to **Individual**. Then, access the Items collection, and navigate to preferred item, and edit the **BorderLineStyle** and **Fill** properties. You may want to navigate the **Items** collection programmatically by **Name** property, here “Germany”.

Layers	MapLayer[] Array
[0]	Arction.LightningChartUltimate.Maps.Pc
[1]	Arction.LightningChartUltimate.Maps.Re
BorderDrawStyle	Individual
Items	Region[] Array
Name	Countries
Priority	1
RegionDrawStyle	Individual
Type	Land
Visible	True
[54]	Arction.LightningChartUltimate.Maps.R
[55]	Arction.LightningChartUltimate.Maps.R
[56]	Arction.LightningChartUltimate.Maps.R
[57]	Arction.LightningChartUltimate.Maps.R
BorderLineStyle	Arction.LightningChartUltimate.L
AntiAliasing	None
Color	Red
Pattern	Solid
PatternScale	1
Width	3
Center	Arction.LightningChartUltimate.PointFlc
Fill	Arction.LightningChartUltimate.F
Bitmap	Arction.LightningChartUltimate.B
Color	White
GradientColor	DarkViolet
GradientDirec	270
GradientFill	Linear
Style	ColorOnly
Name	Germany

Figure 5-98. Setting layer border line and region fill styles to Invidual and editing region in Items collection.

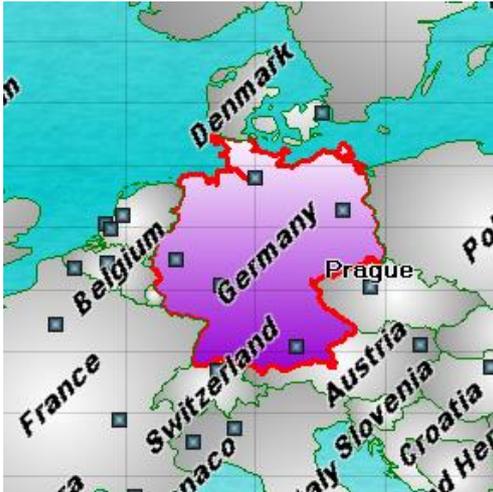


Figure 5-99. Germany region drawn with individual fill and border.

#### 5.25.4 Mouse interactivity

Enable ***MouseInteraction*** for all kind of interoperation with map regions and objects. Regions (land, lakes) and vector layers (rivers, roads) can be pointed with mouse. Once the mouse is over the object, it gets highlighted with ***SimpleHighlightColor***, when ***MouseHighlight*** is set to ***Simple***. When ***MouseHighlight*** is ***Blink***, the object will blink in light and dark colors. By setting ***MouseHighlight*** to ***None***, the object is not highlighted, but it still can be clicked and ***Maps.MouseDownOnMapItem*** event handler to be run.

Map objects may have associated data included, like population or other statistical data. Use ***MouseOverOnMapItem/MouseOverOffMapItem/MouseDownOnMapItem*** event handler to access the data. The data for a map item can be retrieved with ***GetInfo*** method, giving a dictionary of keys and values.

Here's an example of how to show all data in a list box. The item name is displayed in a different text box.

```

private void MouseDownOnMap(MouseDownOnMapItemEventArgs args)
{
    MapItem mapItem = args.MapItem;

    textBoxCountryName.Text =
        m_chart.ViewXY.Maps.Layers[args.Layer].Name
        + ": " + mapItem.Name;
    listBoxItemValues.Items.Clear();
    if (mapItem.GetInfo() != null)
    {
        Dictionary<string, string> dict = mapItem.GetInfo();
        Dictionary<string, string>.KeyCollection keys = dict.Keys;
        foreach (String key in keys)
        {
            String strValue;
            if (dict.TryGetValue(key, out strValue))
            {
                listBoxItemValues.Items.Add(key + ": " + strValue);
            }
        }
    }
}

```

### 5.25.5 Background photos

**By adding a *MapBackground* object in *Maps.Backgrounds* property** allows you to display bitmap images as the backgrounds of the maps. Satellite images or other raster images are available from several GIS data providers. The image can be set to ***Image*** property, and its latitude and longitude range can be set with ***LatitudeMin***, ***LatitudeMax***, ***LongitudeMin*** and ***LongitudeMax*** properties. Outside the ranges set, the image is not displayed.

To show the background through the map layers, it may be necessary to adjust the fill settings for each layer. Use transparent colors or colors with low alpha level.

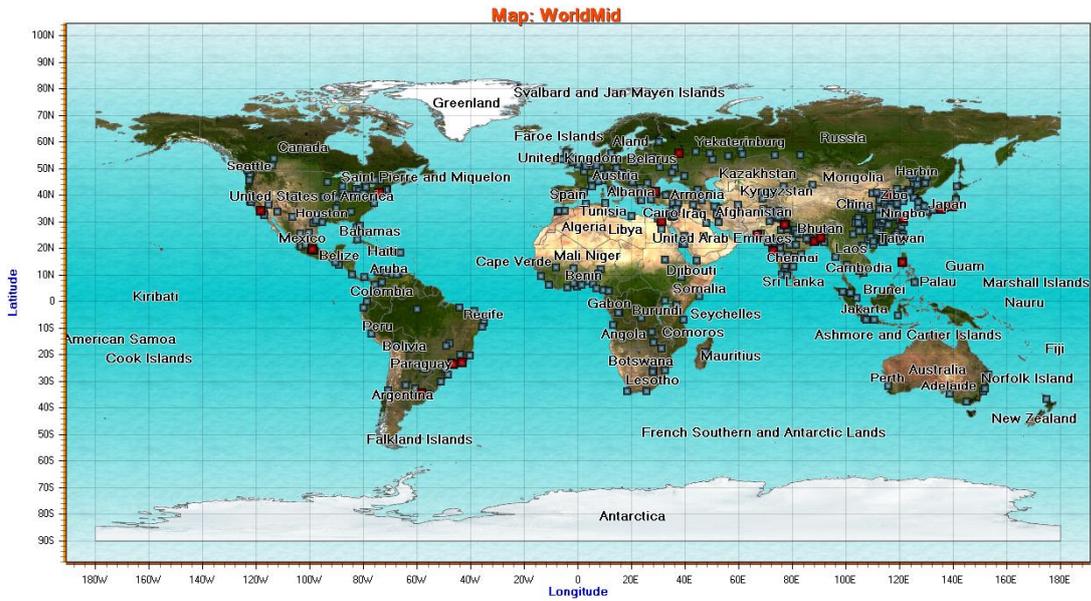


Figure 5-100. Map of the world. LandOptions.FillVisible is set to false, and one background image is set to latitude range of -90...90 and longitude range of -180...180. The map region borders and cities are shown.

### 5.25.6 Combining other series with maps

You can combine the geographical maps with any ViewXY series type. The maps are drawn in the background and the series over them.



Figure 5-101. Map of Europe, with a couple of FreeformPointLine series as routes. Flag markers are added to them as mouse-interactive waypoints.



### 5.25.7 Importing maps from ESRI shape file data

Import feature makes a LightningChart map file (.md) from .shp files. ESRI shapefile (\*.shp) is a widely used map file format, supporting vector and polygon data.

Map wizard can be used to convert shapefile data to LightningChart (LC) map data format. LC format supports layering, so multiple shapefiles can be merged into a single file. Map file structures and objects are pre-processed for maximum run-time performance.

**Tip:** *LightningChart Ultimate demo application has an example of map importing. You can run this import wizard from there, to make custom LC map files through import.*

Conversion is done in minimum of three steps:

1. Selecting files and setting up layers based on the files in the Shapefile Selection Dialog.
2. Determining file text encoding.
3. Selecting items included in the resulting map file.

Note that steps 2 and 3 are repeated for each source shp file. Shapefile does not tell which encoding it uses, so it must be selected by the user.

After the steps, the conversion begins. If the maps are imported from a custom application, the developer is encouraged to setup an event handler, because conversion might take a very long time, so that user can be informed about the conversion progress.

Also if user selects base layer, there might be a considerable delay between the steps, which is due to prefiltering data based on the layer.

#### 5.25.7.1 Programming interface for importing shp data

Conversion is run on a thread that is initialized from Maps.MapConverter class using following method:

```
public bool SelectFilesAndConvert()
```

For monitoring conversion progress there is an event handler delegate:

```
public delegate void ConversionStateChangedHandler(ConversionProgress progress, int i);
```

It is initialized like this:

```
MapConverter mapConverter = new MapConverter();  
mapConverter.ConversionStateChanged += new  
MapConverter.ConversionStateChangedHandler(mapConverter_ConversionStateCh  
anged);
```

## 5.25.7.2 Dialogs

There are usually three dialogs involved in the conversion process. For selecting a filter, there is a distinct dialog.

### 5.25.7.2.1 Shapefile Selection Dialog

After `SelectFilesAndConvert()` function is called, file selection dialog opens. In this dialog user selects the source files and sets up the layering. User can also save the map configuration by selecting proper file at the dialog.

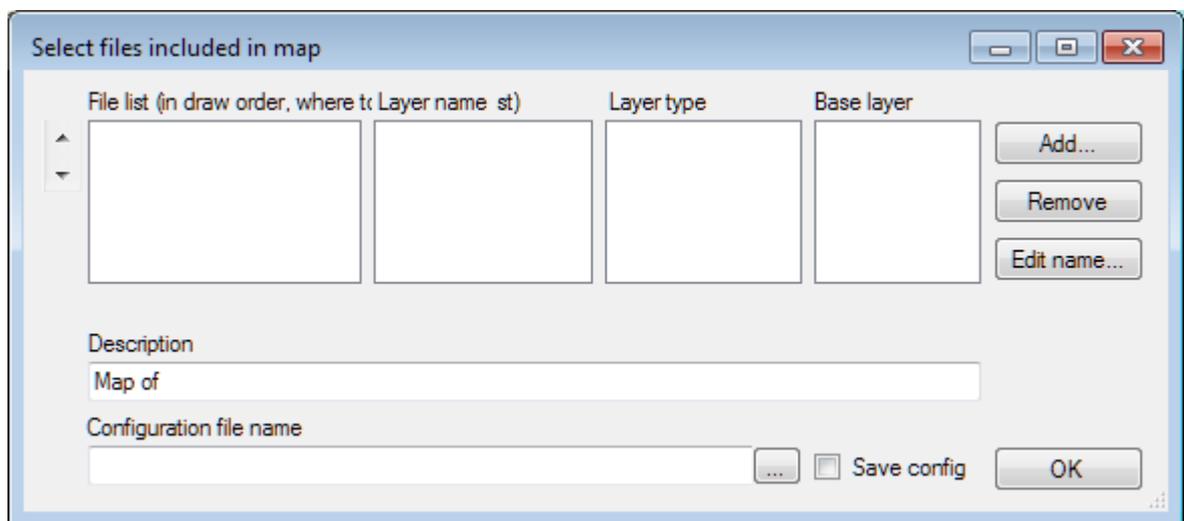


Figure 5-104.. Source shape file selection dialog.

#### File list

Contains list of files in the order in which they are drawn. File data at bottom is drawn last. Order of files can be changed from the up/down buttons on the left of list. Select file and click up/down to move file.

#### Layer name

Name of the layer. E.g. "Countries".

#### Layer type

Type of layer (specifies which options are used to render layer)

- City: layer items are of shapefile type POINT
- Lake: layer items are of shapefile type POLYGON
- Land: layer items are of shapefile type POLYGON
- River: layer items are of shapefile type POLYLINE
- Road: layer items are of shapefile type POLYLINE
- Other: layer items are of shapefile type POLYGON or POLYLINE

**Base layer**

Used to filter upper layer item selection, when user wants a map which contains only single/some countries and there only global map available. E.g. if layer contains countries, only items over the selected countries/country will be included in the resulting map. There is a small offset applied to POINT type, so that if point is near enough of border it's included even if it doesn't overlap with base layer. *If all data from the selected shapefiles are included in the resulting map, don't select base layer as it slows down item selection considerably, because all items are checked if they overlap base layer, which is a very time consuming process.*

**Description**

Free text which is shown in the map properties.

**Configuration file name**

XML configuration file name. Used for importing/replacing a layer. Note! Use single file when creating map configuration as import and replace methods can take only one shp input file.

**Save config**

Check this if you want to save map configuration as xml file for later use. Selecting configuration file automatically sets this checked.

**Add button**

Click to select shapefile to be added to list.

**Remove button**

Removes selected file from list.

**Edit name button**

Click to open "Layer name editor". Set layer name.

**OK button**

Click to advance to next stage (item selection).

### 5.25.7.2.2 Select Record Encoding and Invalid Name Fields

This dialog is used to select file text encoding and fields which have invalid or general name. Shape file encoding may vary and there is no information about the encoding in the file, so user must select valid encoding. The item name may be like "UNK" for multiple items, so in this dialog use can select which item name is emptied. Note that the items are still included in the resulting file, if selected in the next phase.

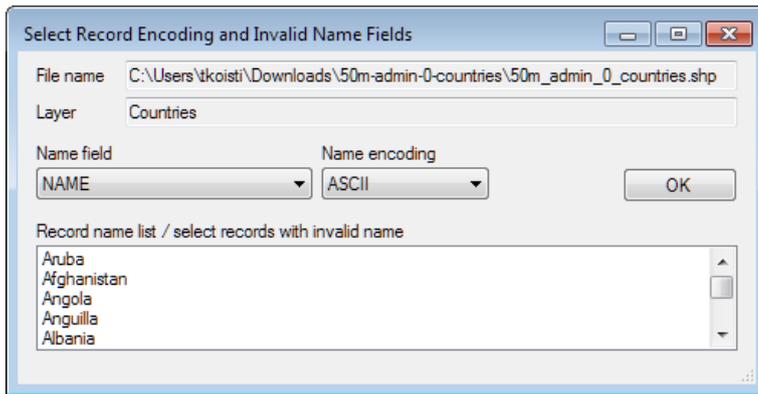


Figure 5-105. 'Record encoding' and 'Invalid name' fields selection dialog.

**File name**

Shape file name for which the encoding applies.

**Layer**

Layer name.

**Name field**

Item name field in the shape file. After selecting different field, the list is updated accordingly.

**Name encoding**

Item name encoding (try different values, if name does not seem to be right). After selecting different encoding, the list is updated accordingly.

**Record name list / select records with invalid name**

List of items for the field selected in the "Name field".

**OK**

Confirm encoding selection (and possible invalid name).

**5.25.7.2.3 Layer data selection dialog**

This dialog is used to select items included in the resulting map file from the shape file. The layer name is concatenated in the title. The dialog is adaptive, so that for certain layers there are some fields which could be selected. E.g. for **River/Road** type layer there will be a Line width selection, which could be set to line width field (if applicable). Note that the data may not contain all the fields asked in the dialog. The **Name** field is mandatory for all items.

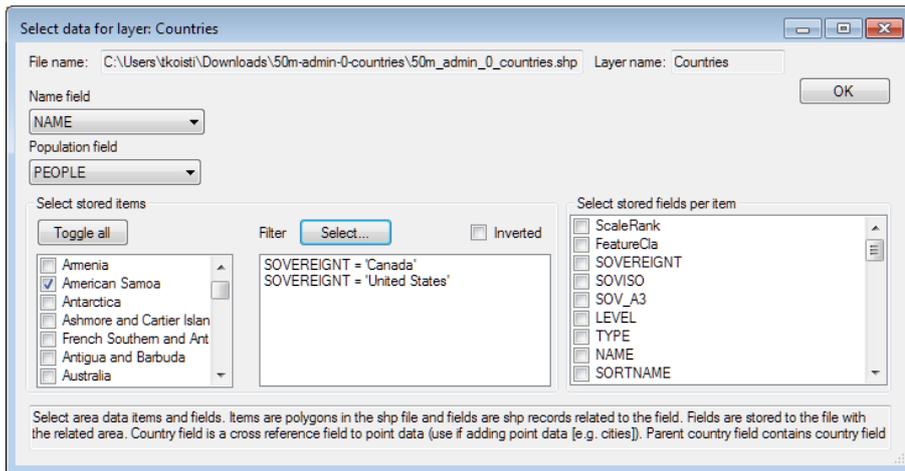


Figure 5-106. Layer data selection dialog.

The user interface items available in the dialog:

**File name**

Name of the file.

**Layer name**

Name of the layer.

**Name field**

Field used for item name. Set automatically from encoding selection dialog, but can be adjusted here also.

**Population field**

Field used for population data.

**Country field**

Country name field.

**Line stroke width field**

Line width. Guides rendering of lines.

**Select stored items**

Select items individually, all or use **Filter** dialog to select subset of items

**Toggle all**

Select all fields from file.

**Filter/Select...**

Select fields which has a field with selected values. In the image above, only items with SOVEREIGNT field set to "Canada" or "United States" are selected in the map.

**Inverted**

Invert filter selection (fields selected with filter are not included in resulting map file).

**Select stored fields per item**

Click on fields which should be included for each item. The fields are key values for Dictionary class, which contains the fields per item.

**5.25.7.2.4 Item filter**

This dialog is opened from Layer data selection dialog and is used to filter items for resulting map.

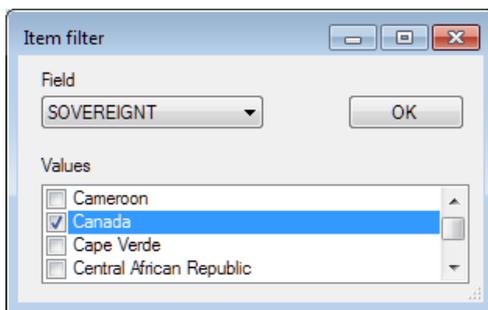


Figure 5-107. Item filter dialog.

**Field**

Select the field on which filtering is based on.

**Values**

Select values which are included in the resulting items.

The above selection means that items which field name SOVEREIGNT contains value "Canada" are included in the resulting map.

**5.25.8 Importing and replacing map layers**

User can import new layers to the map and replace existing layers. There are four methods for importing and replacing a layer in the map from Maps interface. This is very useful when retrieving frequently updated shp data while the software application is running.

**ImportNewLayer** methods inserts a new map layer to given layer index and **ImportReplaceLayer** methods replaces the map layer at the given layer index.

```
public MapConverter.ConversionResult ImportNewLayer(String shpFilename,  
int targetLayerIndex),
```

where **shpFilename** is the name of the source shp file name and **targetLayerIndex** is the index of the new layer. This method uses dialogs presented above for setting up map configuration.

```
public MapConverter.ConversionResult ImportNewLayer(String shpFilename,  
int targetLayerIndex, String configFile),
```

where **shpFilename** is the name of the source shp file name, **targetLayerIndex** is the index of the new layer and **configFile** is map configuration file name. This method uses configuration file created with dialogs presented above.

```
public MapConverter.ConversionResult ImportReplaceLayer(String  
shpFilename, int targetLayerIndex),
```

where **shpFilename** is the name of the source shp file name and **targetLayerIndex** is the index of the new layer. This method uses dialogs presented above for setting up map configuration.

```
public MapConverter.ConversionResult ImportReplaceLayer(String  
shpFilename, int targetLayerIndex, String configFile),
```

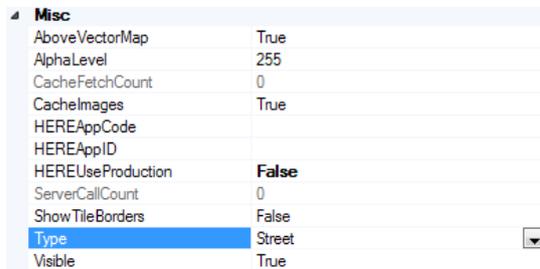
where **shpFilename** is the name of the source shp file name, **targetLayerIndex** is the index of the new layer and **configFile** is map configuration file name. This method uses configuration file created with dialogs presented above.

Configuration file is a plain xml file, which can be edited with a text editor, but editing is not recommended.

## 5.26 Tile maps

LightningChart has support for the following on-line tile data services:

- [Here](#): Street maps, Satellite imagery



Misc	
AboveVectorMap	True
AlphaLevel	255
CacheFetchCount	0
CacheImages	True
HEREAppCode	
HEREAppID	
HEREUseProduction	False
ServerCallCount	0
ShowTileBorders	False
Type	Street
Visible	True

Figure 5-108. Properties of a `TileLayer`.

Add ***TileLayer*** object(s) in **`ViewXY.Maps.TileLayers`** collection. You can insert several layers, and make them semi-transparent with ***AlphaLevel*** property. The ***TileLayer*** objects are rendered in the order of appearance in the ***TileLayers*** collection, first layer in the background. By setting ***AboveVectorMap*** = ***False***, the layer renders before the vector map, if such are defined (see 5.25). By default, the ***TileLayer*** renders after the vector maps.

***TileLayer*** gets information as small images from on-line service provider through http protocol, and shows them in the chart area. The images are refreshed as you zoom or pan the map view.

Loading a new set of tiles will take some time, up to several seconds.

### Tile cache

The chart stores the tiles into a cache folder, which greatly reduces the loading time when panning or zooming frequently in the same region. When the chart needs to show a tile, it first takes look if it can be found in the cache folder, and if not, retrieves it from the web service. In a team use, where many workstations need to access the tile maps, it is wise to select a shared local network server folder. By default, the cache folder is **`c:\Users\[Current user]\AppData\Local\Temp`**.

Set the cache folder in **`ViewXY.Maps.TileCacheFolder`**.

Clear the cache folder by calling **`ViewXY.Maps.ClearTileCacheFolder()`** method.

## 5.26.1 HERE

LightningChart supports tile data service by Here. Developer or end user must make an own contract with Here, to be able to use the Here servers. Free trial keys can be acquired from

<https://developer.here.com/plans/api/consumer-mapping>

### Selecting type

Set **TileLayer.Type = Street** to use street maps. The street maps can be zoomed in very near.

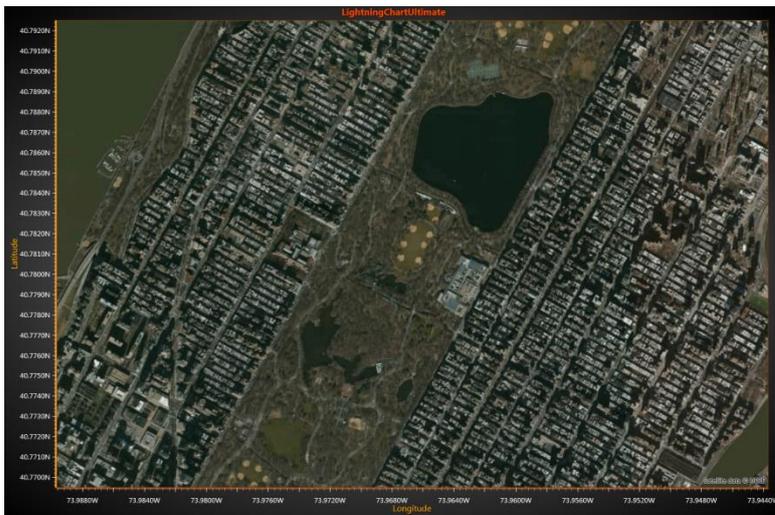


Figure 5-109. TileLayer.Type = Satellite.

Set **TileLayer.Type = Street** to use satellite imagery.

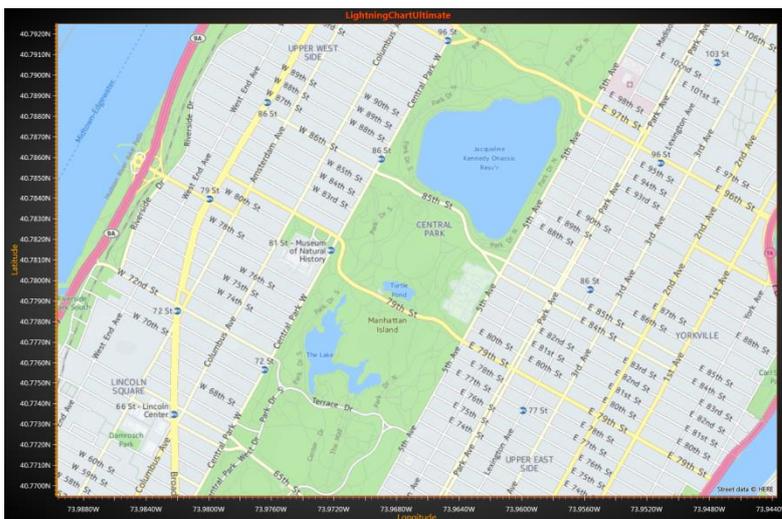


Figure 5-110. TileLayer.Type = Street.

Presenting series and other chart elements, like annotations, is possible.

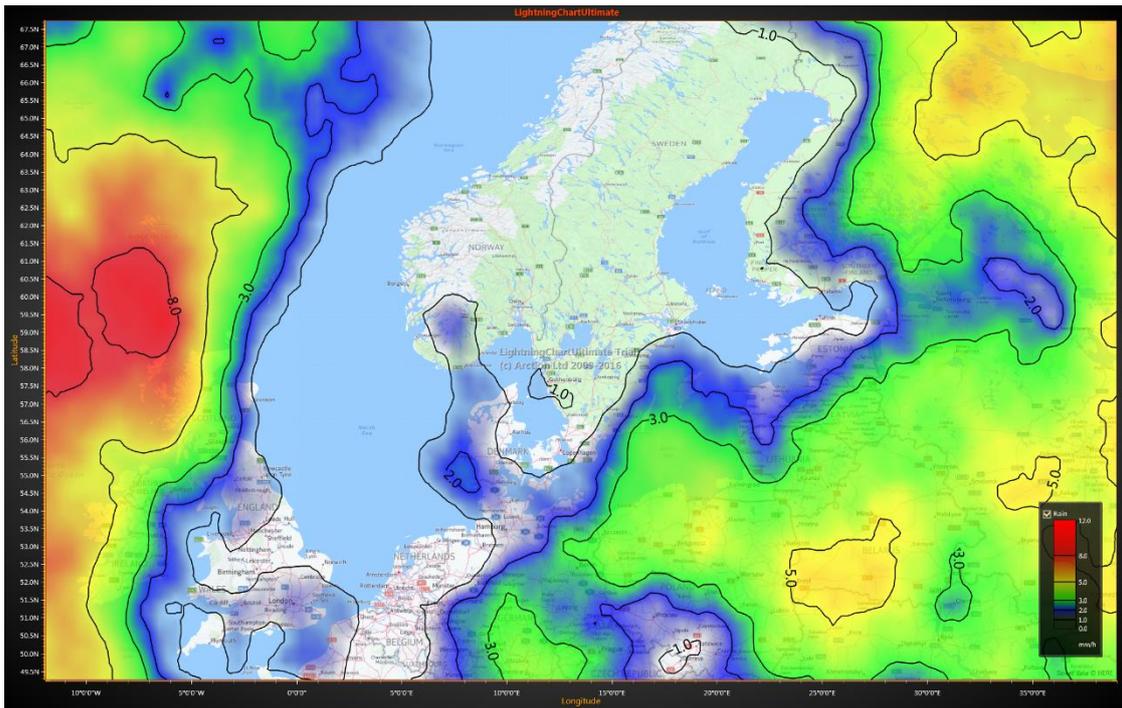


Figure 5-111. Street maps with IntensityGridSeries presenting weather data.

## 5.27 Line series cursors

Line series cursors allow visual analysis of line series data by tracking the values by X coordinate. Series values are can only be resolved with series implementing *ITrackable* interface (*SampleDataSeries*, *PointLineSeries*, *AreaSeries*, *HighLowSeries*). For other series types, Y coordinate is not automatically tracked by cursors. Add *LineSeriesCursor* object into *LineSeriesCursors* collection. Enable *SnapToPoints* to jump the cursor from point to point. Set the cursor tracking style with *Style* property. When *Style* is set to *PointTracking*, you can use any tracking point style, even a bitmap image. When using *HairCrossTracking* style, a horizontal line is drawn at line series point Y value. If multiple points of same series hit in cursor location, line is drawn in the middle of minimum and maximum points.

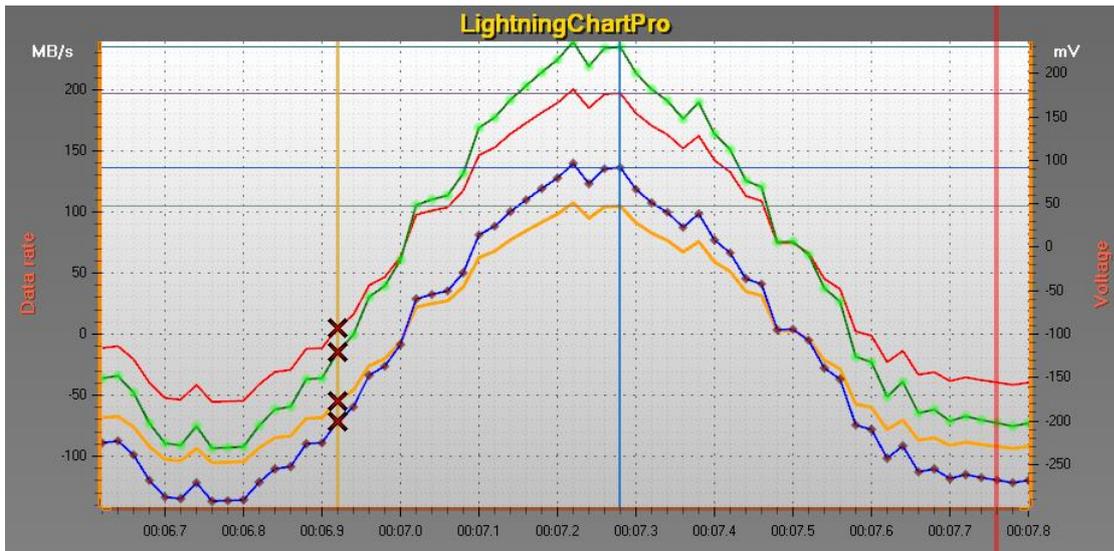


Figure 5-112. Line series cursors: Vertical point tracking cursor, hair-cross tracking cursor and vertical full-length cursor without tracking

By enabling *IndicateTrackingYRange* a horizontal bar is drawn ranging from minimum to maximum of the points hitting in the middle of the cursor.

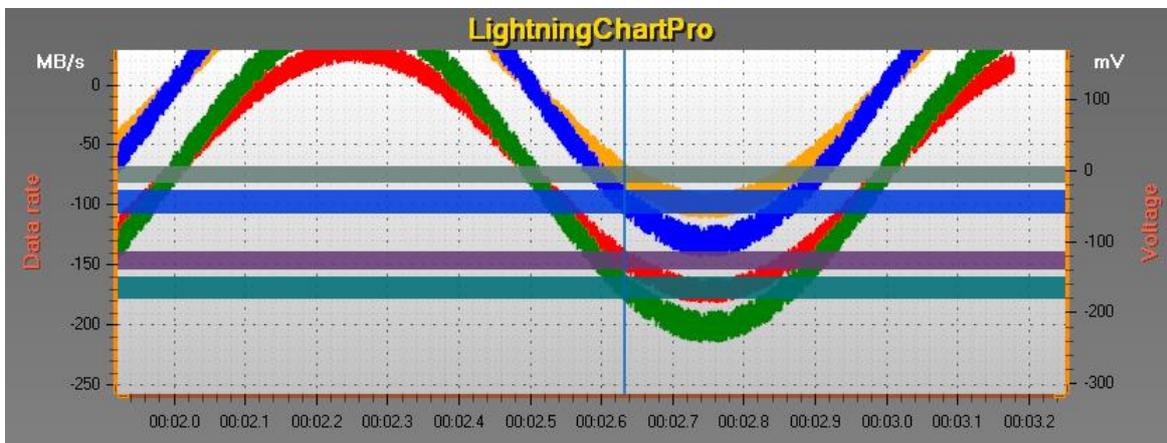


Figure 5-113. Hair-cross cursor with Y range indicator

### 5.27.1 Solving the data values in the position of LineSeriesCursor

The series implementing *ITrackable* interface can be solved by X screen coordinate or by X axis value.

Trackable series have methods for accurate and coarse value solving. The accurate method *SolveYValueAtXValue* loops through data points if necessary and finds the nearest data point match.

The coarse method **SolveYCoordAtXCoord** uses cached rendering data of the series for solving the matching Y screen coordinate.

### 5.27.1.1 Accurate method, solving Y value by X value, by using the data points array

```
LineSeriesValueSolveResult result =
    series.SolveYValueAtXValue(cursor.ValueAtXAxis);

if (result.SolveStatus == LineSeriesSolveStatus.OK)
{
    //PointLineSeries may have two or more points at same X value. If so,
    center it between min and max
    yValue = (result.YMax + result.YMin) / 2.0;
    return true;
}
```

**Note!** When cursor.**SnapToPoints** is disabled, the **SolveYValueAtXValue** returns interpolated value between the points adjacent to it (the intersection of cursor line and the series line).

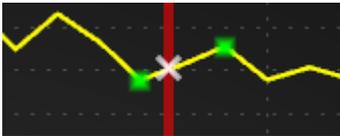


Figure 5-114. SolveYValueAtXValue interpolates the value between adjacent data points when SnapToPoints is disabled.

### 5.27.1.2 Coarse method, solving Y screen coordinate by X coordinate, by using the data points array

```
LineSeriesCoordinateSolveResult result =
series.SolveYCoordAtXCoord((int)Math.Round(fCoordX));

if (result.SolveStatus == LineSeriesSolveStatus.OK)
{
    fCoordY = (result.CoordBottom + result.CoordTop) / 2f;
    if (axisY.CoordToValue((int)Math.Round(fCoordY), out yValue) == false)
    {
        return false;
    }
}
```

When the series holds a lot of data points, say > 100.000, it's typically much faster to use the coarse method. The coarse method can be very inaccurate if the chart size or Y segment height in pixels is low.

The coarse method's screen coordinates can be converted into axis values by calling **CoordToValue** method of X and Y axis.

It is a good practice to use an AnnotationXY object to display values next to the cursor, see 5.19

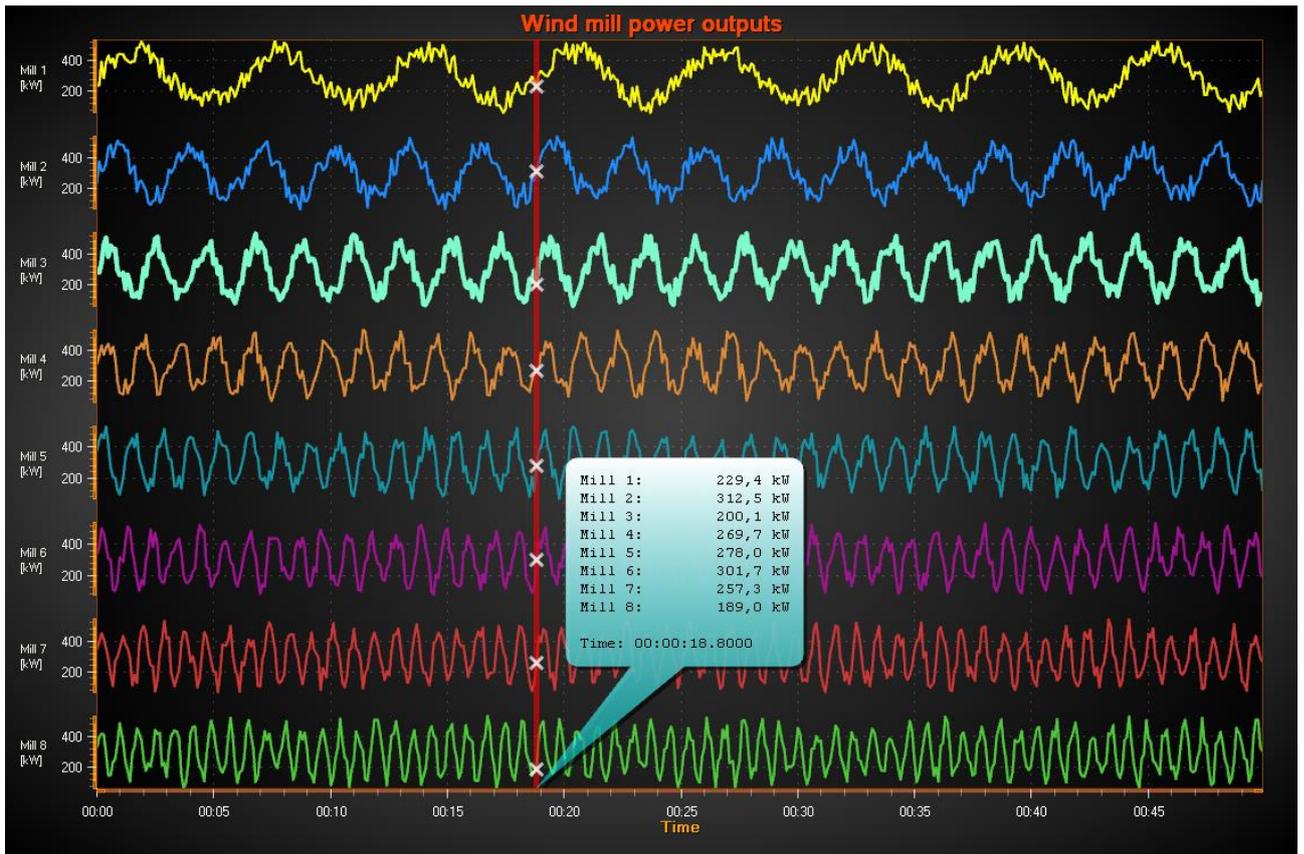


Figure 5-115. LineSeriesCursor used to track PointLineSeries. The values are shown in an AnnotationXY object.

## 5.28 Event markers

Event markers allow marking a point of interest, where something special occurred during real-time monitoring or you just want to mark a piece of data with a special annotation. You can define the marker symbol with **Symbol** property of a marker and a text label with **Label** property. Set the vertical position with **VerticalPosition** property and use **Offset** to shift the object property if necessary. All markers must be assigned with **XValue**, which sets the marker's position on X axis.

To select the shape of the marker, set **Symbol.Shape**. Available shapes are **Rectangle**, **Circle**, **Triangle**, **Flag**, **FlagLightning**, **Cross**, **CrossAim**, **Bitmap**, **HollowBasic**, **HollowBasicActive**, **HollowHarmonic**, **HollowActiveSideband**, **HollowSideband**, **HollowTailedActive** and **HollowTailed**.



Figure 5-116. Markers in trading example.

### 5.28.1 Chart event markers

**ChartMarkers** collection property allows adding chart markers. A chart marker can be used to indicate a point of interests, like “Test person stood up”, “Capacitor bypassed”. Chart event markers can be also understood that they are common to all series. The markers can be dragged with mouse into another location.

### 5.28.2 Line series event markers

Line series have **SeriesEventMarkers** collection property. It can be used to give series specific event markers. The series event markers can be dragged with mouse to another location, while keeping the event marker attached to series values. For doing that, marker's **VerticalPosition** must be set to **TrackSeries**. This is available for series implementing **ITrackable** interface. **VerticalPosition = AtYValue** allows setting the marker vertically to any Y level.

By setting **HorizontalPosition** to **SnapToPoints**, the marker aligns itself horizontally to position of nearest data point. **HorizontalPosition = AtXValue** allows placing the marker horizontally independently of points.

In addition to normal set of shapes, **SeriesEventMarker** supports two special **Symbol.Shape** settings, which allow vertical line with Y axis ticks projection: **HollowYAxis** and **HollowYAxisActive**. They have 1 pixel wide vertical line, and it picks positions of **MajorTicks** and **MinorTicks** from the Y axis the series is attached to. To adjust the tick lengths, edit **YAxis.MajorDivTickStyle.LineLength** and **YAxis.MinorDivTickStyle.LineLength** properties.

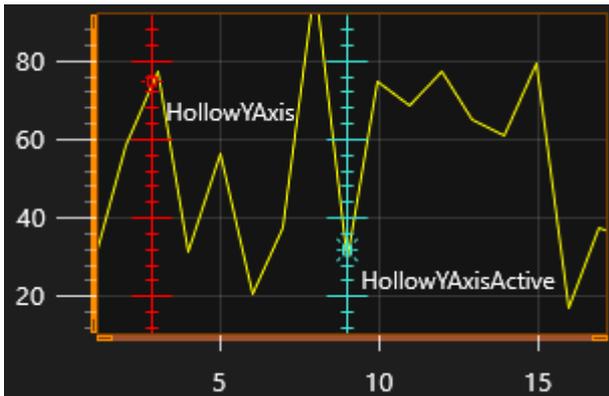


Figure 5-117. Two special SeriesEventMarkers shapes: HollowYAxis and HollowYAxisActive. Very handy when making per-series data cursors.

## 5.29 Persistent series rendering layers

**PersistentSeriesRenderingLayer** can be used for extremely fast rendering of repetitive line/points data, or line/points/high-low/area fill data that is plotted in same X and Y range over and over again.

For example, let's think of a use case of FFT monitoring: Every second 20 new data strips are received. You want to see the newest and all the historic traces. And the monitoring lasts for hours. By rendering this kind of data with regular rendering,  $20 * 60 * 60 = 72000$  new line series are needed every hour. The PC will run out of memory probably before 1 hour is monitored. It is certain that rendering will slow down so badly that it's not usable anymore.

**PersistentSeriesRenderLayer** is kind of a bitmap, that allows adding rendering data incrementally in it. It keeps the graphics until cleared by command. That way, each update round, you'll need only one series to be rendered on the layer, and then the layer rendering on the screen. CPU load or memory footprint doesn't rise. If existing data should be faded away gradually, it can be done by multiplying the alpha of the bitmap pixels.

You can create as many **PersistentSeriesRenderingLayer** objects as you want, and you can render any count of series on each of them, any update round.

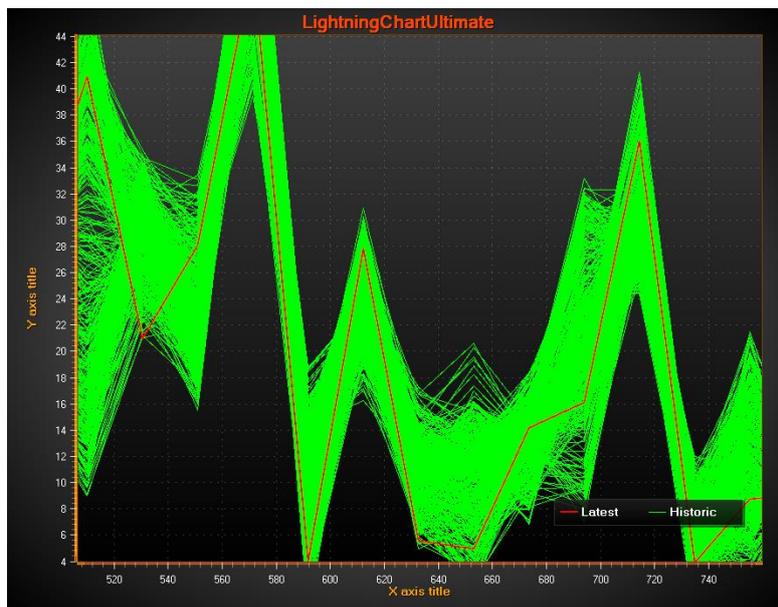


Figure 5-118. Persistent layer shows historical traces, in green color. A regular PointLineSeries is shown over it, in red color.

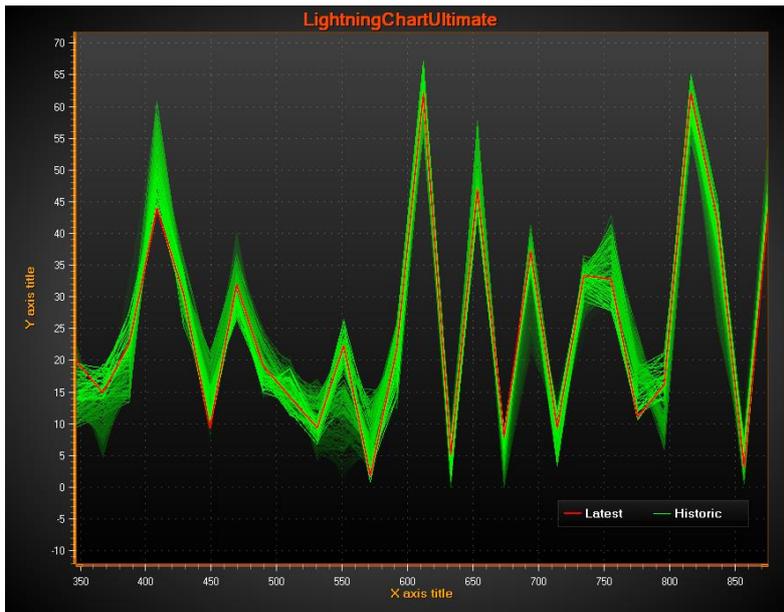


Figure 5-119. Persistent layer shows historical traces, in green color. `MultiplyAlpha` method is called before updating new data in the rendering layer, so it fades the oldest traces away.

### 5.29.1 Creating the layer

The *PersistentSeriesRenderingLayer* is not a sub-property of *ViewXY*, and can't be added with Visual Studio's property grid. *PersistentSeriesRenderingLayer* objects must be created in code. Create it as follows:

```
using Arction.[edition].Charting.Views.ViewXY;

PersistentSeriesRenderingLayer layer = new PersistentSeriesRenderingLayer
(m_chart.ViewXY, m_chart.ViewXY.XAxes[0]);
```

By supplying *ViewXY* object as parameter, it binds the layer in *ViewXY*. Supply the same *XAxis* object that you will be using with the series you render on it.

If you have created several layers, they will render in creation order with the chart.

### 5.29.2 Clearing the layer

*layer.Clear()* clears the layer and initializes the color with `ARGB=(0,255,255,255)`.

*layer.Clear(Color color)* clears the layer with given color. In most cases, it's most useful to set the same color than you use in the background, but set its `A = 0`. With black background, use `layer.Clear(Color.FromArgb(0,0,0,0));`

### 5.29.3 Adjusting layer alpha

**MultiplyAlpha(value)** allows making the layer more transparent or more opaque. Multiplying effects very pixel in the layer separately.

By supplying value < 1, transparency will be increased (decays the layer).

By supplying value > 1, opacity will be increased (brings the layer more visible).

Takes no effect with value of 1.

For example, **MultiplyAlpha(0.8)** sets the alpha to 80% of existing alpha. **MultiplyAlpha(2)** adjust it to 200%.

### 5.29.4 Rendering data into the layer

Render the data into the layer by using any of **PointLineSeries**, **SampleDataSeries**, **FreeformPointLineSeries**, **HighLowSeries** or **AreaSeries** objects. They can be series that have been added into **ViewXY.PointLineSeries**, **ViewXY.SampleDataSeries**, **ViewXY.FreeformPointLineSeries**, **ViewXY.HighLowSeries** or **ViewXY.AreaSeries** collection. You can also use a temporary series that have not been added into the **ViewXY.PointLineSeries**, **ViewXY.SampleDataSeries** or **ViewXY.FreeformPointLineSeries**, **ViewXY.HighLowSeries** or **ViewXY.AreaSeries** collection. Fill the data in the series in regular way (see section 5.5.4 for **PointLineSeries**, 5.6.2 for **SampleDataSeries**, 5.7 for **FreeformPointLineSeries**, 5.9.4 for **HighLowSeries** and 5.10.1 for **AreaSeries**).

**layer.RenderSeries(PointLineSeriesBase series)**: Render one series on the layer.

**layer.RenderSeries(List<PointLineSeriesBase> seriesList)**: Render all given series on the layer. More efficient than calling **layer.RenderSeries(PointLineSeriesBase series)** for each series separately.

**Note!** All the given series will be rendered on the layer, even if their **Visible** is set to **False**.

**Note!** The **X** axis that you use with the series, must be same than supplied for **PersistentSeriesRenderingLayer** constructor. Otherwise, it will skip the series object.

**Note!** **RenderSeries** is for rendering INTO the layer. The layer itself will be rendered just before regular series (**PointLineSeries**, **SampleDataSeries**, **FreeformPointLineSeries**, **HighLowSeries**, **AreaSeries**).

### 5.29.5 Disposing the layer

To dispose the layer and prevent it from rendering with the chart, call **layer.Dispose()**.

### 5.29.6 Anti-aliasing data in the layer

To anti-alias the data in the chart rendering stage, set **layer.AntiAliasing** to **True**. It will make the anti-aliasing also if the hardware doesn't support it.

### 5.29.7 Getting list of layers

`ViewXY.GetPersistentSeriesRenderingLayers()` returns list of all created layers, including **PersistentSeriesRenderingIntensityLayers**.

### 5.29.8 Some layer limitations you should be aware of

Due to its special rendering technique, please keep these limitations in mind:

- X axis **ScrollMode** must be set to **None**. Real-time scrolling of X axis is not possible in this approach.
- Zooming, panning, axis adjustment and chart resize will cause the image to be in un-sync with axis ranges. These features should be disabled when using persistent plotting, or the application logic made so that it clears the layer and recreates temporarily the older line series for new layer rendering (there's event handlers for axis range change and resize).
- Chart resizing will clear the layer. Also resume from Windows desktop lock state.
- Mouse interactivity is not supported on the series rendered only on the layer
- EMF/WMF/SVG export, copy to clipboard in vector format, print in vector format don't support the layer. Only raster formats are supported.

### 5.30 Persistent series rendering intensity layers

*PersistentSeriesRenderingIntensityLayer* allows collecting traces into a layer, and coloring it by the hit count per pixel. The coloring is made by aid of value-range palette. The traces can be with same series types than in *PersistentSeriesRenderingLayer* (see section 5.29), and is very much similar to it, main difference is the coloring. When rendering a trace in location of same pixel again with second rendering call, the intensity of it grows, so it gets higher value in the value-range palette.

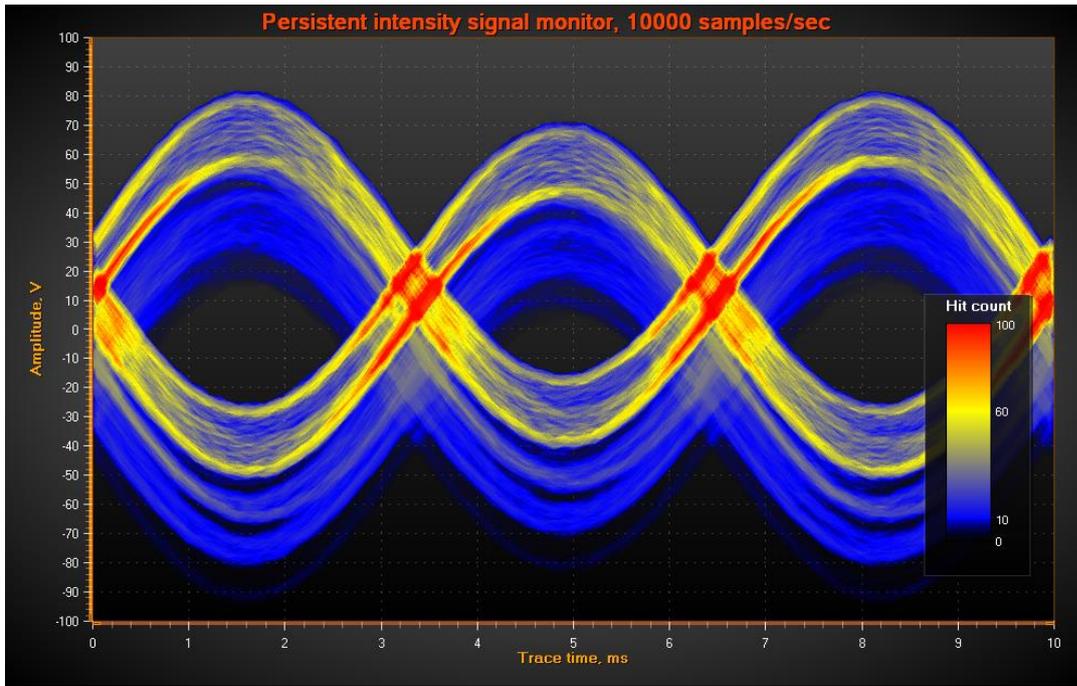


Figure 5-120. Persistent intensity layer highlights in where the activity has concentrated, here in yellow and red.

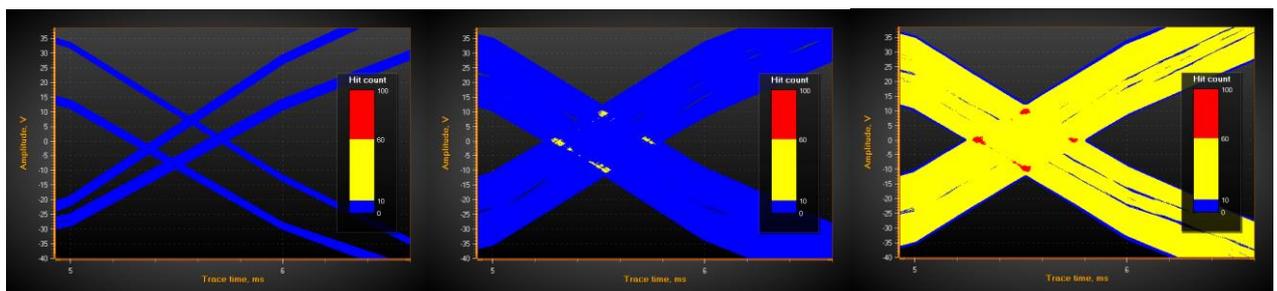


Figure 5-121. Repetitive signal trace is rendered in the same region. On the left, only a couple of traces have been rendered on the layer, so it shows all colors in blue. In the picture on the center, a lot of traces have been rendered, but mostly on different coordinates. In the intersections of the traces, the hit count exceeds the trace count of 10 defined in the palette for yellow color threshold. In the rightmost picture hundreds of traces have been rendered in total, and intersections start to exceed threshold defined for red color.

### 5.30.1 Creating the layer

The **PersistentSeriesRenderingIntensityLayer** is not a sub-property of **ViewXY**, and can't be added with Visual Studio's property grid. **PersistentSeriesRenderingIntensityLayer** objects must be created in code.

Create it as follows:

```
using Arction.LightningChartUltimate.Views.ViewXY;

PersistentSeriesRenderingIntensityLayer layer = new
PersistentSeriesRenderingIntensityLayer(m_chart.ViewXY,
m_chart.ViewXY.XAxes[0]);
```

### 5.30.2 Clearing the layer

**layer.Clear()** clears the layer and resets the counters.

### 5.30.3 Changing palette colors

Define the palette type and steps in **ValueRangePalette** property of the layer. **ValueRangePalette.Type = Gradient** makes a gradient coloring, **ValueRangePalette.Type = Uniform** makes the layer render with discrete color steps.

### 5.30.4 Adjusting the intensity effect of new trace and decay of old traces

Use **NewTraceIntensity** property to control how great intensity effect new trace rendered with **RenderSeries** call gets. Typical value could be 1...100, depending on rapidly you want the color range to fill up with your traces.

Use **HistoryIntensityFactor** to adjust the how much the old traces are decayed. Typical value would be in range 0.5 – 0.99.

Note that setting **HistoryIntensityFactor** itself doesn't update the layer. Next call to **RenderSeries** is needed until it takes effect.

### 5.30.5 Rendering data into the layer

Render a *PointLineSeries*, *FreeformPointLineSeries*, *SampleDataSeries*, *HighLowSeries* or *AreaSeries* to the layer by `RenderSeries` method.

***layer.RenderSeries(PointLineSeriesBase series)***: Render one series on the layer.

***layer.RenderSeries(List<PointLineSeriesBase> seriesList)***: Render all given series on the layer. No performance gain over ***layer.RenderSeries(PointLineSeriesBase series)*** though.

When the data is updated into the layer, ***NewTraceIntensity*** is used for the new trace. Old trace data is decayed with ***HistoryIntensityFactor***, at the same time. ***layer.RenderSeries(List<PointLineSeriesBase> seriesList)*** decays old traces after every series object.

### 5.30.6 Disposing the layer

To dispose the layer and prevent it from rendering with the chart, call ***layer.Dispose()***.

### 5.30.7 Anti-aliasing data in the layer

To anti-alias the data in the chart rendering stage, set ***layer.AntiAliasing*** to ***True***. It will make the anti-aliasing also if the hardware doesn't support it.

### 5.30.8 Getting list of layers

`ViewXY.GetPersistentSeriesRenderingLayers()` returns list of all created layers, including ***PersistentSeriesRenderingLayers***.

## 6. View3D

**View3D** allows visualizing data in 3D space. 3D model can be zoomed, rotated and lit up with various ways. **Different series types can be placed into same 3D view, to make a combined visualization.**

<b>View3D</b>	<b>3D chart view</b>
Annotations	<b>(Collection)</b>
BarSeries3D	<b>(Collection)</b>
> BarViewOptions	
> Camera	
ClipContents	False
> Dimensions	
> FrameBox	
> LegendBox	
Lights	<b>(Collection)</b>
MeshModels	<b>(Collection)</b>
> OrientationArrows	
PointLineSeries3D	<b>(Collection)</b>
Polygons	<b>(Collection)</b>
Rectangles	<b>(Collection)</b>
SurfaceGridSeries3D	<b>(Collection)</b>
SurfaceMeshSeries3D	<b>(Collection)</b>
VolumeModels	<b>(Collection)</b>
> WallOnBack	
> WallOnBottom	
> WallOnFront	
> WallOnLeft	
> WallOnRight	
> WallOnTop	
WaterfallSeries3D	<b>(Collection)</b>
> XAxisPrimary3D	
> XAxisSecondary3D	
> YAxisPrimary3D	
> YAxisSecondary3D	
> ZAxisPrimary3D	
> ZAxisSecondary3D	
> ZoomPanOptions	

Figure 6-1. View3D object main tree.

## 6.1 3D model and dimensions

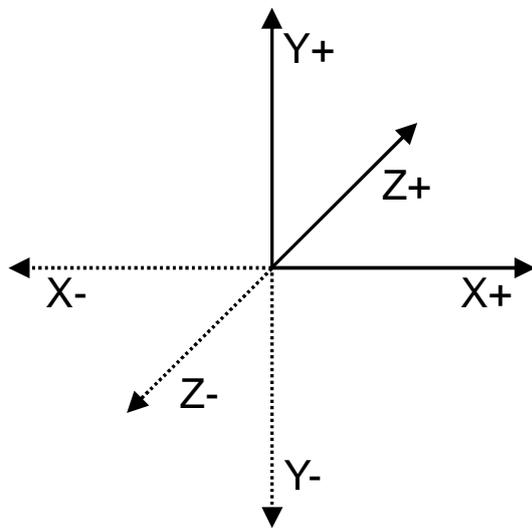


Figure 6-2. 3D model positive and negative directions.

3D model is constructed in the center of 3D world. Dimension magnitudes define the size of the model box in 3D space. Walls and axis sizes are defined with this dimensions box. Use the **Dimensions** property to set each dimension magnitude.

When no camera rotation is not defined, positive X direction is to the right, positive Y dimension upwards and positive Z direction inwards to the screen.

### 6.1.1 World coordinates

Some 3D objects use “*World coordinates*”, not axis values. For example, lights are positioned in this way, to be independent from axis ranges. World coordinates can be called also as “*3D model space coordinates*”.

The origin [0,0,0] is in the center of the model. The actual 3D model space range from [-Dimensions.X/2 to Dimensions.X/2], [-Dimensions.Y/2 to Dimensions.Y/2] and [-Dimensions.Z/2 to Dimensions.Z/2].

LightningChart provides methods to convert values between series values, axis values, world coordinates and screen coordinates. See the demo application examples and help documentation for details.

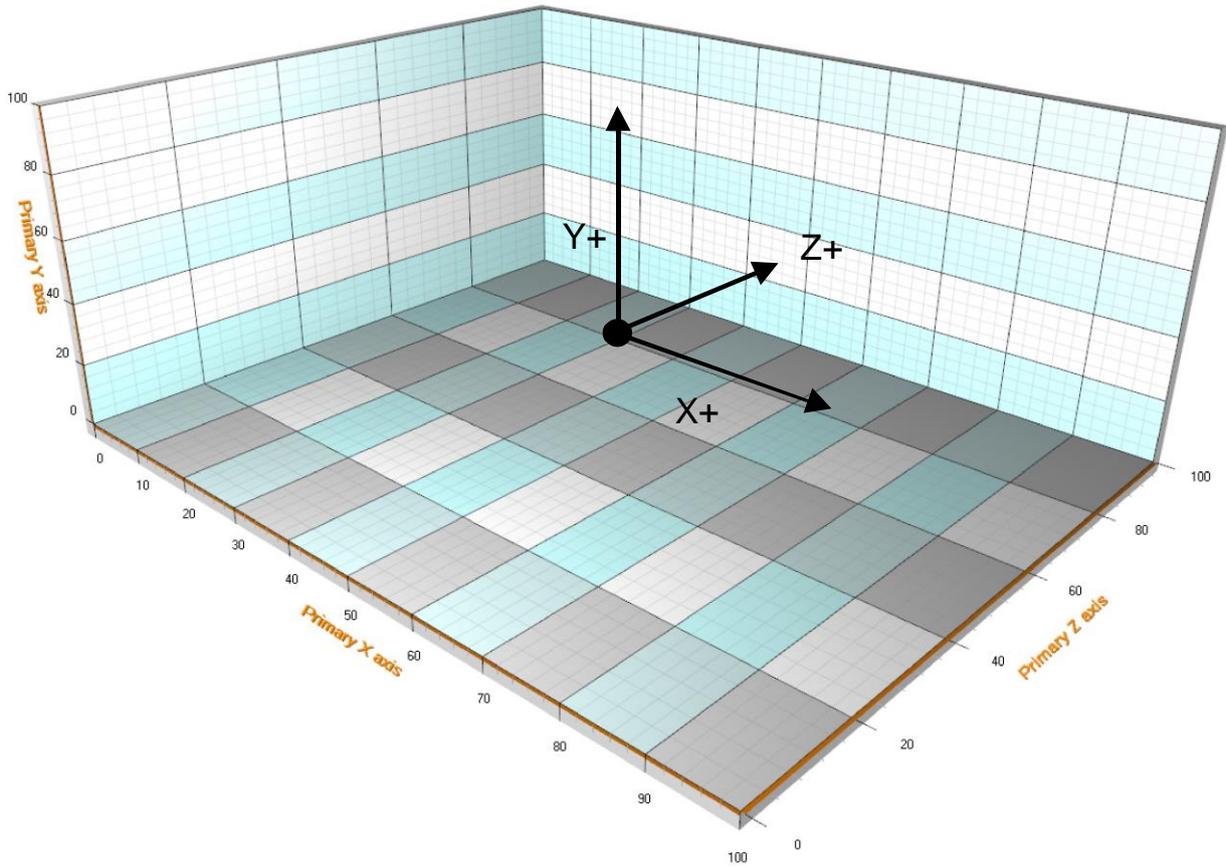


Figure 6-3. Example of 3D view setup. Dimensions are set to X=100, Y=40, Z=80. Walls visible: left, bottom, back. Perspective camera is used.

## 6.2 Walls

Walls (*WallOnFront*, *WallOnBack*, *WallOnTop*, *WallOnBottom*, *WallOnLeft*, *WallOnRight*) are used to present axis grids and gridstrips and to give a base for the axes. By default, bottom, left, right, back and front walls are visible. Their **AutoHide** property is set true. As you rotate the view, the obstructing walls are temporarily hidden so that they don't block the view for chart contents. To force a wall visible, set **Visible = true** and **AutoHide = false**.

Use *XGridAxis*, *YGridAxis*, *ZGridAxis*, *GridStripColorX*, *GridStripColorY*, *GridStripColorZ* and *GridStrips* properties to select from which axes the grid is applied and coloring the grid strips. The available properties depend on the wall orientation.

## 6.3 FrameBox

Instead of walls, you may want to prefer simplified 3D box presentation. Set all the Visible = false for all walls, and set **FrameBox.Style = AllEdges**. Set the color or the frame with **FrameBox.LineColor**.

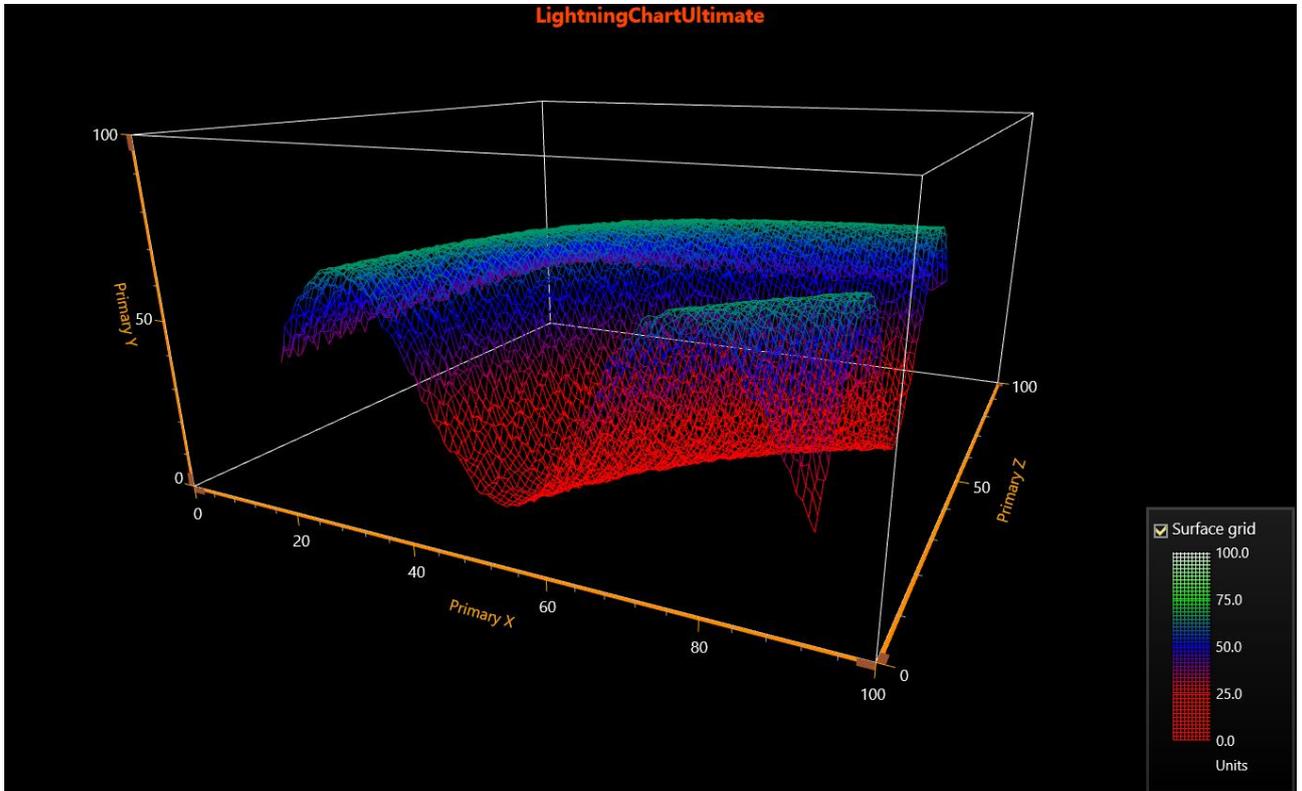


Figure 6-4. FrameBox visible, walls hidden.

## 6.4 Camera

Camera	
MinimumViewDistance	150
OrthographicCamera	False
RotationX	20
RotationXMaximum	89.99
RotationXMinimum	-89.99
RotationY	-25
RotationYMaximum	720
RotationYMinimum	-720
RotationZ	0
RotationZMaximum	720
RotationZMinimum	-720
Target	
X	0
Y	-10
Z	0
ViewDistance	180

Figure 6-5. Camera properties.

Camera type, location, distance and target together determine the 3D viewpoint. Use **RotationX**, **RotationY**, **RotationZ** and **ViewDistance** to set the camera position in the 3D model space. Target the camera to preferred direction by setting the **Target** property. Change to orthographic camera by setting **OrthographicCamera** true. With orthographic camera, the **ViewDistance** does not effect. Perspective camera shows the model in real-life way, and orthographic camera is better for certain technical applications.

**RotationX**, **RotationY** and **RotationZ** can be limited by setting boundaries in **RotationXMinimum**, **RotationXMaximum**, **RotationYMinimum**, **RotationYMaximum**, **RotationZMinimum** and **RotationZMaximum** properties.

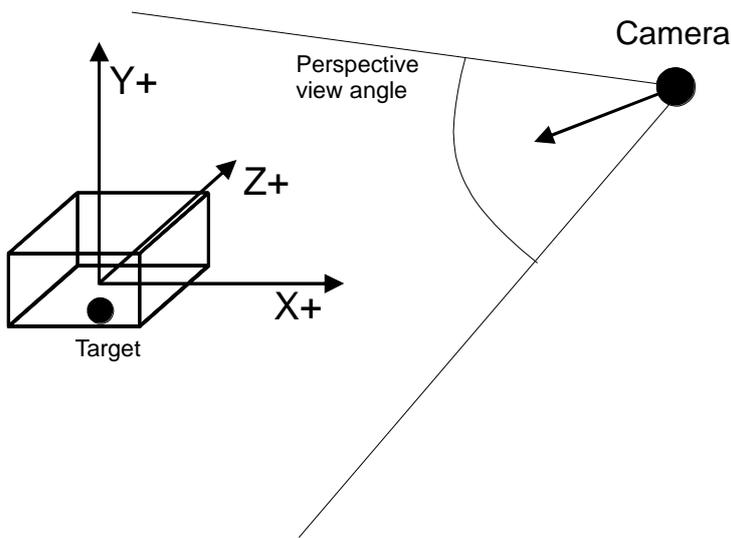


Figure 6-6. Perspective camera presentation in 3D space.

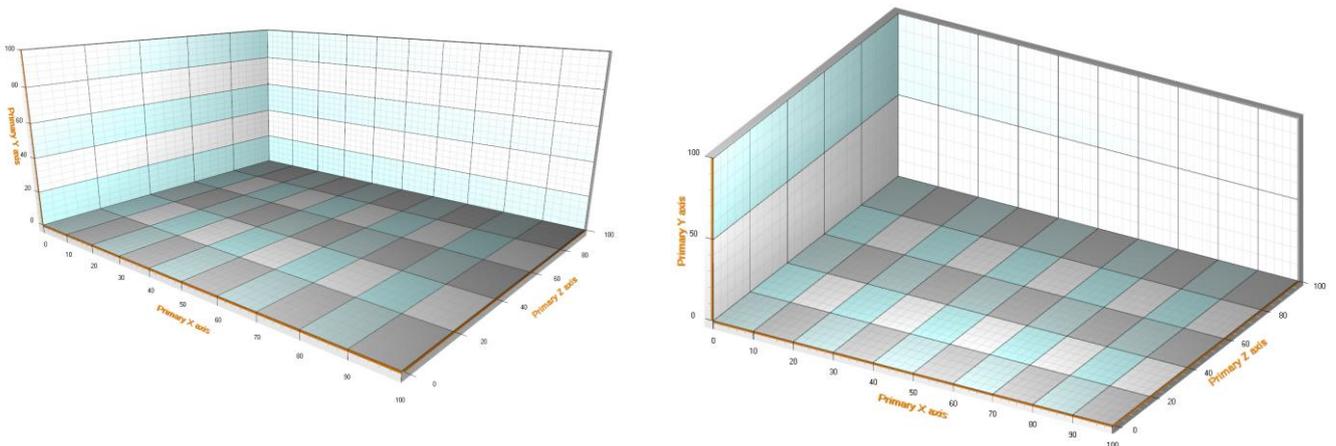


Figure 6-7. Perspective camera view and orthographic camera view in 3D space.

## 6.4.1 Predefined cameras

Use *SetPredefinedCamera* method of *View3D.Camera* to set one of the predefined cameras.

## 6.5 Lights

Lights can be freely positioned anywhere in the 3D model space. Several lights can be added into *Lights* collection property. There are two different light types: *Directional* and *PointOfLight*.

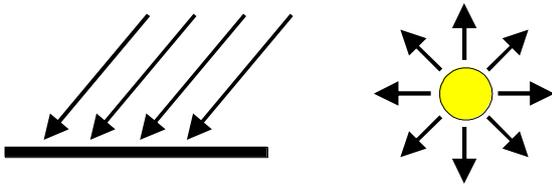


Figure 6-8. Directional light and point of light.

**NOTE!** Some series types allow suppressing lighting totally from its surface, by using *SuppressLighting* property. Check it's not enabled if you want the series to be correctly lit. Surface series have *LightedSurface* property, which selects the surface side that is correctly lit.

**NOTE!** If you place all the lights inside the 3D model box, the wall edges may appear very dark, possibly making axis ticks hardly visible. Adjust axis tick coloring in such case.

### 6.5.1 Directional light

In *Directional* light, the light rays are parallel, and the light intensity does not attenuate as the distance increases. The light flux gets direction from *Location* and *Target* properties. Use *LocationFromCamera* property to use the location of camera as source of light.

### 6.5.2 Point of light

In *PointOfLight* intensity attenuates as the distance grows. Use *AttenuationConstant*, *AttenuationLinear* and *AttenuationQuadratic* properties to control the attenuation over distance. *Target* has no meaning with this light type, as the light is distributed equally to all directions.

### 6.5.3 Lights and materials

All 3D objects have a **Material** property. Material tells how to react to lights. Material's **DiffuseColor** reacts with **DiffuseColor** of a light. Material's **SpecularColor** reacts with light's **SpecularColor**. Diffuse color can be understood as a matte base color, and specular color is the color that reflects off the lit surface. Using high **SpecularPower** gives the object a metallic look.

Surface series have **ColorSaturation** property, valid range is 0...100%. High value boosts the surface fill colors, and reduces the shading effect.

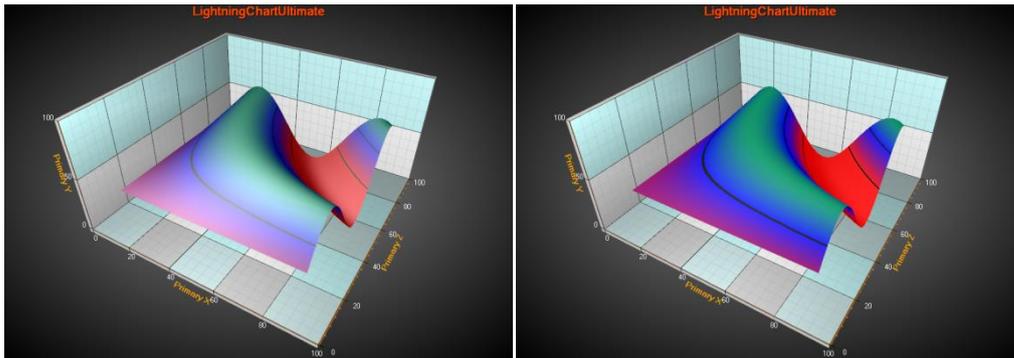


Figure 6-9. The picture on the left has ColorSaturation = 50%. The picture on the right has ColorSaturation = 85%.

## 6.5.4 Predefined lighting schemes

Use *SetPredefinedLightingScheme* method of View3D to select a built-in predefined lighting scheme.

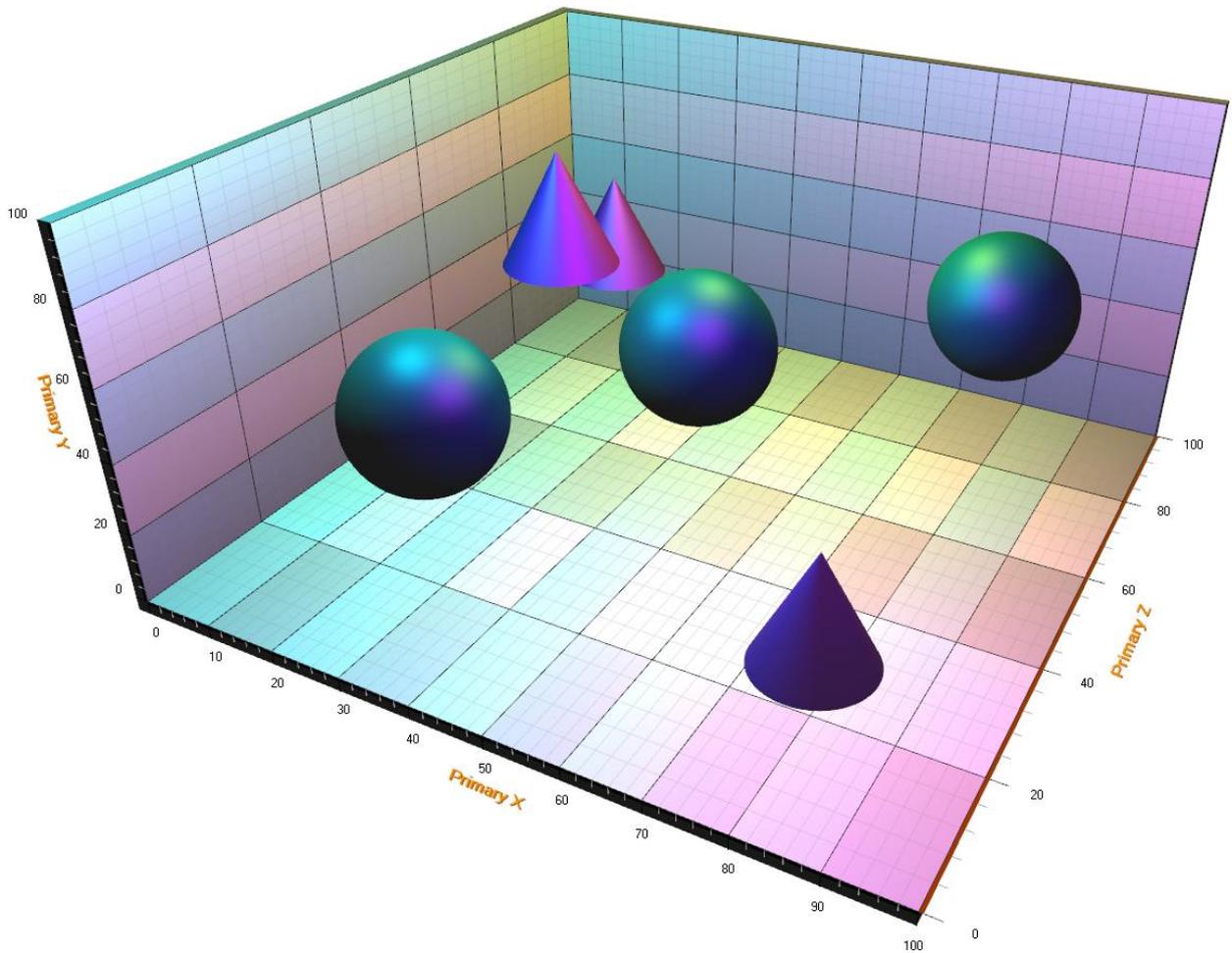


Figure 6-10. Predefined 'DiscoCMY' scheme in use. The scheme is composed from three differently colored PointOfLights near the ceiling. The spheres and cones are made with a couple of series of type PointLineSeries3D.

## 6.6 Axes

For each dimension, there's two axes: primary and secondary. So, under **View3D**, the following axis properties are available: **XAxisPrimary3D**, **XAxisSecondary3D**, **YAxisPrimary3D**, **YAxisSecondary3D**, **ZAxisPrimary3D** and **ZAxisSecondary3D**.

In general, the 3D axes behave very much like **ViewXY**'s axes. Many of the properties and methods are similar.

### 6.6.1 Location

The axes can be positioned in 3D model box corners. Use **Location** property of an axis to adjust that.

- For X axis, the **Location** options are: **BottomFront**, **BottomBack**, **TopFront** and **TopBack**.
- For Y axis, the **Location** options are: **FrontLeft**, **FrontRight**, **BackLeft** and **BackRight**.
- For Z axis, the **Location** options are: **BottomLeft**, **BottomRight**, **TopLeft** and **TopRight**.

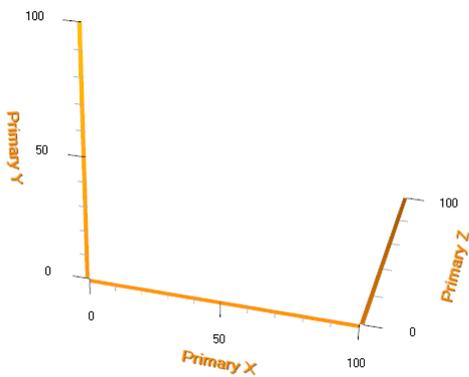


Figure 6-11. Default axis location setup, XAxisPrimary at BottomFront, YAxisPrimary at FrontLeft and ZAxisPrimary in BottomRight.

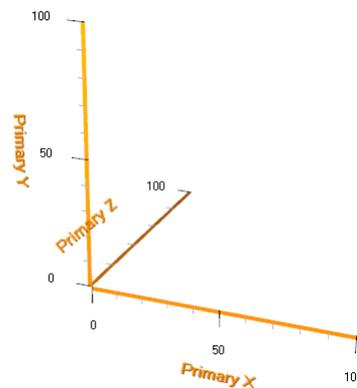


Figure 6-12. ZAxisPrimary location set to BottomLeft.

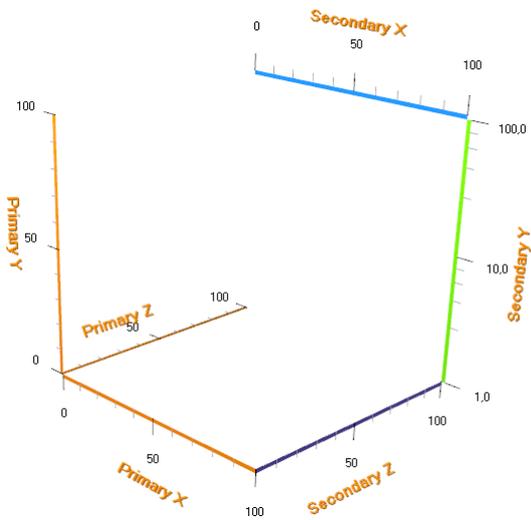


Figure 6-13. Secondary axes set visible and their locations and colors set arbitrarily. Secondary Y axis ScaleType set to Logarithmic.

## 6.6.2 Orientation

Each axis can be oriented in two planes.

- X axis: XY and XZ planes
- Y axis: XY and YZ planes
- Z axis: XZ and YZ planes

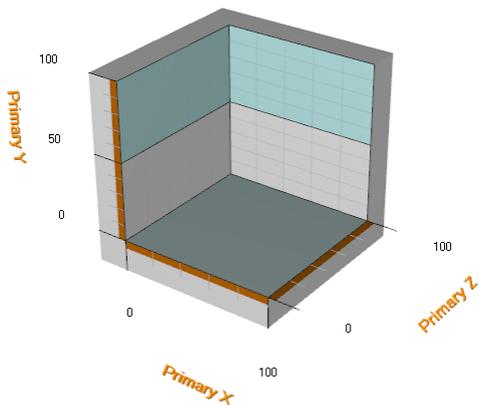


Figure 6-14. X axis orientation is set to XY, Y axis orientation to XY, Z axis orientation to XZ.

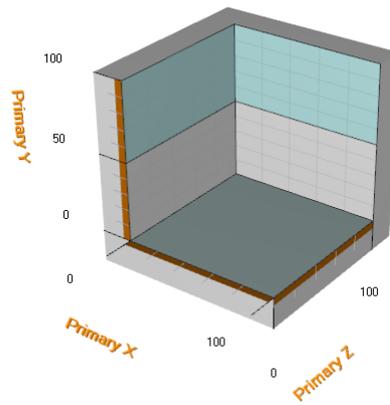


Figure 6-15. Y axis orientation stays same, but X axis orientation is changed to XZ and Z axis orientation is changed to ZY plane.

### 6.6.3 CornerAlignment

The axis alignment in 3D model box corners can be changed with **CornerAlignment** property. Use **MajorDivTickStyle** and **MinorDivTickStyle Alignment** properties to control the text alignment.

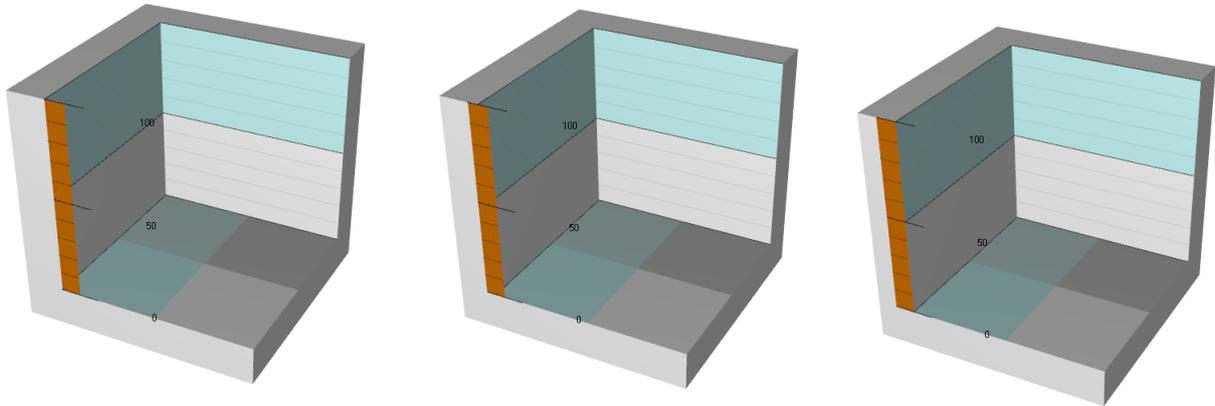


Figure 6-16. First figure: Only Y axis is visible in this example. Y Axis **CornerAlignment** is set to **Inside**. Alignment properties in **MajorDivTickStyle** and **MinorDivTickStyle** are set to **Near**. Second figure: **CornerAlignment** is set to **AtCorner**. Third picture: **CornerAlignment** is set to **Outside**.

## 6.7 3D series, general

View3D's series allow data visualization in different ways and formats. All series are bound to axis value ranges. For each dimension, you can select to bind the series to primary or secondary axis. Use **XAxisBinding**, **YAxisBinding** and **ZAxisBinding** properties to control that.

## 6.8 PointLineSeries3D

PointLineSeries3D allows presenting points and line in 3D space. For points, there are many basic 3D shapes available. Points are connected together with line, if **LineVisible** property is true.

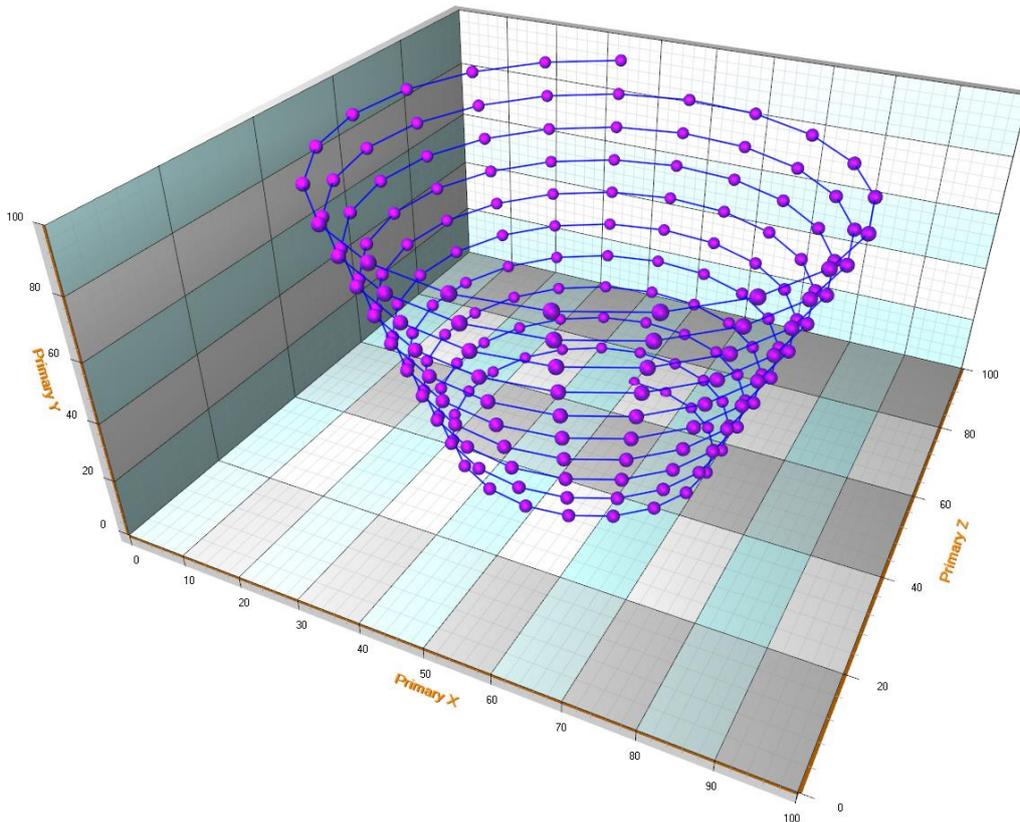


Figure 6-17. A PointLineSeries3D example. PointStyle's Shape is set to Sphere.

### 6.8.1 Point styles

Points can be shown as real 3D points, or as 2D shapes.

Point Style	
DetailLevel	30
▸ Rotation3D	
▾ Shape2D	
Angle	<b>45</b>
Antialiasing	True
BitmapAlphaLevel	255
BitmapImage	<input type="checkbox"/> <b>(none)</b>
BitmapImageTintColor	<input type="checkbox"/> White
BodyThickness	<b>3</b>
BorderColor	<input type="checkbox"/> <b>128, 0, 0, 0</b>
BorderWidth	0
Color1	<input type="checkbox"/> <b>Red</b>
Color2	<input type="checkbox"/> <b>Black</b>
Color3	<input type="checkbox"/> Black
GradientFill	Edge
Height	<b>13</b>
LinearGradientDirection	Down
Shape	<b>Cross</b>
UseImageSize	True
Width	<b>13</b>
Shape3D	Box
ShapeType	<b>Shape2D</b>
▸ Size3D	

Figure 6-18. PointStyle property tree. ShapeType is the switch between 2D and 3D shapes.

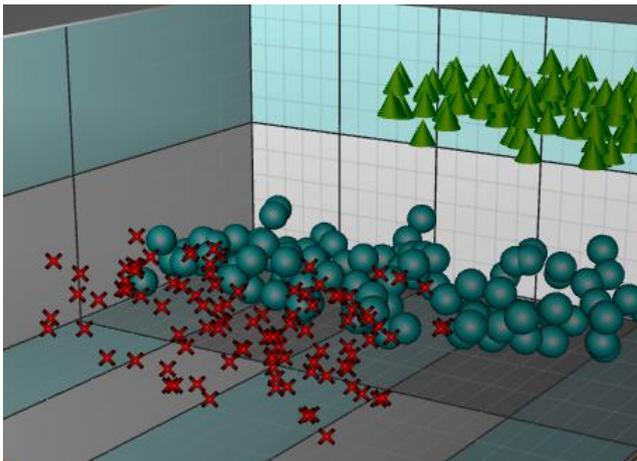


Figure 6-19. Red crosses are have ShapeType = Shape2D. Teal and Green objects have ShapeType = Shape3D.

**NOTE!** 2D shapes are rendered on top of all 3D objects and they don't have any support for hiding them based on other objects visibility.

## 6.8.2 Line styles

LineStyle	
AntiAliasing	<b>Normal</b>
Color	<span style="color: yellow;">■</span> Yellow
LineOptimization	<b>Hairline</b>
Pattern	Solid
PatternScale	1
Width	<b>0.2</b>

Figure 6-20. LineStyle properties.

The lines can be rendered as shaded 3D lines or as 1 pixel wide hair line.

When having a lot of data in the series, setting **LineOptimization = Hairline** is recommended, otherwise performance is impaired.

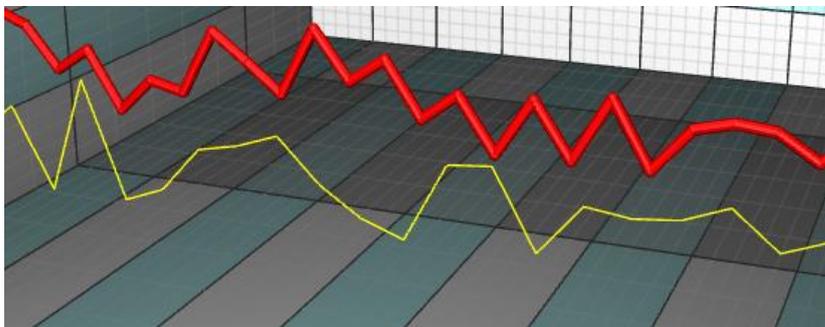


Figure 6-21. Yellow line: LineStyle.LineOptimization = Hairline. Red Line: LineStyle.LineOptimization = NormalShaded.

## 6.8.3 Adding points

PointLineSeries3D support two different point formats

- Points property (SeriesPoint3D array)
- PointsCompact property (SeriesPointCompact3D)

Set the point format to be used in **PointsType** property.

**Note! Bindable WPF chart does not support PointsCompact.**

### 6.8.3.1 Points

By using Points property, all the advanced coloring of points are supported. SeriesPoint3D structure consists of the following fields:

<b>double X:</b>	X axis value
<b>double Y:</b>	Y axis value
<b>double Z:</b>	Z axis value
<b>Color color:</b>	individual data point color, only applies when <b>IndividualPointColors</b> is enabled, or <b>MultiColorLine</b> is enabled.
<b>double sizeFactor:</b>	size factor multiplies the size defined by <b>PointStyle.Size</b> . Only applies when using <b>IndividualPointSizes</b> is enabled.
<b>object Tag:</b>	freely assignable auxiliary object, for example to attach some details.

The points must be added in code. Use **AddPoints(...)** method to add points to the end of existing points.

```
SeriesPoint3D[] pointsArray = new SeriesPoint3D [3];
pointsArray [0] = new SeriesPoint3D (50, 50, 50);
pointsArray [1] = new SeriesPoint3D (30, 50, 20);
pointsArray [2] = new SeriesPoint3D (80, 50, 80);

chart.View3D.PointLineSeries3D[0].AddPoints(pointsArray); //Add points to the
end
```

To set whole series data at once, and overwrite old points, assign the new point array directly:

```
chart.View3D.PointLineSeries[0].Points = pointsArray; //Assign the points
array
```

### 6.8.3.2 PointsCompact

PointsCompact property enables low memory consumption, important when having a lot of data points.

SeriesPointCompact3D structure consists of the following fields:

<b>float X:</b>	X axis value
<b>float Y:</b>	Y axis value
<b>float Z:</b>	Z axis value

```
SeriesPointCompact3D[] pointsArray = new SeriesPointCompact3D[3];
pointsArray [0] = new SeriesPointCompact3D(50, 50, 50);
pointsArray [1] = new SeriesPointCompact3D(30, 50, 20);
pointsArray [2] = new SeriesPointCompact3D(80, 50, 80);

chart.View3D.PointLineSeries3D[0].AddPoints(pointsArray); //Add points to the
end
```

To set whole series data at once, and overwrite old points, assign the new point array directly:

```
chart.View3D.PointLineSeries[0].PointsCompact = pointsArray; //Assign the points array
```

#### 6.8.4 Coloring points individually

By setting **IndividualPointColors = True**, the color fields of points apply instead of **Material.DiffuseColor**.

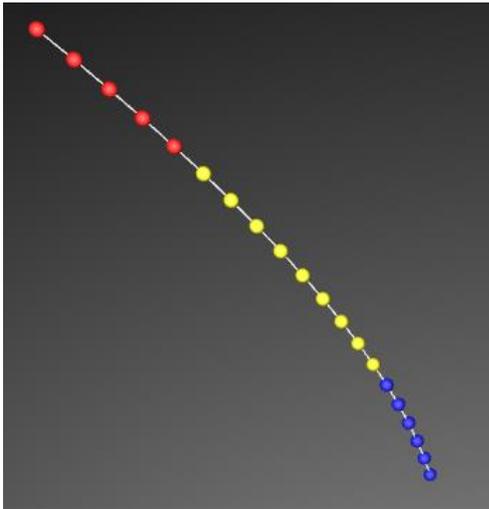


Figure 6-22. IndividualPointColors in use.

**Note!** Individual point coloring is not supported when having **PointsType = PointsCompact**.

#### 6.8.5 Setting points sizes individually

By setting **IndividualPointSizes = True**, **sizeFactor** fields from the points take effect. The factor multiplies the size defined in **PointStyle.Size**.

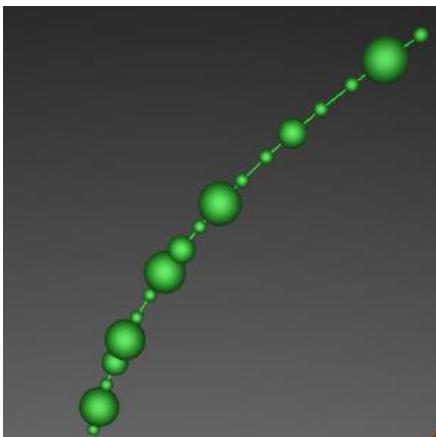


Figure 6-23. IndividualPointSizes in use.

**Note! Individual point sizes are not supported when having `PointsType = PointsCompact`.**

### 6.8.6 Multi-coloring line

To color the line with given data point colors, set **`MultiColorLine = True`**. The chart interpolates the color gradients between adjacent points.

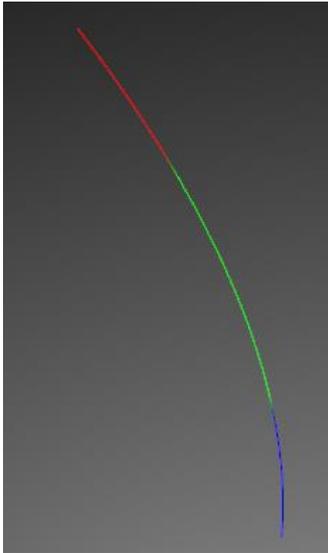


Figure 6-24. `MultiColorLine` enabled.

**Note! `MultiColorLine` is not supported when having `PointsType = PointsCompact`.**

### 6.8.7 Displaying millions of scatter points

To be able to show a very high count of scatter points, set **`PointsOptimization = Pixels`**. Then each series point will render as a single pixel.

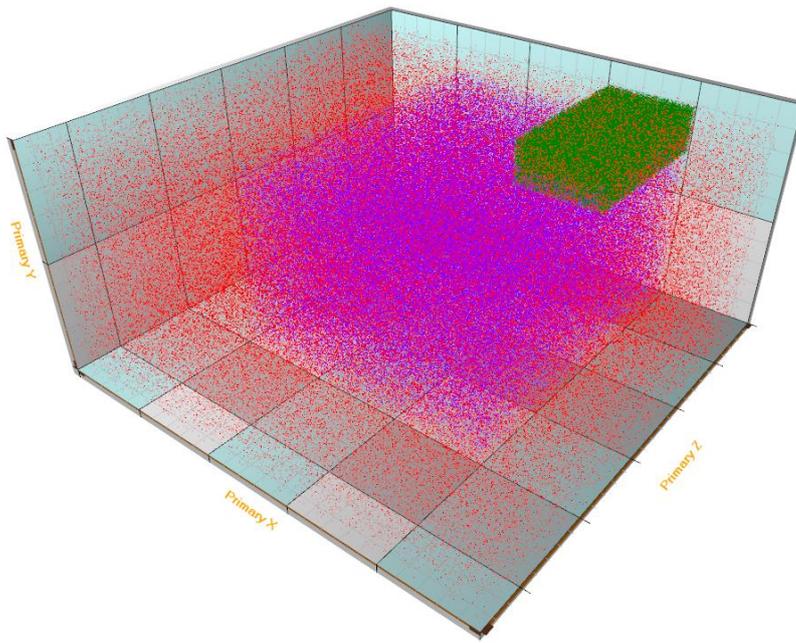


Figure 6-25. Millions of scatter points. LineVisible = False, PointsVisible = True, PointsOptimization = Pixels.

## 6.9 SurfaceGridSeries3D

SurfaceGridSeries3D allows visualizing data as a 3D surface. In SurfaceGridSeries3D, nodes are equally spaced in X dimension, and they are equally spaced in Z dimension, as well.

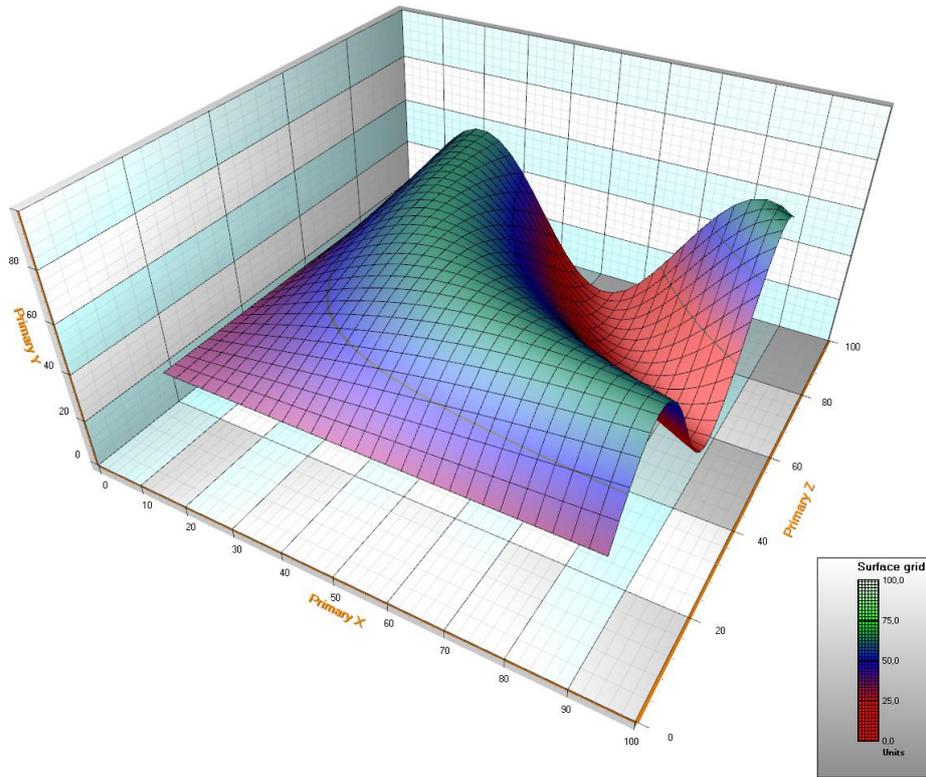


Figure 6-26. Surface grid series with default style. Height data is made with a sine formula. Legend box shows the height coloring intervals.

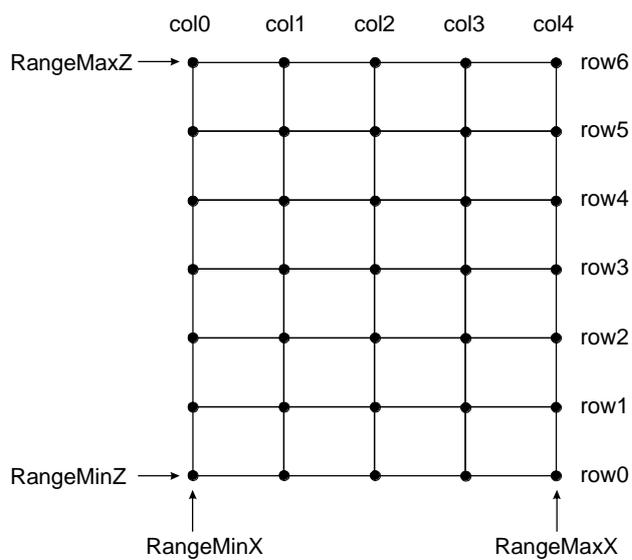


Figure 6-27. Surface grid nodes. SizeX = 5, SizeZ = 7.

Node distances are automatically calculated as

$$\text{node distance } X = \frac{\text{RangeMaxX} - \text{RangeMinX}}{\text{SizeX} - 1}$$

$$\text{node distance } Z = \frac{\text{RangeMaxZ} - \text{RangeMinZ}}{\text{SizeZ} - 1}$$

### 6.9.1 Setting surface grid data

- Set X range by using **RangeMinX** and **RangeMaxX** properties, to order the minimum and maximum value in assigned X axis.
- Set Z range by using **RangeMinZ** and **RangeMaxZ** properties, to order the minimum and maximum value in assigned Z axis.
- Set **SizeX** and **SizeZ** properties to give the grid a size as columns and rows.
- Set Y values for all nodes:

#### Method, with Data array index

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexZ = 0; nodeIndexZ < rowCount; nodeIndexZ ++)  
    {  
        Y //some height value.  
        gridSeries.Data[iNodeX, iNodeZ].Y = Y;  
    }  
}  
gridSeries.InvalidateData(); //Notify new values are ready and to refresh
```

#### Alternative method, usage of SetDataValue

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexZ = 0; nodeIndexZ < rowCount; nodeIndexZ ++)  
    {  
        Y //some height value  
        gridSeries.SetDataValue(nodeIndexX, nodeIndexX,  
            0, //X value is irrelevant in grid  
            Y,  
            0, //Z value is irrelevant in grid  
            Color.Green); //Source point colors are not used in this  
            example, so use any color here  
    }  
}  
gridSeries.InvalidateData(); //Notify new values are ready and to refresh
```

## 6.9.2 Creating surface from bitmap file

You can create a surface from a bitmap image. Use ***SetHeightDataFromBitmap*** method to achieve that. The surface gets the size of the bitmap size (if no anti-aliasing or resampling is used). For each bitmap image pixel, Red, Green and Blue values are summed. The greater the sum, the higher will be the height data value for that node. Black and dark colors get lower values and bright and white colors get higher values.

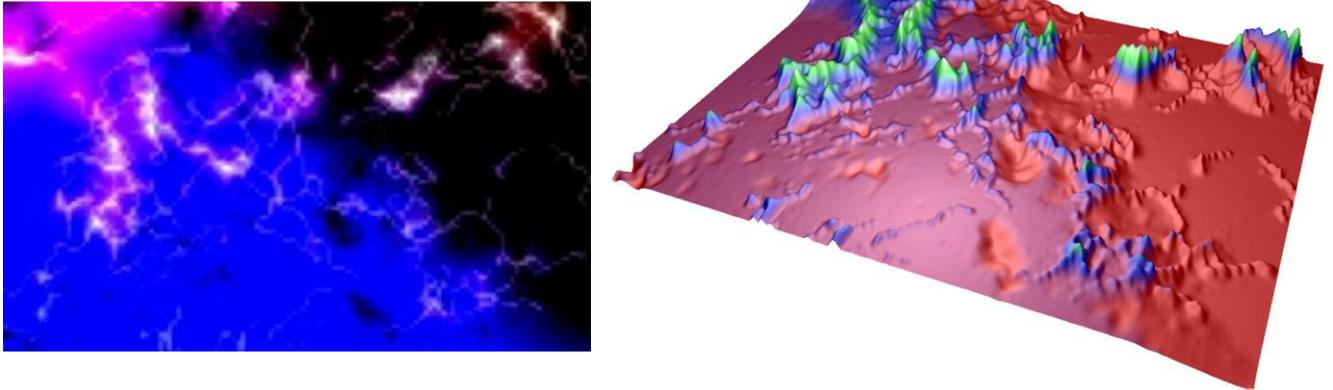


Figure 6-28. Source bitmap and calculated surface height data. Dark values stay low and bright values get higher in the surface.

### 6.9.3 Fill styles

Use **Fill** property to select the filling style. The following options are available

- **None:** By using this, no filling is applied. This is the selection you may want to use with wireframe mesh.
- **FromSurfacePoints:** The colors of the Data property nodes are used.
- **Toned:** ToneColor applies
- **PalettedByY:** Coloring by Y values by palette, see section 6.9.4.
- **PalettedByValue:** Coloring by **SurfacePoint's Value** fields by palette, see section 6.9.4.
- **Bitmap:** Bitmap image is stretched to cover the whole surface. Set the bitmap image in **BitmapFill** property. **BitmapFill** property has sub-properties to mirror the image vertically and horizontally.

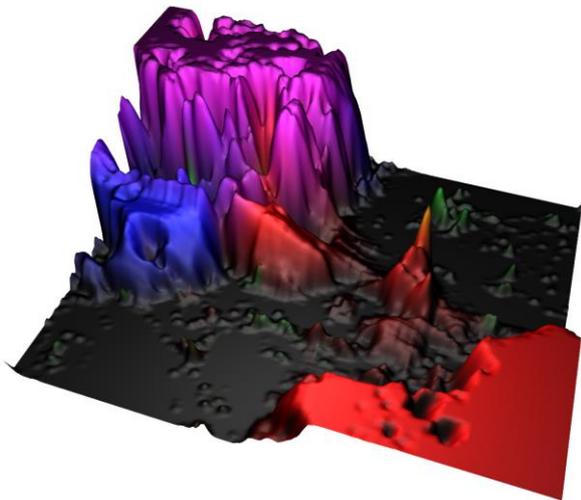


Figure 6-30. FromSurfacePoints fill. Color per data point.

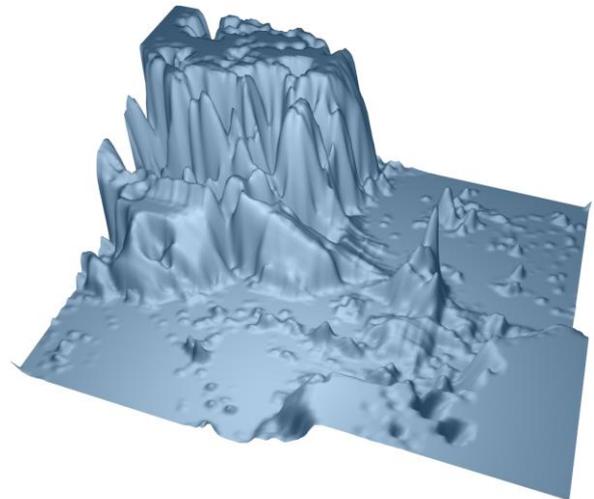


Figure 6-29. Toned fill.

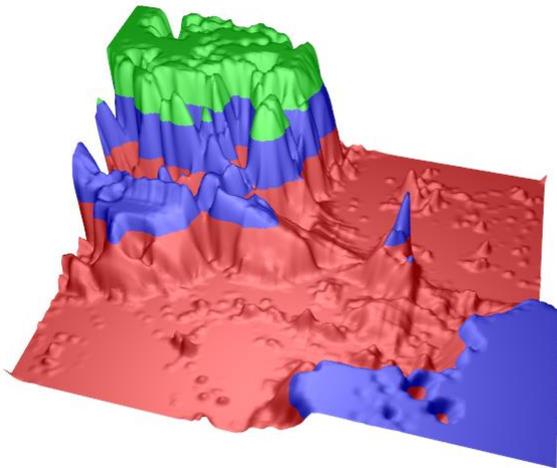


Figure 6-31. PalettedByY



Figure 6-32. Bitmap fill.

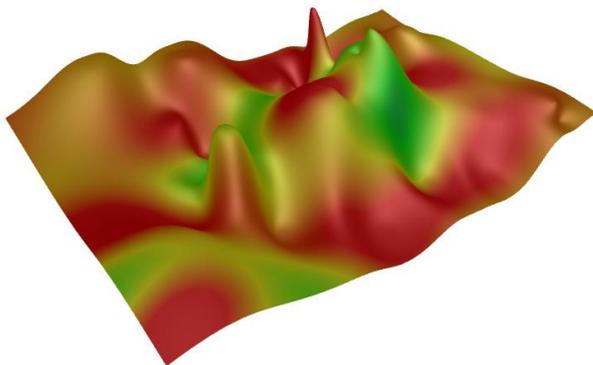


Figure 6-33. PalettedByValue.

## 6.9.4 Contour palette

With **ContourPalette** property, you can define color steps for height coloring. **ContourPalette** can be used for

- **Fill** (see section 6.9.3)
- **Wireframe mesh** (see section 6.9.5)
- **Contour lines** (see section 0)

For contour palette, you can define unlimited count of steps. Each step has a height value and the corresponding color.

The palette is defined with **MinValue**, **Type** and **Steps** properties. For **Type**, there's two choices: **Uniform** and **Gradient**. The contour palette of the previous figure (pay attention the legend box):

- **MinValue**: 0
- **Type**: Uniform
- **Steps**:
  - Steps[0]: MaxValue:25, Color: Red
  - Steps[1]: MaxValue:50, Color: Blue
  - Steps[2]: MaxValue:75, Color: Lime
  - Steps[3]: MaxValue:100, Color: White

The height values below first step value are colored with first step's color.

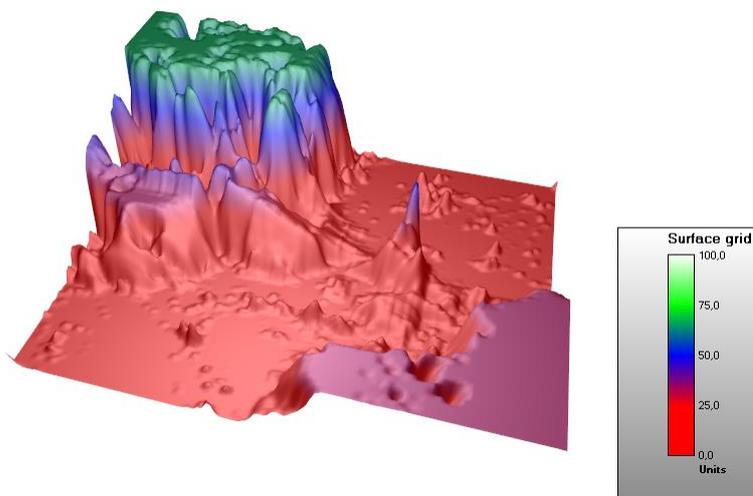


Figure 6-34. Surface grid series contour palette **Type** is set to **Gradient**.

## 6.9.5 Wireframe mesh

Use **WireframeType** to select the wireframe style. The options are:

- **None**: no wireframe
- **Wireframe**: a solid color wireframe. Use **WireframeLineStyle.Color** to set the color
- **WireframePalettedByY**: the wireframe coloring follows SurfacePoint's **Y** field **ContourPalette** (see section 6.9.4)
- **WireframePalettedByValue**: the wireframe coloring follows SurfacePoint's **Value** field, **ContourPalette** (see section 6.9.4)
- **WireframeSourcePointColored**: the wireframe coloring follows the color of surface nodes
- **Dots**: solid color dots are drawn in the node positions
- **DotsPalettedByY**: dots are drawn in the node positions, and colored by **ContourPalette**, by **Y** field of SurfacePoints
- **DotsPalettedByValue**: dots are drawn in the node positions, and colored by **ContourPalette**, by **Value** field of SurfacePoints
- **DotsSourcePointColored**: dots are drawn in the node positions, coloring follows the color of surface nodes

The wireframe line style (color, width, pattern) can be edited by using **WireframeLineStyle**.

**Note!** Palette colored wireframe lines and dots are available only when **WireframeLineStyle.Width = 1** and **WireframeLineStyle.Pattern = Solid**.

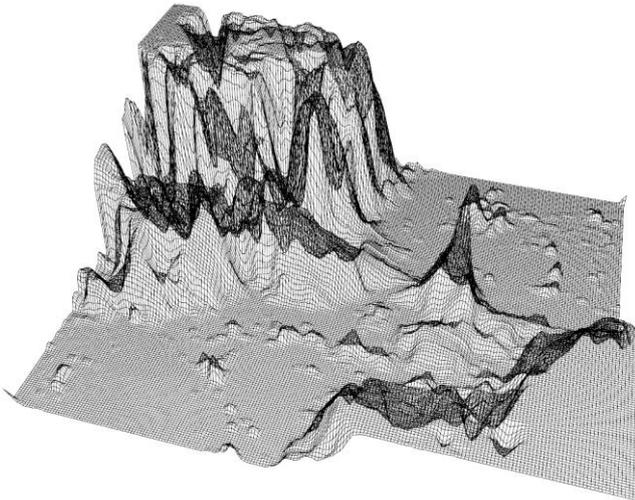


Figure 6-36. WireframeType = Wireframe.

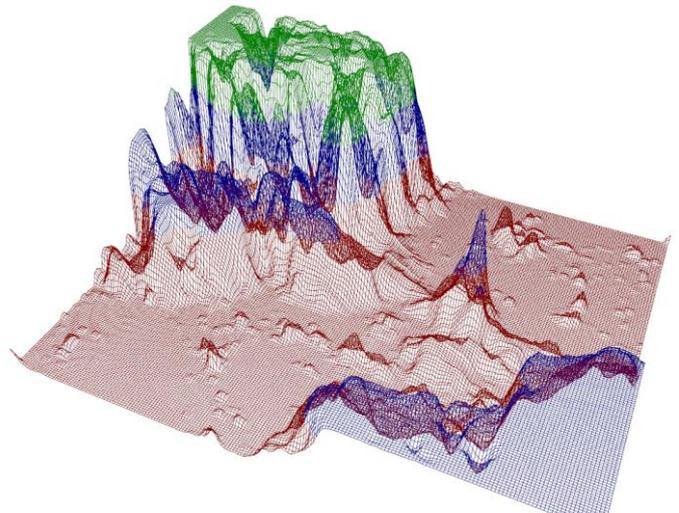


Figure 6-35. WireframeType = WireframePalettedByY.

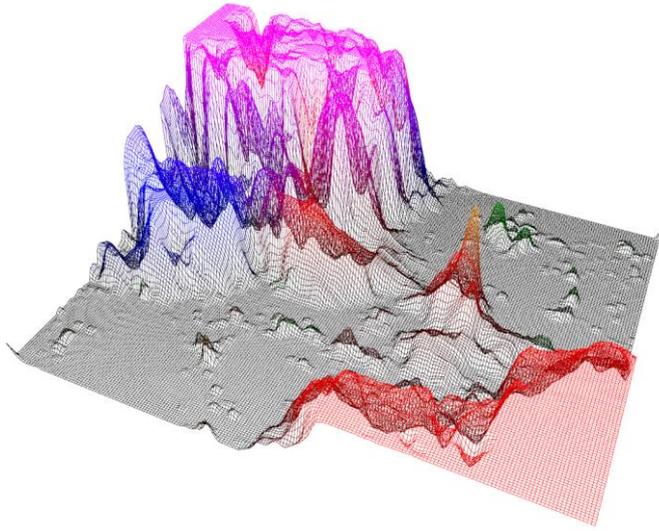


Figure 6-38. WireframeType = SourcePointColored.

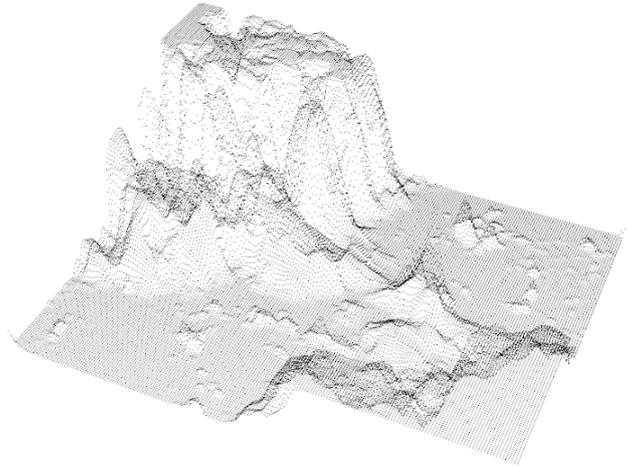


Figure 6-37. WireframeType = Dots.

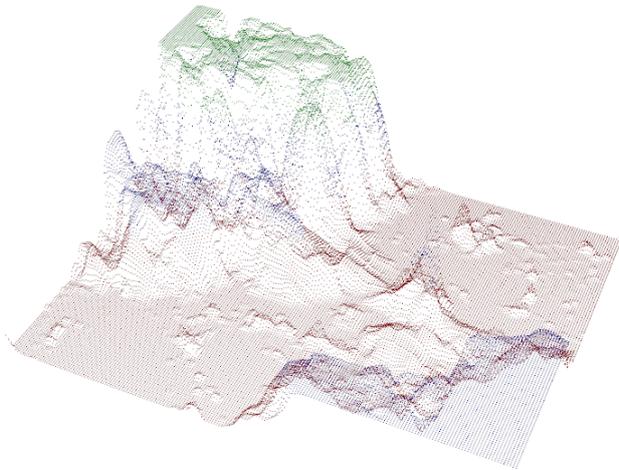


Figure 6-39. WireframeType = DotsPalettedByY.

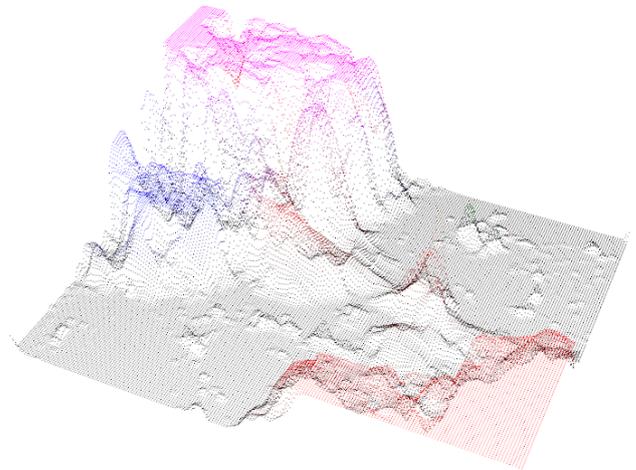


Figure 6-40. WireframeType = DotsSourcePointColored.

### 6.9.5.1 Some notes when using wireframe simultaneously with fill

When fill and wireframe are drawn in same position in 3D model, *Z-fighting* may appear. It can be seen as broken wireframe lines. That is because it's impossible for GPU to determine which object is closer to camera.

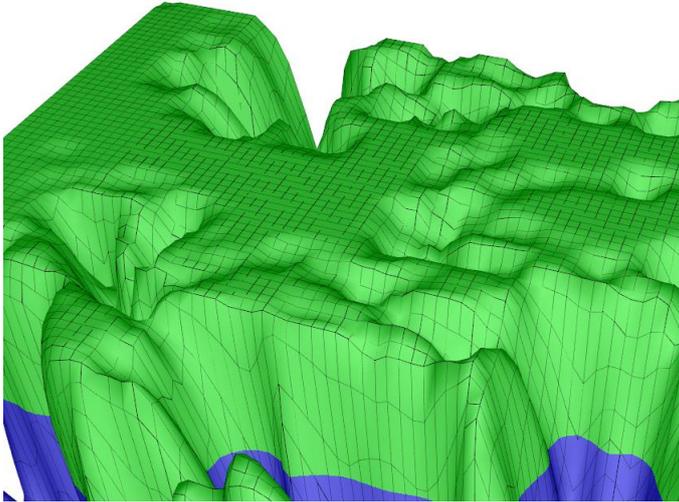


Figure 6-41. Surface grid wireframe with filling. Z-fighting appears as broken wireframe lines.

To prevent Z-fighting from occurring, use **WireframeOffset** or **DrawWireframeThrough** property. By using **WireframeOffset** the wireframe is shifted slightly towards some direction in 3D model space. **DrawWireframeThrough** draws the wireframe through the filling, whether or not the part of the surface is visible to the camera.

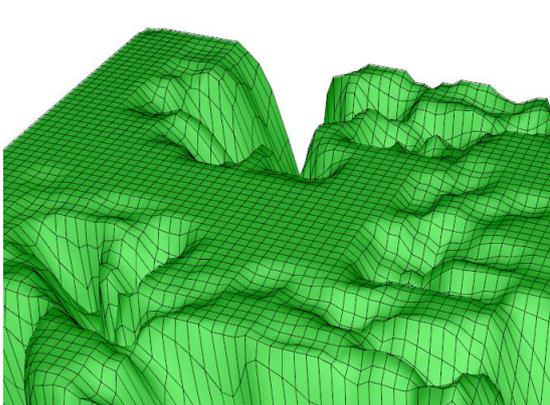


Figure 6-42. WireframeOffset = (X=0; Y=0.1; Z=0).

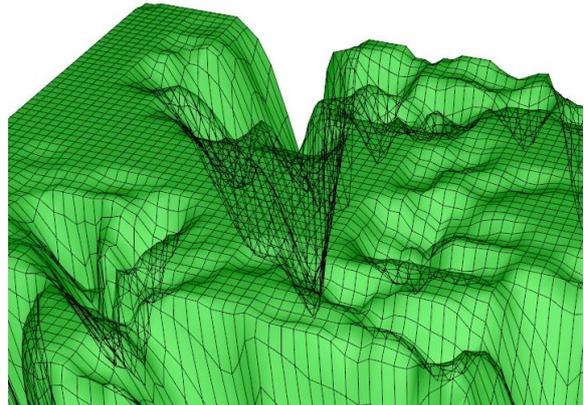


Figure 6-43. DrawWireframeThrough is enabled.

## 6.9.6 Contour lines

Contour lines allow quick interpretation of height data without filling the surface with paletted fill. Contour lines can be used combined with fill and wireframe. By setting **ContourLineStyle** property, contour lines can be drawn with different styles:

- **None**: no contour lines are shown
- **FastColorZones**: The lines are drawn as thin vertical zones. Very powerful rendering, suits very well for continuously updated or animated surface. Steep height changes are shown as thin line, as gently sloping height differences are shown with thick line. All lines use same color defined with **ContourLineStyle.Color** property. The zone height can be set by **FastContourZoneRange** property.
- **FastPalettedZones**: Like **FastColorZones**, but line coloring follows **ContourPalette** options (see section 6.9.4)
- **ColorLineByY and ColorLineByValue**: Contour lines are made with actual lines. Rendering takes longer than **FastColorZones**. The line width can be adjusted with **ContourLineStyle.Width** property. Contour line can be shifted with **WireframeOffset** property, to remove possible Z-fighting with filling.
- **PalettedLineByY and PalettedLineByValue**: Like **ColorLineByY and ColorLineByValue**, but line coloring follows **ContourPalette** options (see section 6.9.4)

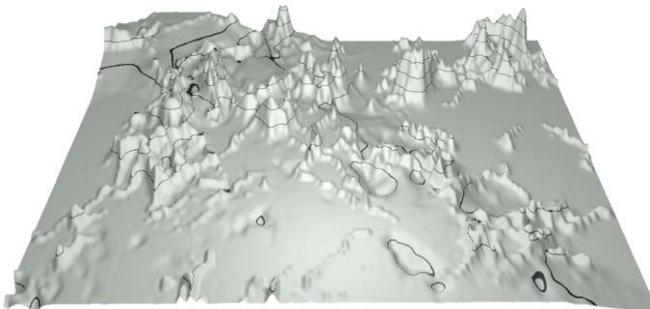


Figure 6-44. ContourLineStyle = FastColorZones.

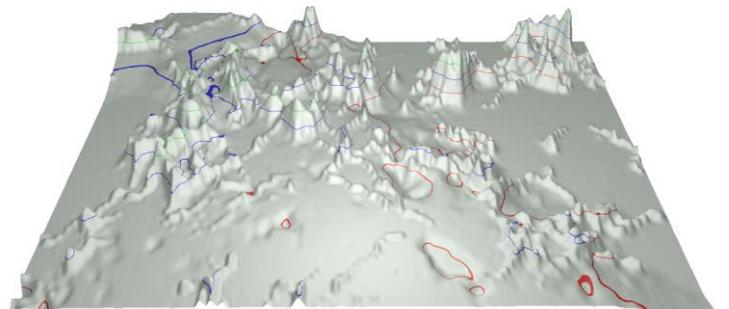


Figure 6-45. ContourLineStyle = FastPalettedZones.

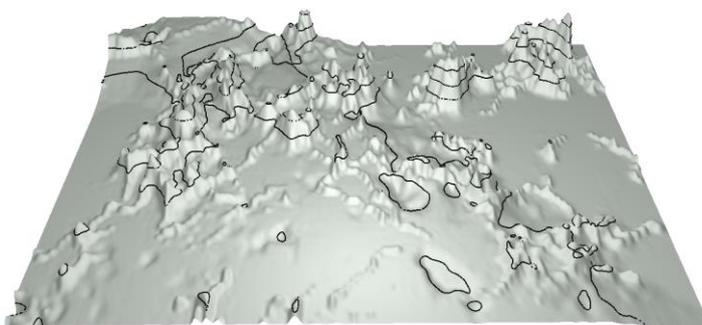


Figure 6-46. ContourLineStyle = ColorLine.

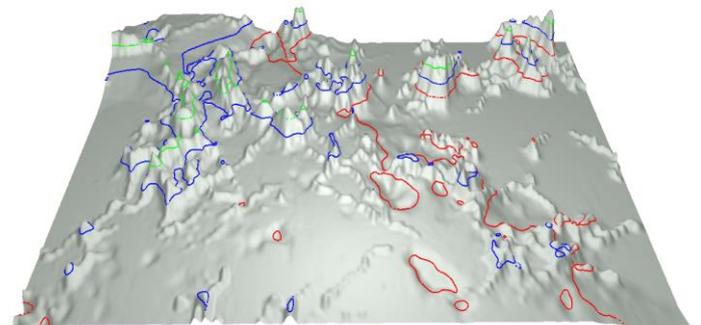


Figure 6-47. ContourLineStyle = PalettedLine.

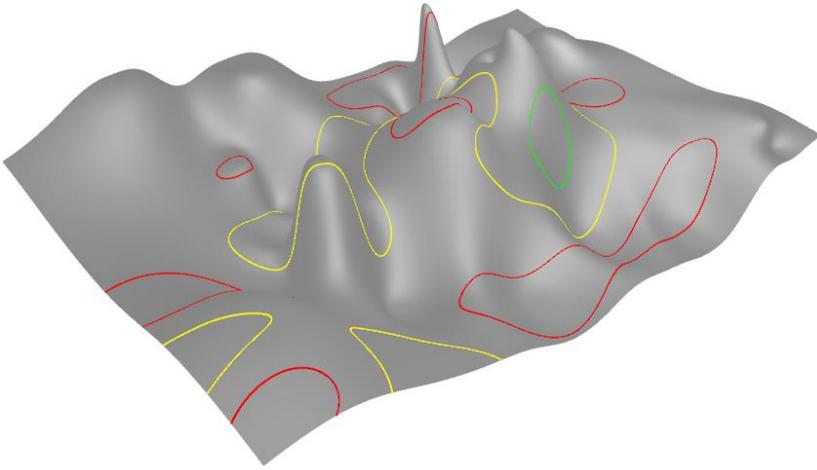


Figure 6-48. `ContourLineType = PalettedLineByValue`.

### 6.9.7 Scrolling surface data

`SurfaceGridSeries3D` and `SurfaceMeshSeries` has ***InsertRowBackAndScroll*** and ***InsertColumnBackAndScroll*** methods for performance optimized periodical data adding into last column or last row. Consider the

following 3D spectrum display. New FFT values are added as last row (close to camera), and the old data and time axis (Z axis) must be scrolled. The oldest surface values must be dropped off.

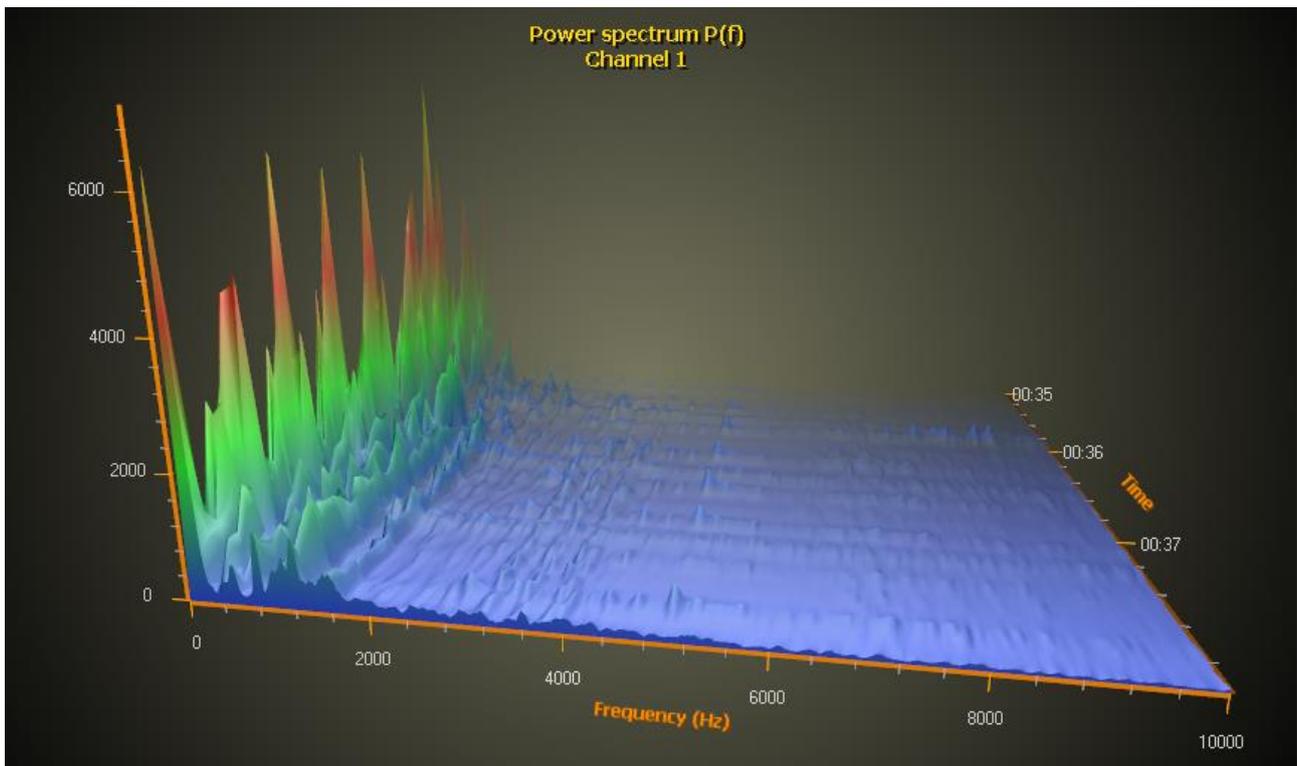


Figure 6-49. Presenting 3D spectrum with surface grid. InsertRowBackAndScroll method is used for performance optimized data adding. Fadeaway property is 100 to make the surface smoothly fade away towards back. A perspective camera is used.

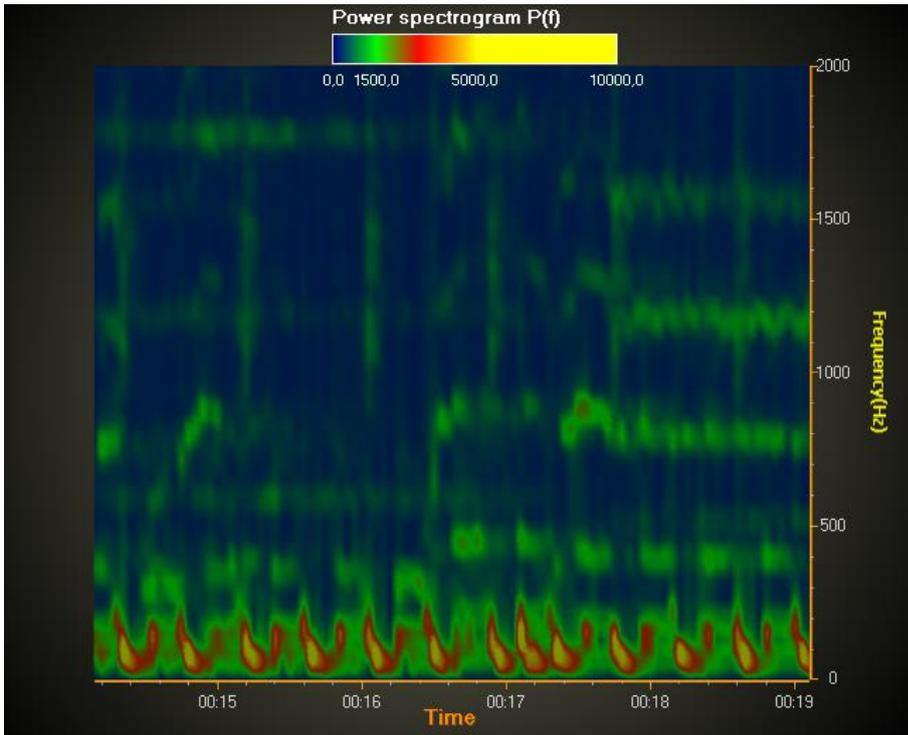


Figure 6-50. Presenting spectrogram with surface grid. InsertColumnBackAndScroll method is used. An orthographic camera above the model is used to give straight and perpendicular projection. SuppressLighting is enabled for removing the light reflections. Fadeaway = 0 for making the grid series fully visible.

## 6.10 SurfaceMeshSeries3D

**SurfaceMeshSeries3D** is almost similar to **SurfaceGridSeries3D**. The biggest difference is that surface nodes can be positioned freely in 3D space. The surface does not have to be rectangular. **SurfaceMeshSeries3D** allows warping the surface virtually to any shape, like a sphere or human head.

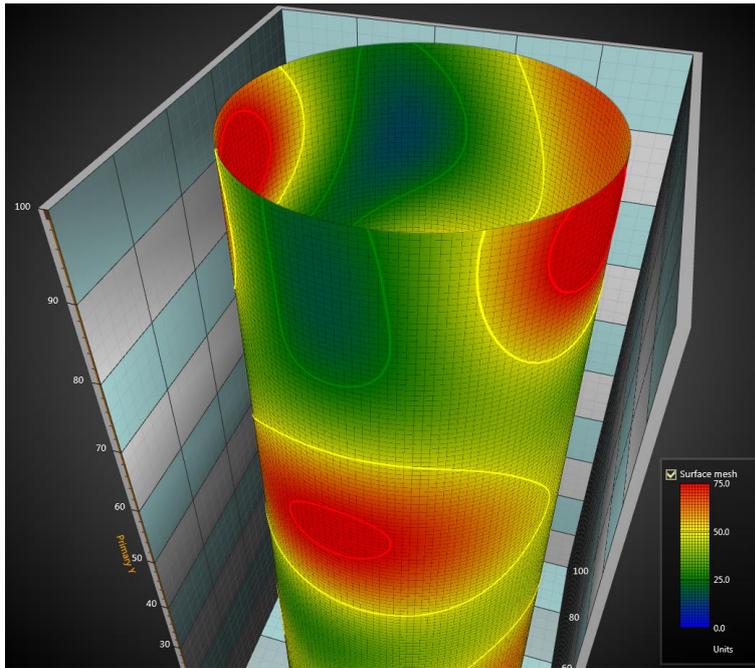


Figure 6-51. SurfaceMeshSeries3D, geometry made as a pipe.

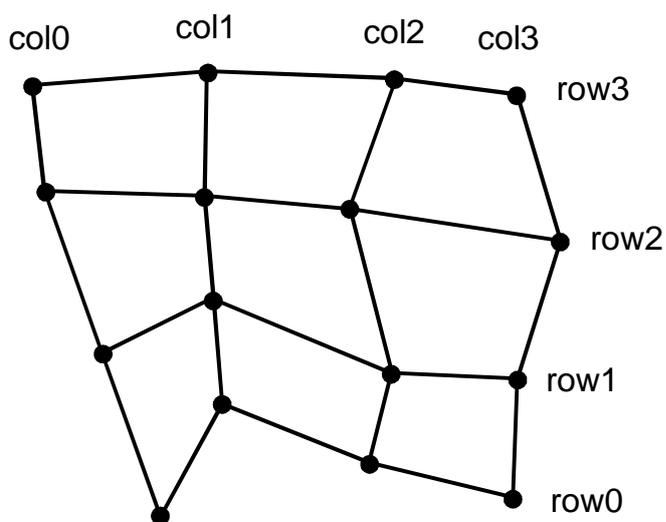


Figure 6-52. Surface mesh nodes. SizeX = 4, SizeZ = 4.

### 6.10.1 Setting surface mesh data

- Set **SizeX** and **SizeZ** properties to give the grid a size as columns and rows.
- Set X, Y and Z values for all nodes:

#### Method, with Data array index

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexZ = 0; nodeIndexZ < rowCount; nodeIndexZ ++)  
    {  
        meshSeries.Data[nodeIndexX, nodeIndexZ].Y = xValue;  
        meshSeries.Data[nodeIndexX, nodeIndexZ].Y = yValue;  
        meshSeries.Data[nodeIndexX, nodeIndexZ].Z = zValue;  
        meshSeries.Data[nodeIndexX, nodeIndexZ].Value = dataValue;  
    }  
}  
meshSeries.InvalidateData(); //Notify new values are ready to refresh
```

#### Alternative method, usage of SetDataValue

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexZ = 0; nodeIndexZ < rowCount; nodeIndexZ ++)  
    {  
        meshSeries.SetDataValue(nodeIndexX, nodeIndexZ,  
            xValue,  
            yValue,  
            zValue,  
            dataValue,  
            Color.Green); //Source point colors are not used in this  
                           example, so use any color here  
    }  
}  
meshSeries.InvalidateData(); //Notify new values are ready to refresh
```

## 6.10.2 Visualizing point clouds in 3D

SurfaceMeshSeries3D can be used also for randomly placed points cloud visualization, **up to millions of points**.

- Set the points in the **Data** array, so that the data array is at least 2x2 of size.
- Set **Fill = None, ContourLineType = None**
- Set **WireFrameType = DotsPalettedByY or DotsPalettedByValue**
- Set the point size in pixels in **WireFrameLineStyle.Width**

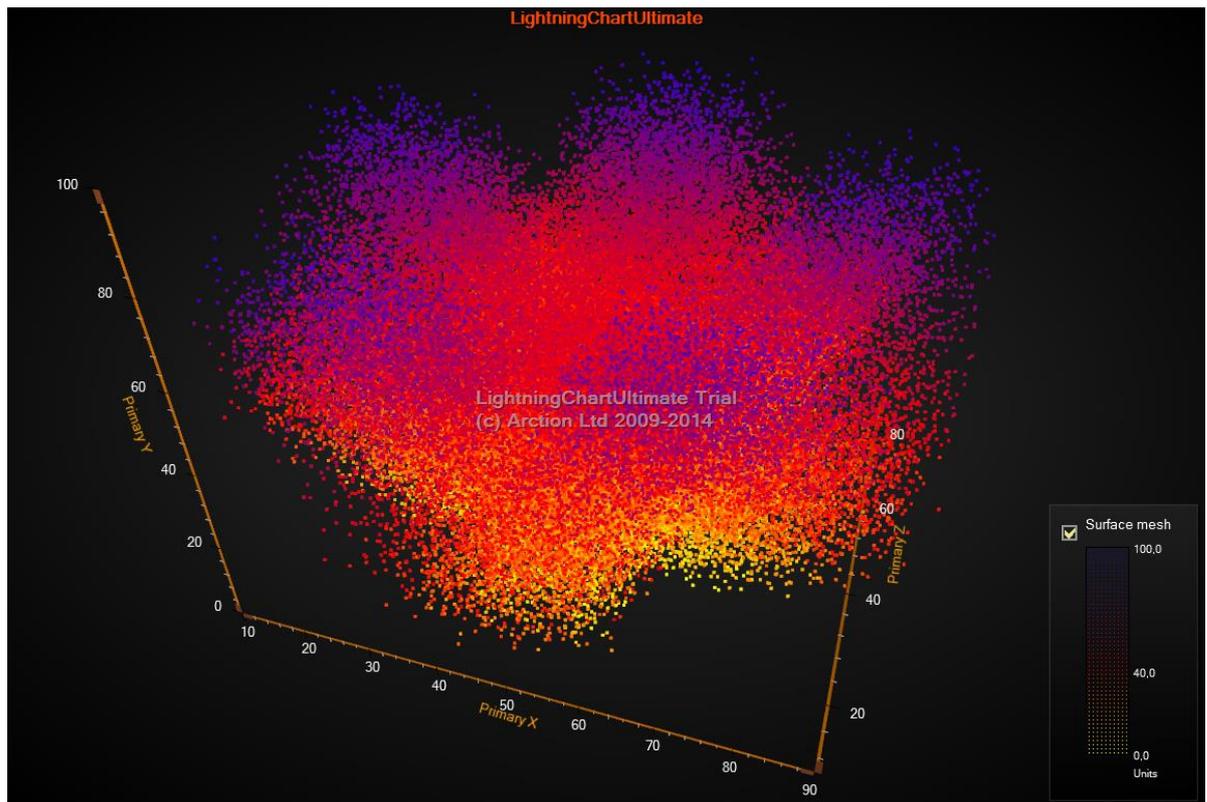


Figure 6-53. Point cloud visualization with palette coloring of points.

## 6.11 WaterfallSeries3D

With **WaterfallSeries3D**, the data is visualized in area strips. Areas can be filled, wire-framed and contour-lined like **SurfaceGridSeries3D**, see section 6.9. In Y-dimension, area starts from **BaseLevel** property value. The node data can be set like in **SurfaceMeshSeries3D**, see section 6.10.1.

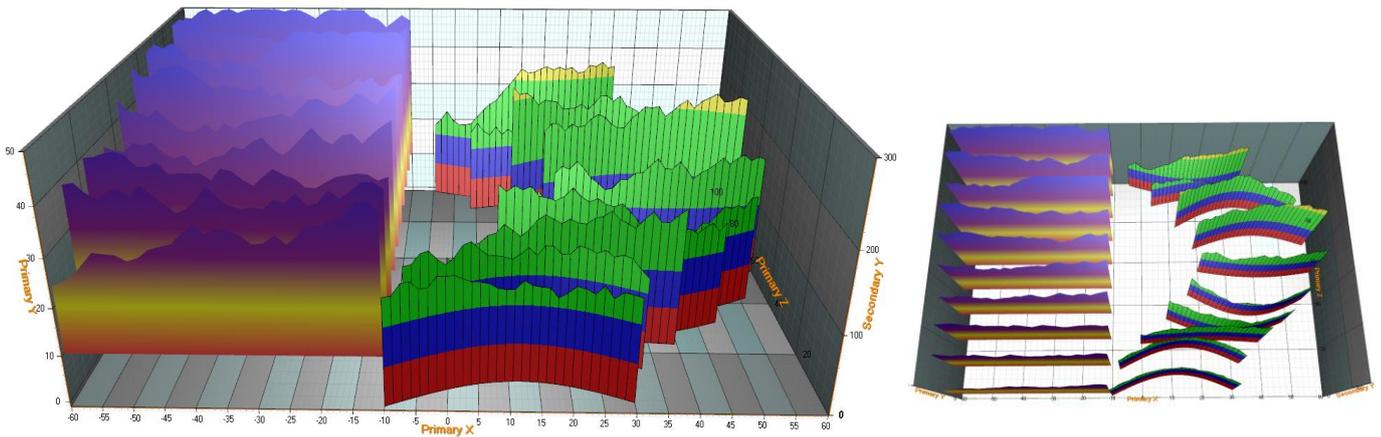


Figure 6-54. Two waterfall series. On the left violet series, X and Z are in rectangular form. BaseLevel = 10. On the right red-green-blue series, X and Z values are bent, and each row is placed in different horizontal location.

WaterfallSeries3D is especially handy for presenting traditional 3D spectrum.

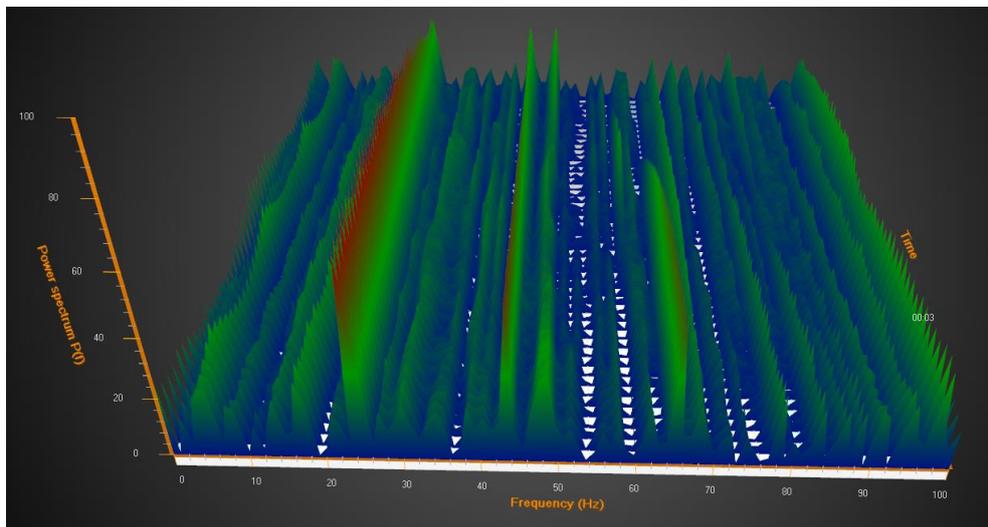


Figure 6-55. Waterfall series used for traditional spectrum presentation.

## 6.12 BarSeries3D

**BarSeries3D** allow bar data visualization in 3D.

### 6.12.1 Bars grouping

Bar series can be grouped with many options available in **BarViewOptions** property of **View3D**. **BarViewOptions.ViewGrouping** controls how the bars are grouped in the 3D view.

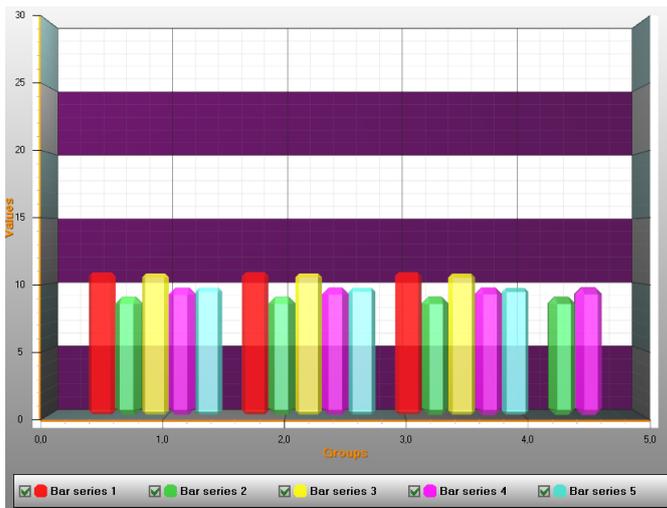


Figure 6-56. **BarViewOptions.ViewGrouping = GroupedIndexedFitWidth**. Bar widths and group gaps are arranged to fit the width nicely.

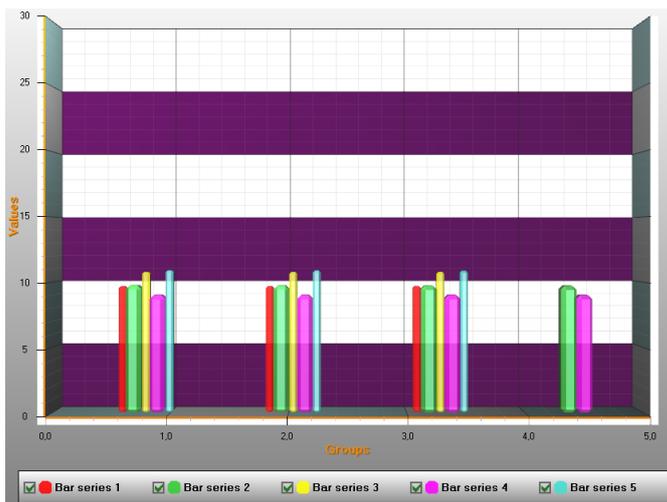


Figure 6-57. **BarViewOptions.ViewGrouping = GroupedIndexed**. Original bar widths apply and groups are arranged to fit the chart width.

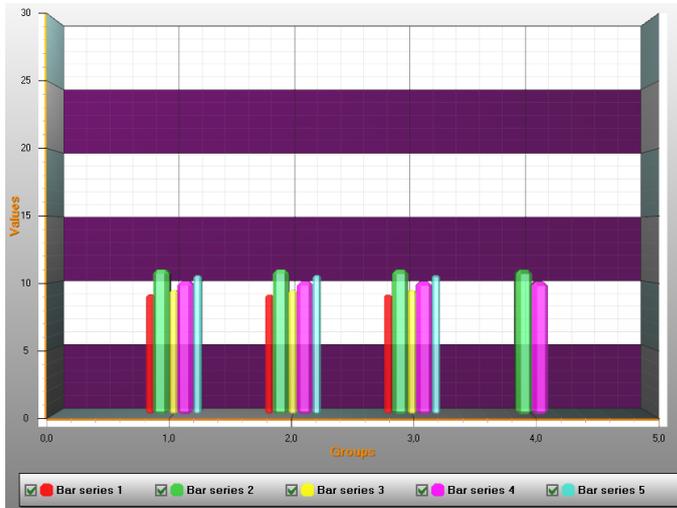


Figure 6-58. `BarViewOptions.ViewGrouping = GroupedByXValue`. Bar X values apply.



Figure 6-59. `BarViewOptions.ViewGrouping = StackedIndexed`. All bars having same index are stacked.

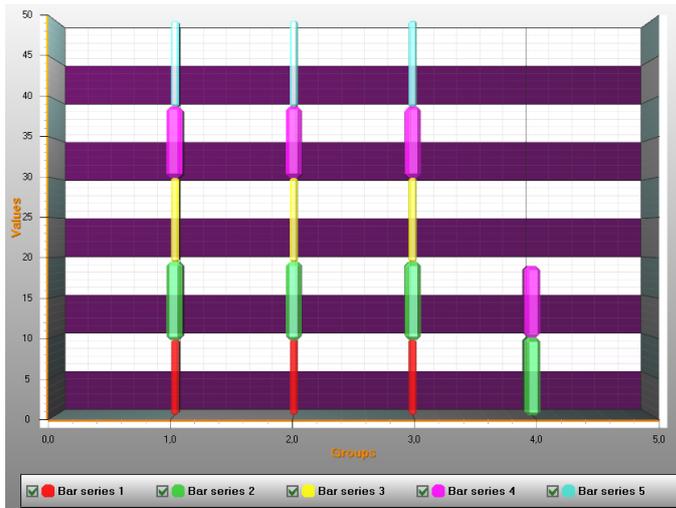


Figure 6-60. BarViewOptions.ViewGrouping = StackedByXValue. All bars having same X value are stacked. This example picture looks same than with StackedIndexed, as the X values and indices are same.

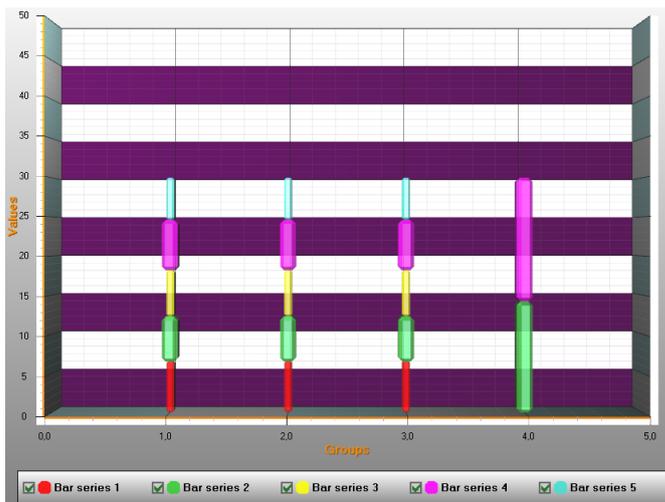


Figure 6-61. BarViewOptions.ViewGrouping = StackedStretchedToSum. All bars having same X value are stacked and stretched to StackSum.

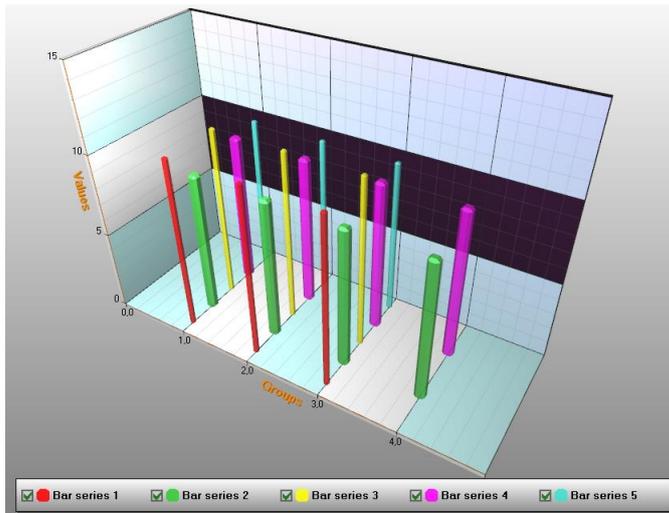


Figure 6-62. `BarViewOptions.ViewGrouping = Manhattan`. The first series values are shown nearest to camera and the last series farthest. Bar X values control the bar position in X dimension.

### 6.12.2 Bar styles

`BarSeries3D` has **Shape** property for controlling the bar shape. Furthermore, with some shapes, you can use **CornerPercentage** to change corner rounding and **DetailLevel** to change the visual quality.

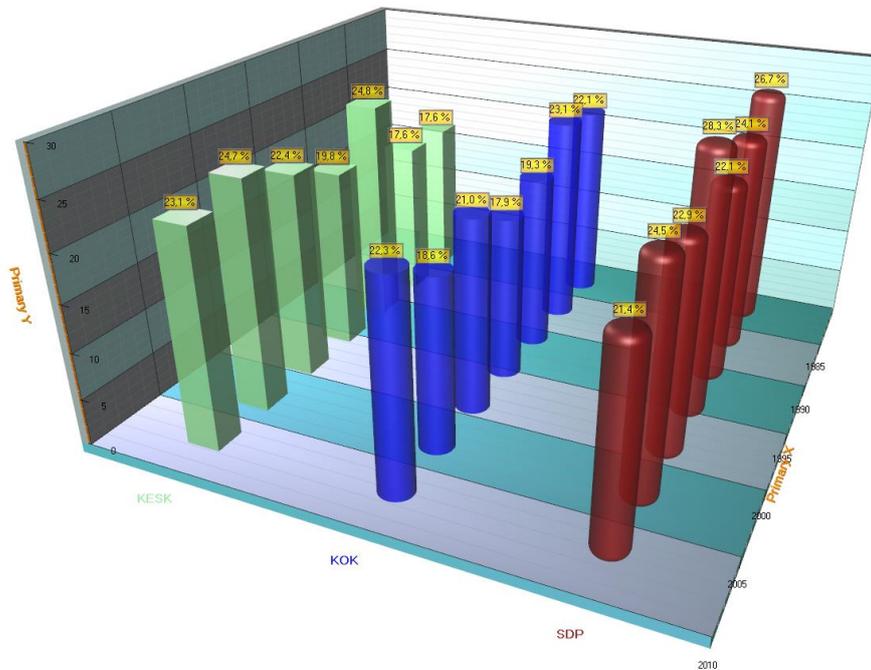


Figure 6-63. Shapes: Simple, Cylinder and RoundedCylinder.

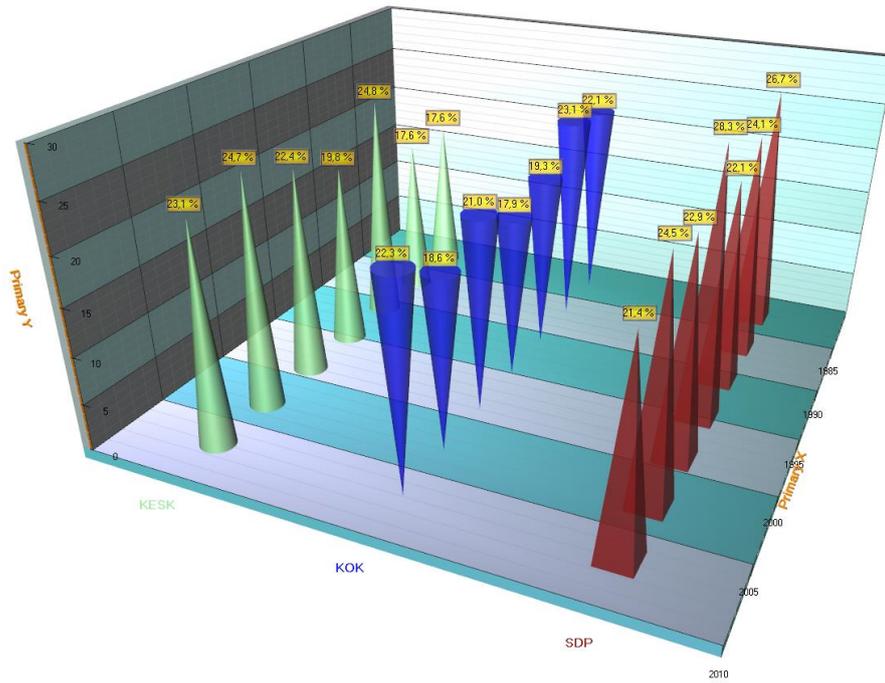


Figure 6-64. Shapes: Cone, ReversedCone and Pyramid.

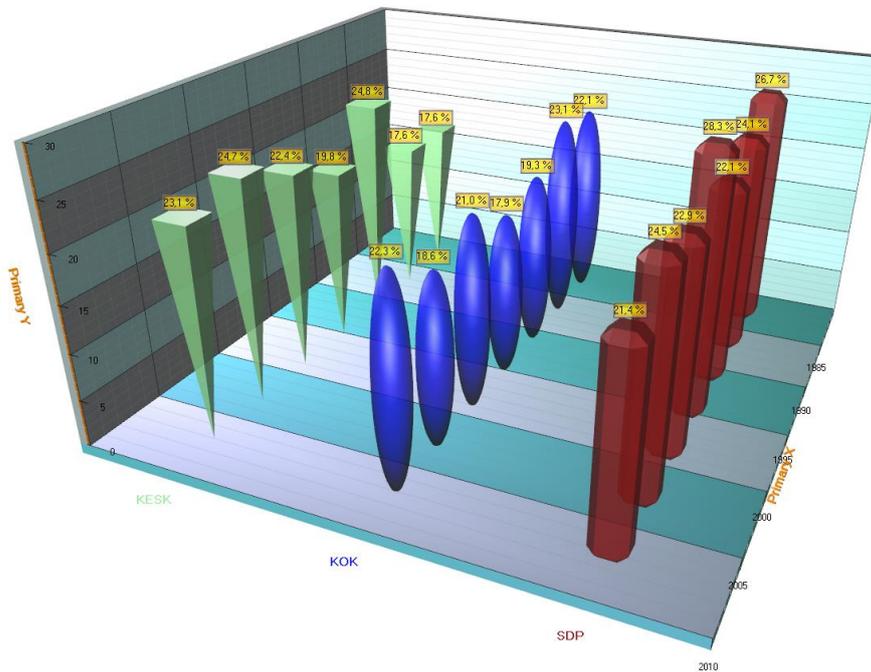


Figure 6-65. Shapes: ReversedPyramid, Ellipsoid and Beveled.

### 6.12.3 Setting bar series data

Bar series data can be added like this:

```
// create new values array
BarSeriesValue3D[] values = new BarSeriesValue3D[3];
values[0] = new BarSeriesValue3D(20, 45, 5, "");
values[1] = new BarSeriesValue3D(30, 50, 5, "");
values[2] = new BarSeriesValue3D(40, 35, 5, "");

// add values to series
chart.View3D.BarSeries3D[0].AddValues(values, false);
```

### 6.12.4 Showing bars horizontally

Bars are drawn in Y axis direction. To show the bars vertically, rotate the camera 90 degrees.

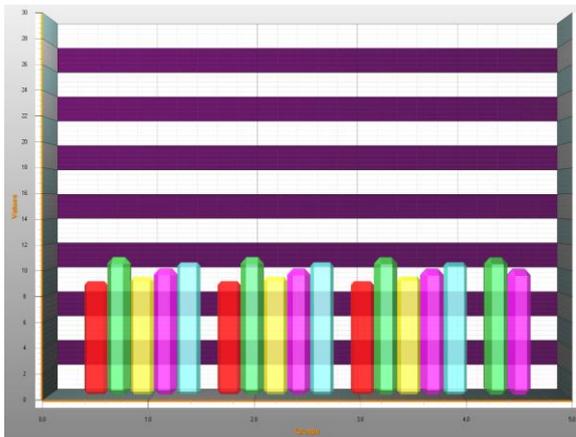


Figure 6-66. Vertical bars view.

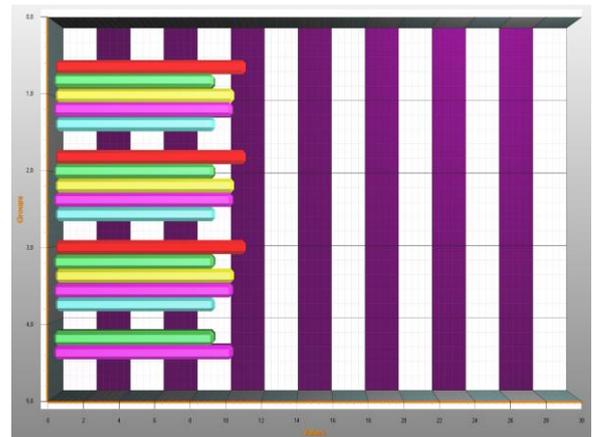


Figure 6-67. Horizontal bars view.

This code sets up the previous figure **vertical bars** view.

```
chart.BeginUpdate();
chart.View3D.Camera.RotationX = 0;
chart.View3D.Camera.RotationY = 0;
chart.View3D.Camera.RotationZ = 0;
chart.View3D.Camera.ViewDistance = 170;
chart.View3D.YAxisPrimary3D.Location = AxisYLocation3D.FrontLeft;
chart.View3D.Dimensions.Y = 100;
chart.View3D.Dimensions.X = 150;
chart.EndUpdate();
```

And the code sets up the previous figure **horizontal bars** view.

```
chart.BeginUpdate();
chart.View3D.Dimensions.Y = 150;
chart.View3D.Dimensions.X = 100;
chart.View3D.YAxisPrimary3D.Location = AxisYLocation3D.FrontRight;
chart.View3D.Camera.RotationX = 0;
chart.View3D.Camera.RotationY = 0;
chart.View3D.Camera.RotationZ = 90;
chart.View3D.Camera.ViewDistance = 170;
chart.EndUpdate();
```

## 6.13 MeshModels

**MeshModels** list property allows inserting 3D models from external 3D model editors into LightningChart View3D. The models can be imported in OBJ format, a generic format in 3D modeling applications and game engines.

**NOTE!** LightningChart v.7 onwards does not support Direct3D X-format files (\*.x) anymore, since DirectX 11 does not support it.

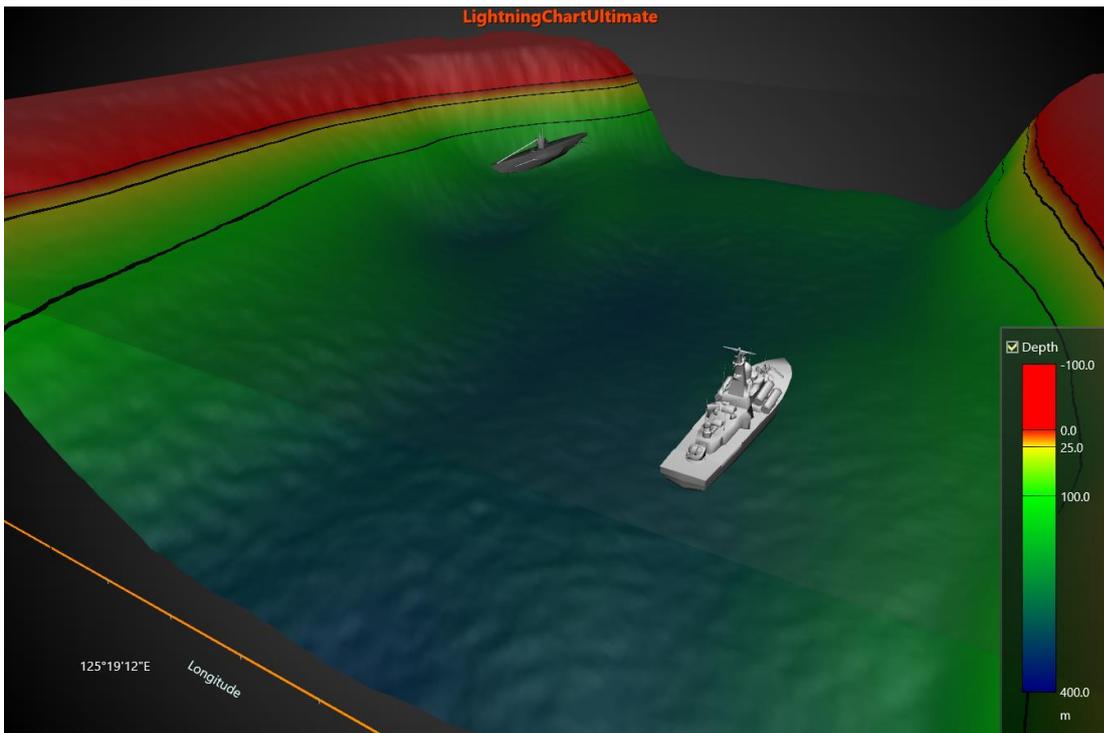


Figure 6-68. Battleship and submarine models loaded in View3D, over SurfaceGridSeries3D visualizing seabed depth data.

### 6.13.1 Loading a model

- To load a model from file, set the path and file name into **ModelFileName** property, or use **LoadFromFile** method. When loading the model from file, texture fills are loaded too, if they exist in the same path and MTL file and image files are accessible.
- To load model from stream, use **LoadFromStream** method. The stream reading method only reads geometry and materials, but not textures.
- To load model from a resource, use **LoadFromResource** method.

### 6.13.2 Positioning, scaling and rotating the model

A **MeshModel** object **Position** follows the X, Y and Z axes it has been assigned to. You can rotate the model by editing **Rotation** property. **Size** can be defined with **Size** property, which is a collection of factors for original model size, and does not follow axis ranges or 3D world dimensions.

### 6.13.3 Enabling fill and wireframe

- To show fill, set **Fill** = True
- To show wireframe, set **WireFrame** = True, and set preferred line color in **WireFrameLineColor**.

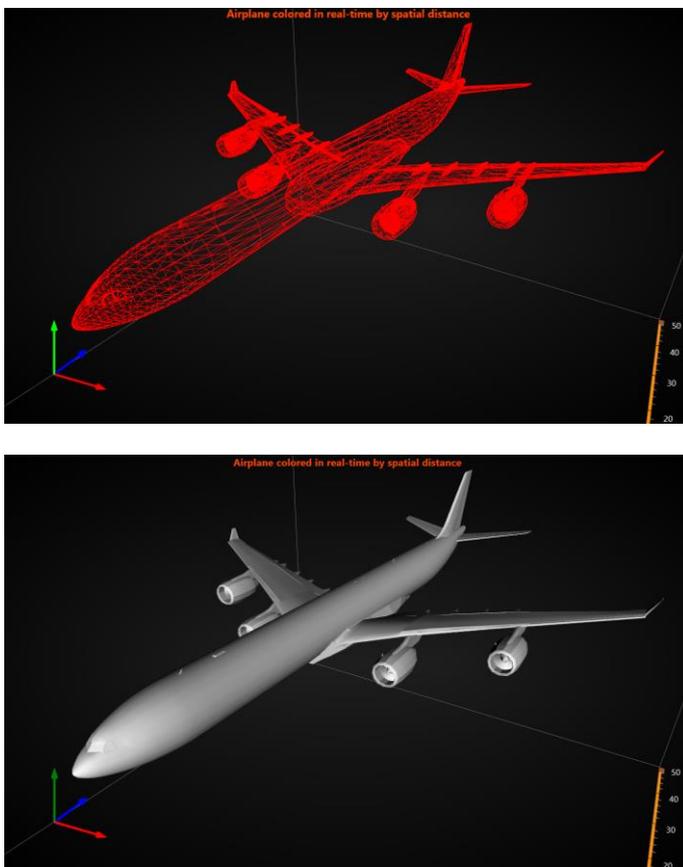


Figure 6-69. Airplane shown as wireframe (**WireFrameLineColor** = Red), and with default gray fill.

### 6.13.4 Custom-coloring fill

By default, the model renders with the colors in the OBJ model. To apply custom coloring for model's vertices, use ***UpdateFillColors(int[] colors)*** method. This method can be called also periodically, to apply real-time color updates.

***GeometryConstructed*** event reports position of vertices in *axis values space*, in as ***X, Y*** and ***Z*** arrays. They are especially needed when applying coloring e.g. by spatial distance of other chart objects, such as data points. Subscribe to ***GeometryConstructed*** event handler in the initialization phase, and then unsubscribe when not needed anymore.

***UpdateFillColors*** requires ARGB ***colors*** array that is equal length of vertex positions (***X.Length***). One color for each vertex.

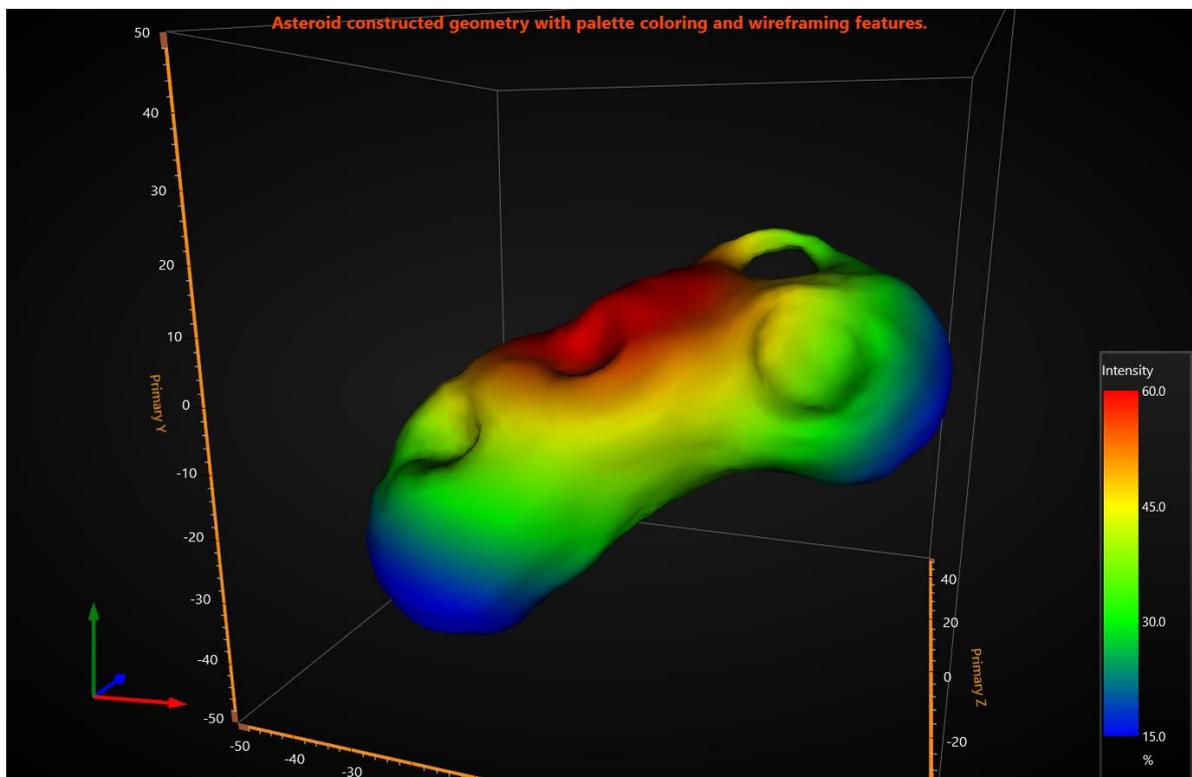


Figure 6-70. MeshModel colored by spatial distance utilizing ***UpdateFillColors*** method.

NOTE: You can utilize ***ChartTools.ConvertDataToColorsByFixedIntervalPalette*** method to convert data values into colors (ARGB int), by your given palette steps.

### 6.13.5 Custom-coloring wireframe

In similar way, you can color the wireframe with custom colors.

Use ***GeometryConstructed*** event handler to learn required colors array length, and use ***UpdateWireframeColors*** method to apply new colors.

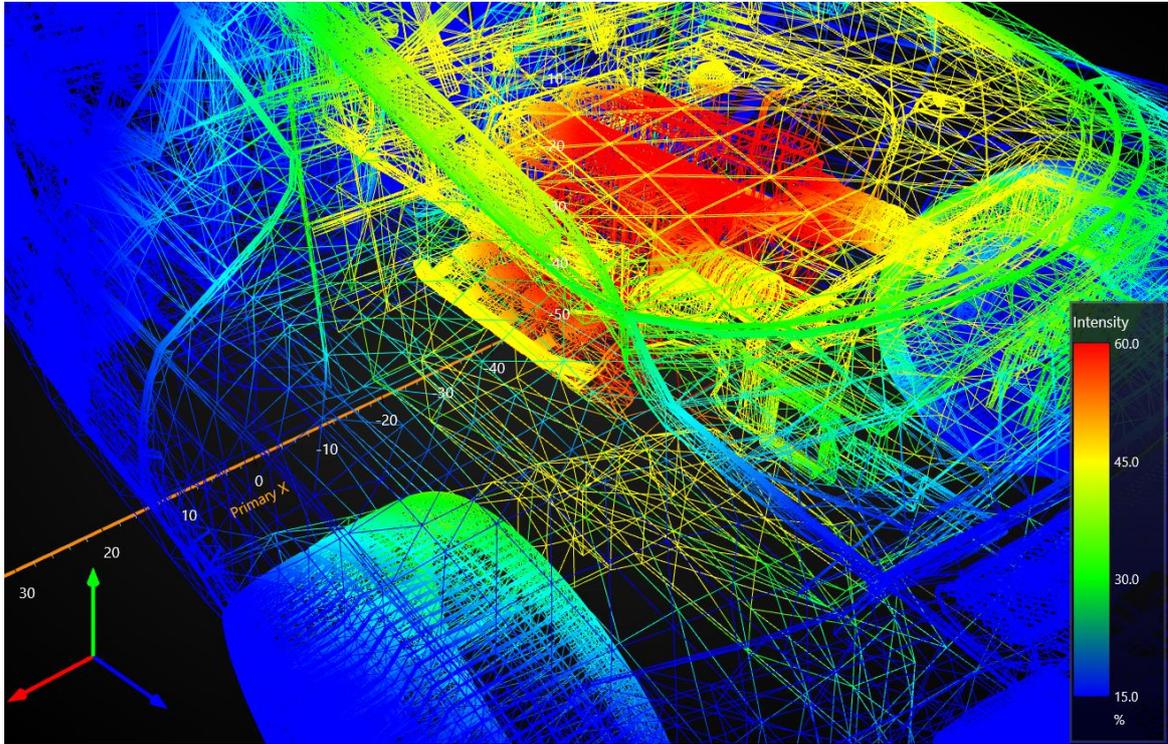


Figure 6-71. MeshModel wireframe colored by spatial distance utilizing ***UpdateWireframeColors*** method.

### 6.13.6 Reverse vertices winding order

Some models are made with reverse winding order and therefore culling makes them invisible. If your model does not show up correctly, change ***ClockwiseVertexWinding*** setting.

### 6.13.7 Constructing MeshModel programmatically from vertices

Starting from v.8.2, **MeshModel** supports constructing the **MeshModel** geometry programmatically. It allows visualizing objects and shapes that have been produced as a result of computation.

The following **Create** methods are available

- *Create(positions, colors, indices)*
- *Create(positions, colors, normals, indices)*
- *Create(positions, textureCoordinates, bitmap, textureWrapMode, indices)*
- *Create(positions, normals, textureCoordinates, bitmap, textureWrapMode, indices)*

Index array (**indices**) parameters are optional. If provided they will define which vertices, colors, light normals and texture coordinates to use from the arrays given. Using indices saves resources when same vertices are shared between multiple triangles.

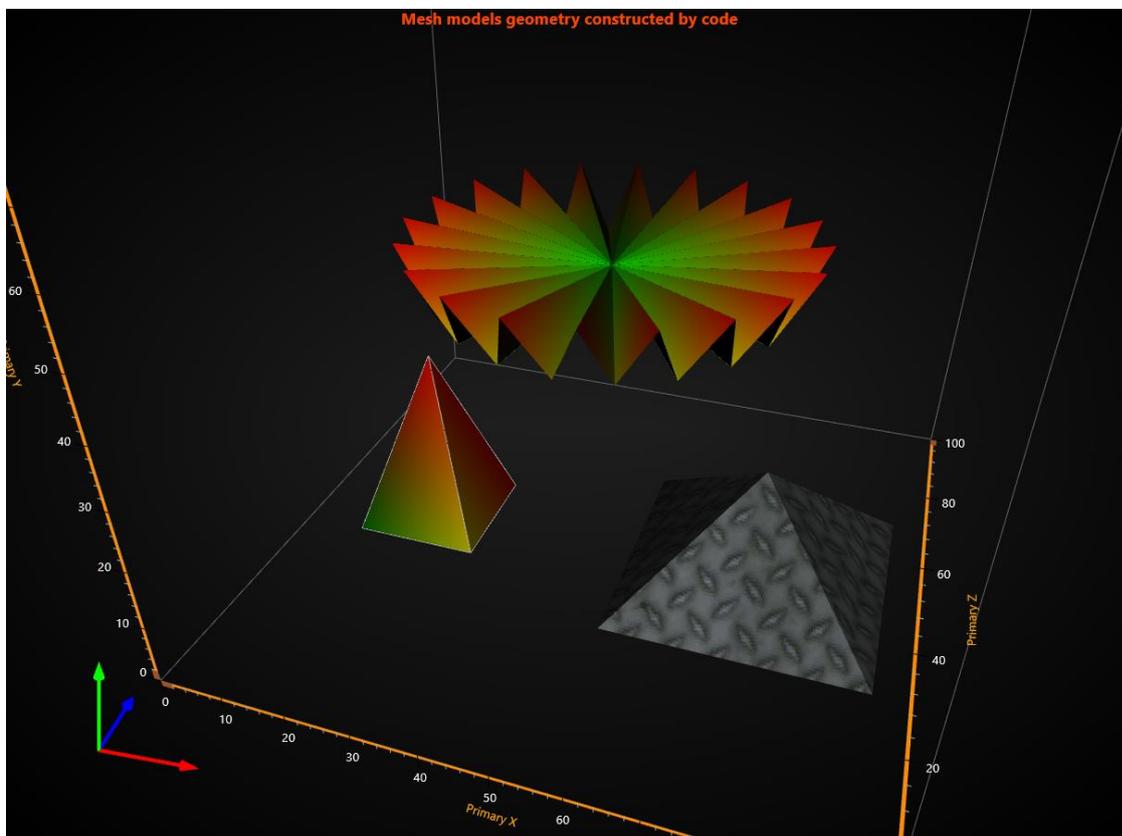


Figure 6-72. MeshModels constructed by code.

The rotation, scaling and positioning properties etc., as well as events, apply also for a **MeshModel** created programmatically from vertices, in a similar way than they work for loaded objects.

### 6.13.7.1 Updating the bitmap fill efficiently

When the **MeshModel** has been created by using **Create** method, and supplying bitmap and texture coordinates as arguments, it is possible to update the bitmap very efficiently without reconstructing the geometry. Call **UpdateFillBitmap** method to update.

**Note!** **UpdateFillBitmap** method is not applicable for models loaded from OBJ files.

### 6.13.8 Tracing the model with mouse

**MeshModel** has triangle-based tracing for mouse position.

Use **MouseTriangleTraced** event. It will indicate the **nearest** triangle to camera, where the mouse hits.

The event arguments have the following info:

- **IntersectionPointAxisValues**: intersection point of triangle face, in axes values
- **ModelSpaceTriangleCoordinates**: array of 3 triangle corners (vertices) the mouse is hitting, in 3D model space coordinates
- **WorldSpaceTriangleCoordinates**: array of 3 triangle corners (vertices) the mouse is hitting, in 3D world space coordinates.
- **NearestCoordinateIndex**: Index of nearest coordinate index of traced triangle, value of 0...2. Use the index to extract the coordinate from **ModelSpaceTriangleCoordinates** or **WorldSpaceTriangleCoordinates** array.

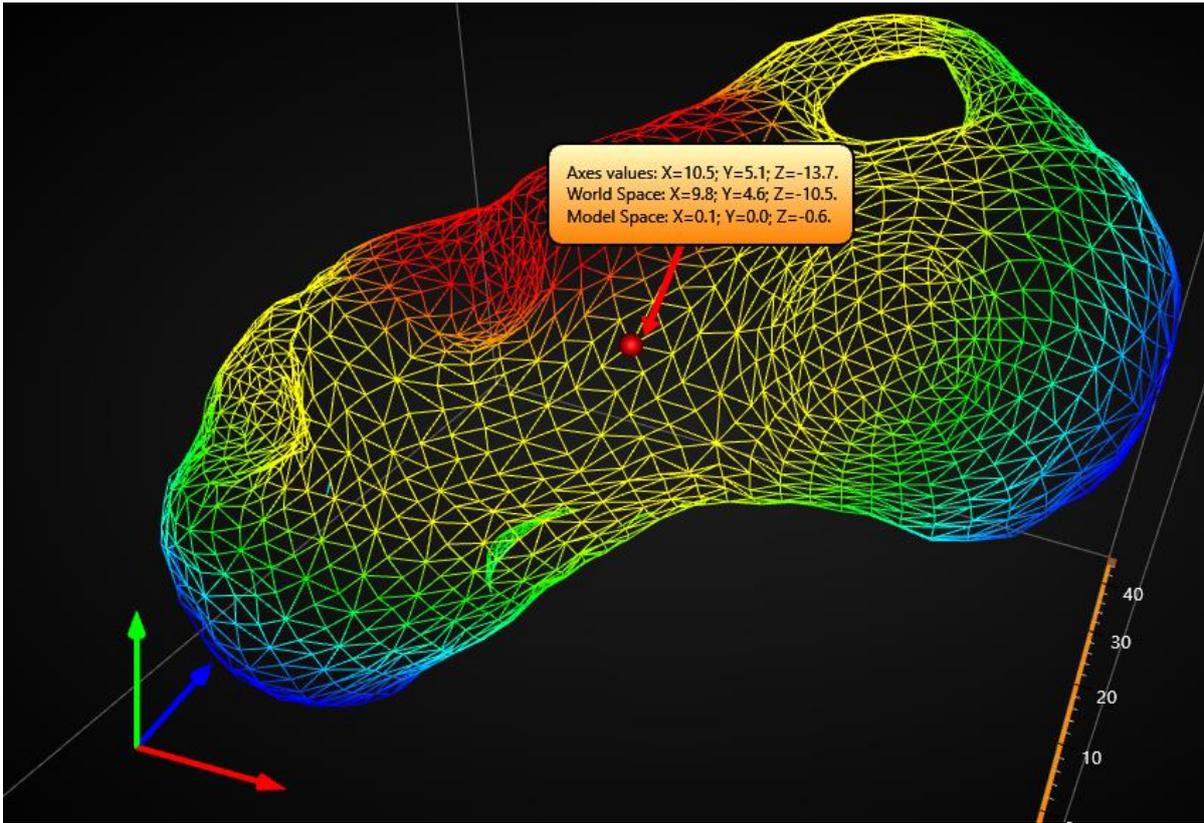


Figure 6-73. Tracing MeshModels with mouse. Traced result is shown in an Annotation.

## 6.14 VolumeModels

**VolumeModels** is a tool for volume data visualisation via Direct Volume Rendering. **VolumeModel** takes the volume data inside, and visualize it. LightningChart's volume rendering engine is based on the **Volume Ray Casting**.

An image is produced by the algorithm via the volume data sampling along the tracks of the rays which travel inside the dataset. A simple realisation of hardware acceleration for **Volume Ray Casting** requires generating boundaries for a volume object. Usually, they are represented by a cube. High rendering quality with the lack of artefacts, and usage of the interchangeable ray function are the main advantages of this technology.

**Ray Function** is the core of the algorithm, and provides it with a very high level of flexibility. The technique is very powerful because it specifies the way how the data is sampled and combined. This makes it a very useful tool for a feature extraction.

**Note!** **VolumeModels** are available only when **DirectX 11** renderer is used.

### 6.14.1 Loading of data

There are several ways how the data can be imported to the **VolumeModel**:

- Data can be supplied to the **Data property** as a collection of images which represent slices of the dataset
- Data can be supplied directly to the constructor of the **VolumeModel** in various ways
- Data can be supplied to the **VolumeModel** via one of the load functions

Load functions and constructors allow to supply data as a collection of slices (same as **Data** property do) or as a string with a path to the folder with the slices (as .Net supported image extension). The data can also be provided as a texture map created by our tool. The texture map consists of slices, and its supplement also needs an additional information about the number of slices on the picture. This is required for an efficient usage of GPU input buffers. The texture map can be created by **ChartTools.CreateMap** function. Direct input of texture map is used to speed up the start of an application for a very big dataset.

### 6.14.2 Properties

**VolumeModel** contains typical properties of a 3D object in **LightningChart**, like **Visible, Rotation, Size, Position, MouseInteraction, and MouseHighLight**. In addition, the object also has very specific properties, which define how Volume Rendering engine handles it.

Brightness	
B	<b>10</b>
G	<b>10</b>
R	<b>10</b>
Darkness	
B	<b>0.7</b>
G	<b>0.7</b>
R	<b>0.7</b>
EmptySpaceSkipping	128
MouseHighlight	Simple
MouseInteraction	True
Opacity	<b>0.15000000596046448</b>
Position	
X	<b>55</b>
Y	<b>50</b>
Z	<b>50</b>
RayFunction	<b>Accumulation</b>
Rotation	
X	<b>270</b>
Y	0
Z	<b>180</b>
SamplingRateOptions	
Enabled	<b>False</b>
Inerthness	<b>2</b>
ManualSamplingRate	512
> SamplingRateRange	
TargetFPS	<b>15</b>
Size	
Depth	<b>100</b>
Height	<b>75</b>
Width	<b>100</b>
SliceRange	
▼ Max	
X	<b>0.7</b>
Y	<b>0.8</b>
Z	<b>1</b>
▼ Min	
X	<b>0.3</b>
Y	<b>0.2</b>
Z	0
Smoothness	<b>2</b>
Threshold	
▼ Max	
B	<b>1</b>
G	<b>1</b>
R	<b>1</b>
▼ Min	
B	<b>0.5</b>
G	<b>0.5</b>
R	<b>0.5</b>
Visible	True
XAxisBinding	Primary
YAxisBinding	Primary
ZAxisBinding	Primary

Figure 6-74. Property tree of VolumeModels

### 6.14.3 Ray Function

It allows choosing one of the three ways of voxel sampling and composition available in **LightningChart Volume Rendering Engine**:

- **Accumulation Ray Function** collects and combines as much data as possible. The visualization which is produced by this technique looks like a semi-transparent gel. The figure below shows an example of **Accumulation Ray Function** application, visualizing a medical dataset.



Figure 6-75. Example of a medical application for the Accumulation Ray Function

- **Maximum Intensity Ray Function** takes into account only the brightest value sampled by the ray. Visually it provides a very similar result to X-ray images. It allows to get an additional information about the internal structure of the object. **Maximum Intensity Ray Function** applications for ultrasound wave's interference simulation and skeleton visualization are shown below.

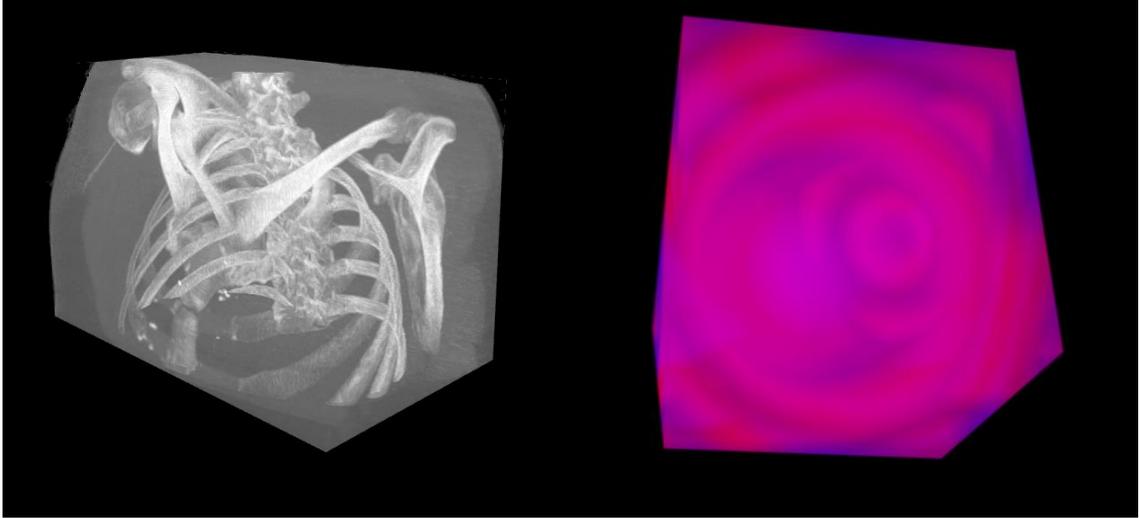


Figure 6-76. Examples of a Maximum Intensity Ray Function application

- **Isosurface Ray Function** draws the model surface in a way that it looks like a polygonal model rendering. The result is very similar to those produced by **Indirect Volume Rendering**. Figures show examples of **Isosurface Ray Function** applications for the visualization of human skull CT and simulation of water flow.

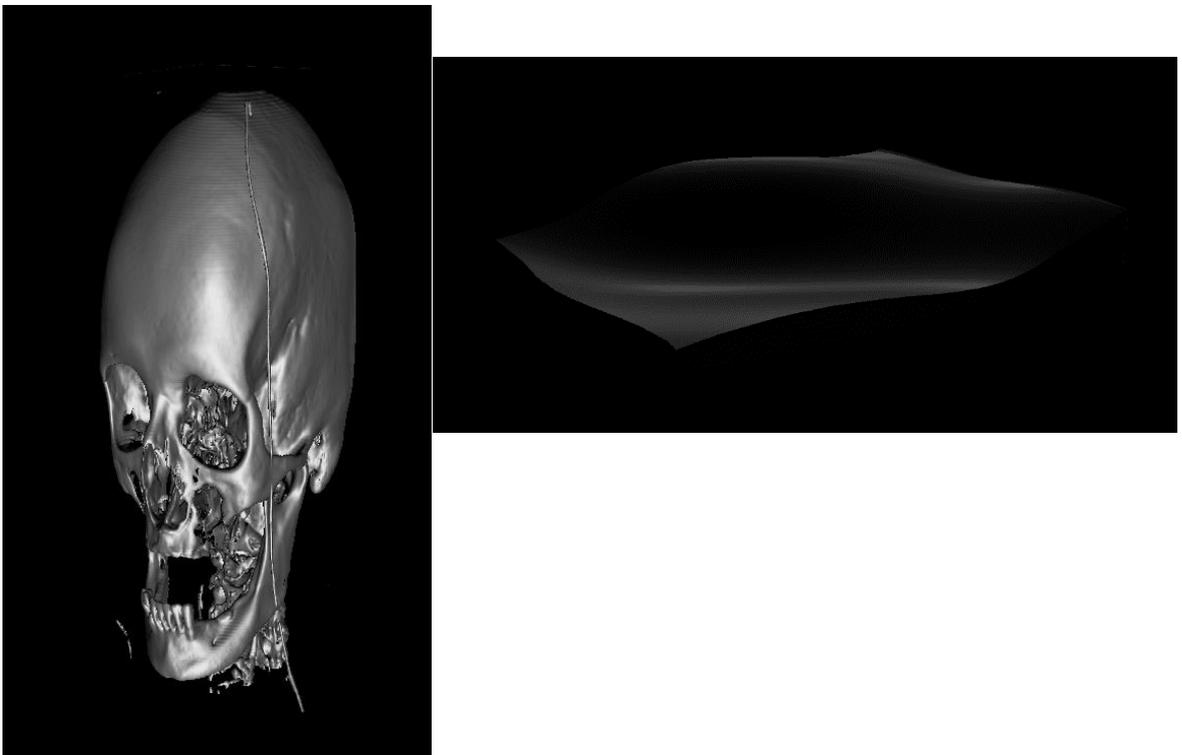


Figure 6-77. Examples of an Isosurface Ray Function application

#### 6.14.4 Threshold

The **Volume Rendering Engine** is able to apply a threshold range by the property to the **VolumeModel**. There is a separate boundary for every colour channel. The voxel is visualized only if it is lower than the high boundary, and higher than the low at all the channels. Acceptable areas are invisible. This property is not taken into consideration by the mouse hit test.

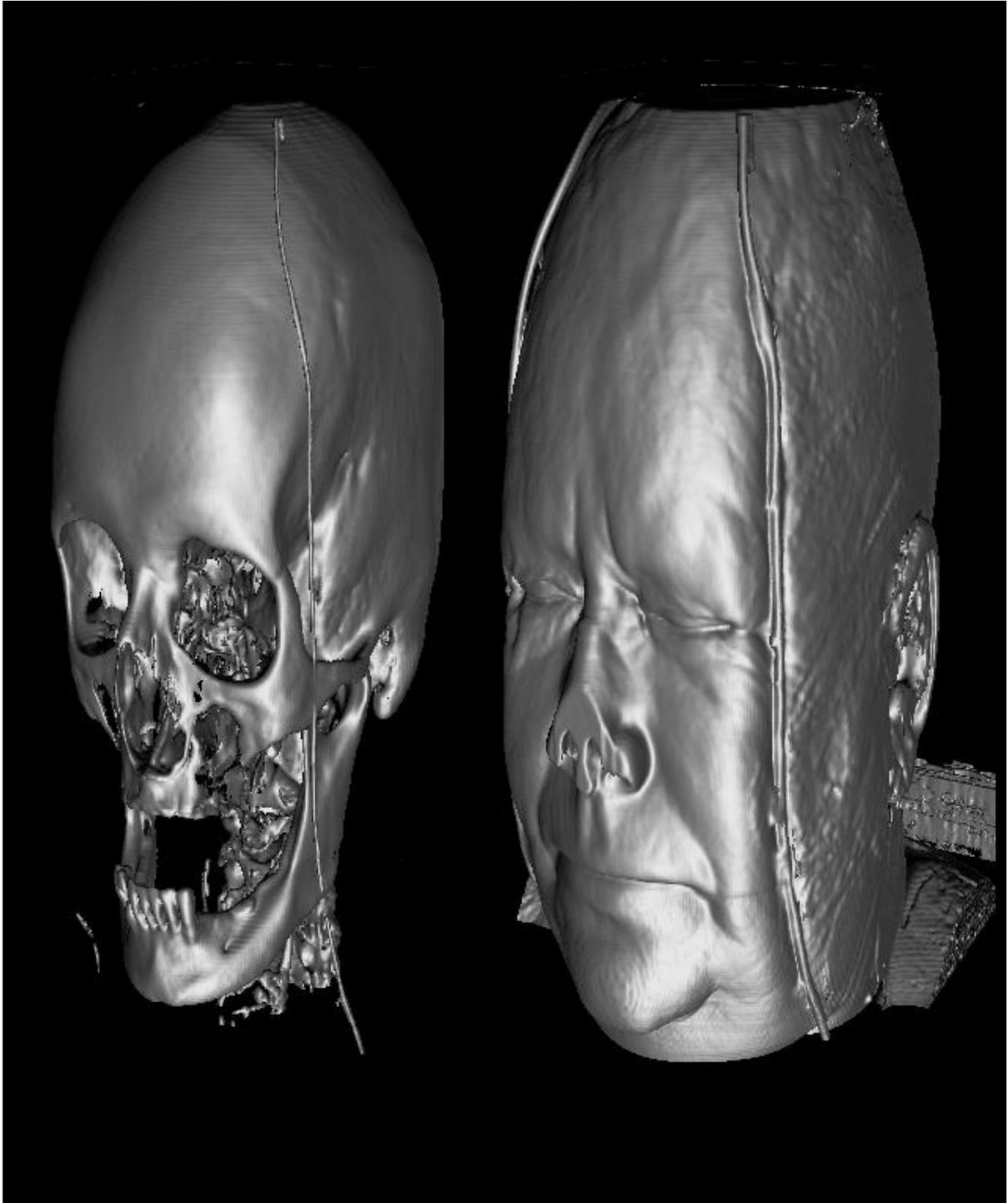


Figure 6-78. Example of two different threshold settings

### 6.14.5 Slice Range

The property allows cutting away a part of the **VolumeModel**. It is a very useful tool for the exploration of the object's internal structure. Slice Range contains two boundaries. Each of them is represented by three pointing float values.

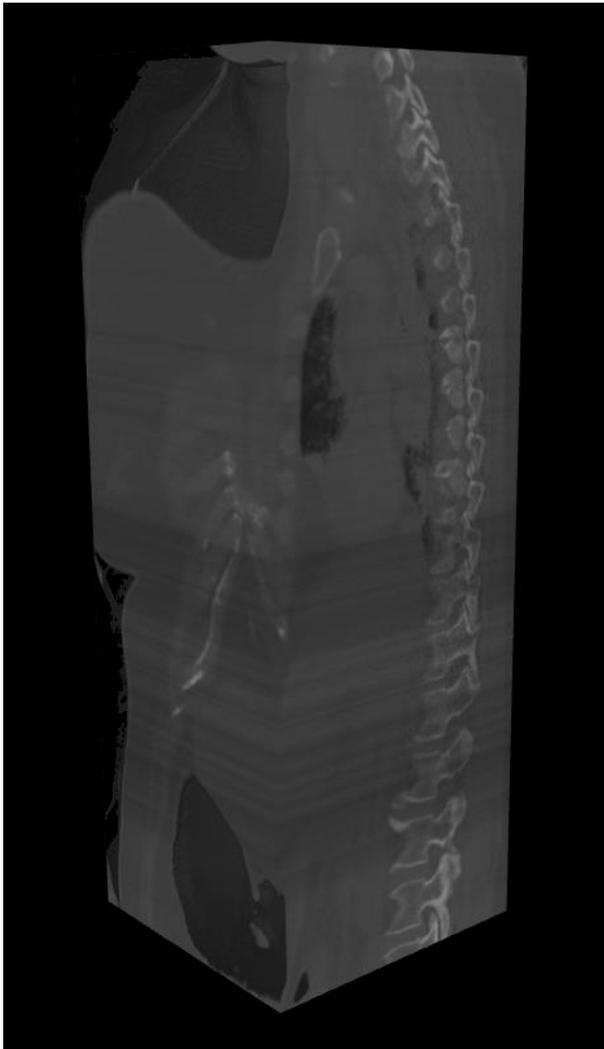


Figure 6-79. Example of Accumulation Ray Function and SliceRange modification

### 6.14.6 Sampling Rate Options

**Sampling Rate** is very important to the final image quality. It defines how often the volume dataset is sampled along the ray's track. Higher sampling rate produces better quality but requires more powerful hardware. **Sampling Rate** significantly influences the **Accumulation Ray Function**. Artefacts produced by the low sampling rate are less noticeable when using **Maximal Intensity function**. **Isosurface Ray Function** can be too sharp at a very high sampling rate. Usually, the sweet spot equals the number of voxels on the side which is placed along the ray tracks.

**SamplingRateOptions** contains several options for **SamplingRateManager**. **SamplingRateManager** is needed to reach the optimal balance between quality and frame rate for a particular hardware. The usage of sampling rate manager has to be turned on by the property **Enabled**, otherwise, the **ManualSamplingRate** value would be used. **SamplingRateRange** defines the boundaries for **SamplingRateManager**. **Inertness** specifies how rapid are the reaction of sampling rate in case of performance changes. **TargetFPS** is an aim, which should be achieved by sampling rate manager if it is possible.



Figure 6-80. Example of low sampling rate: 32(left), 64(right)

### 6.14.7 Smoothness

It prevents too high detalization of the surface. The factor makes the surface smoother, and reduces some noise and other artefacts.

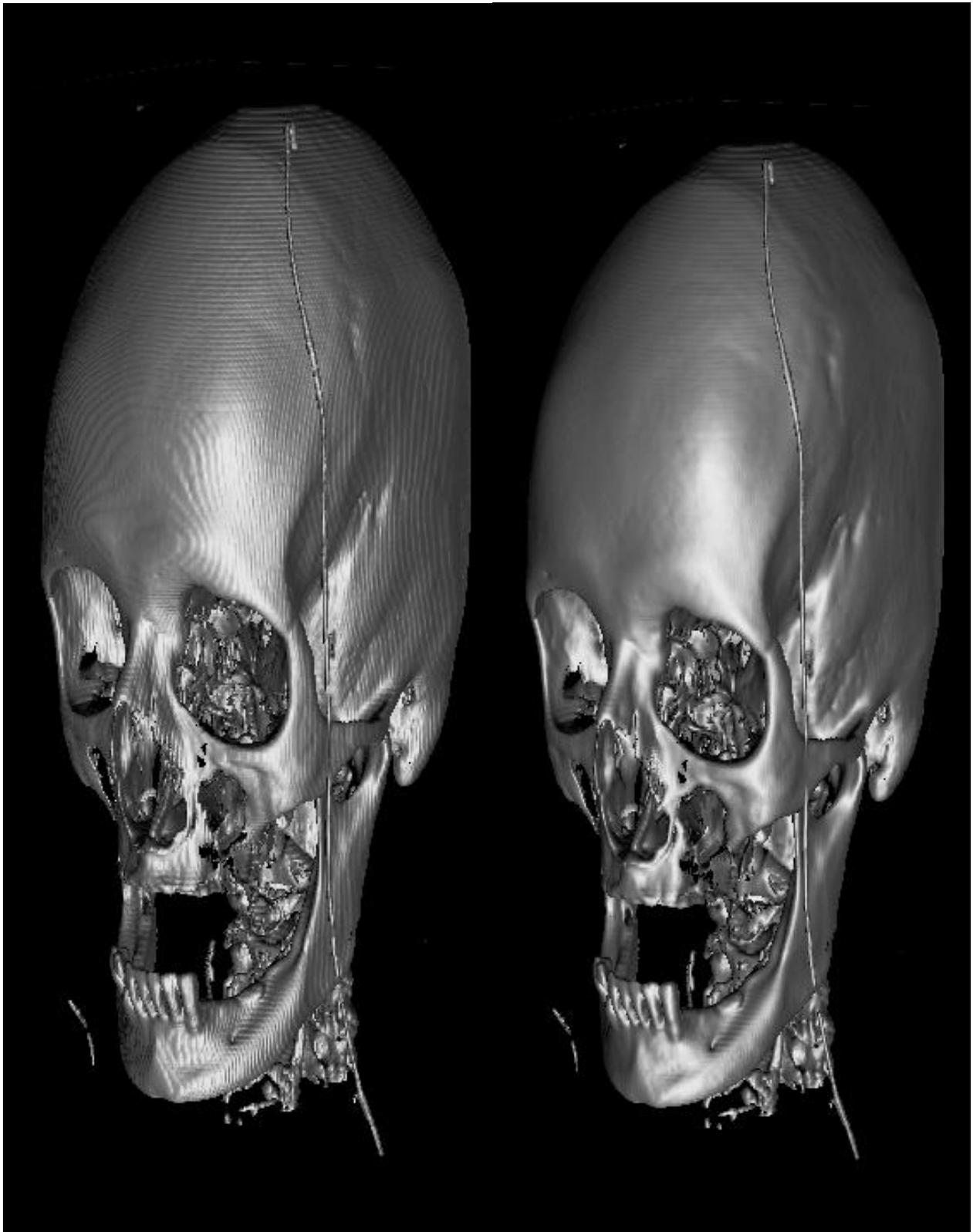


Figure 6-81. Example of too high sampling rate, fixed by smoothness property

### 6.14.8 EmptySpaceSkipping

Defines a resolution of empty space, skipping sampling. A low value (16-32) of **EmptySpaceSkipping** improves the performance but can cause artefacts in the model edges.

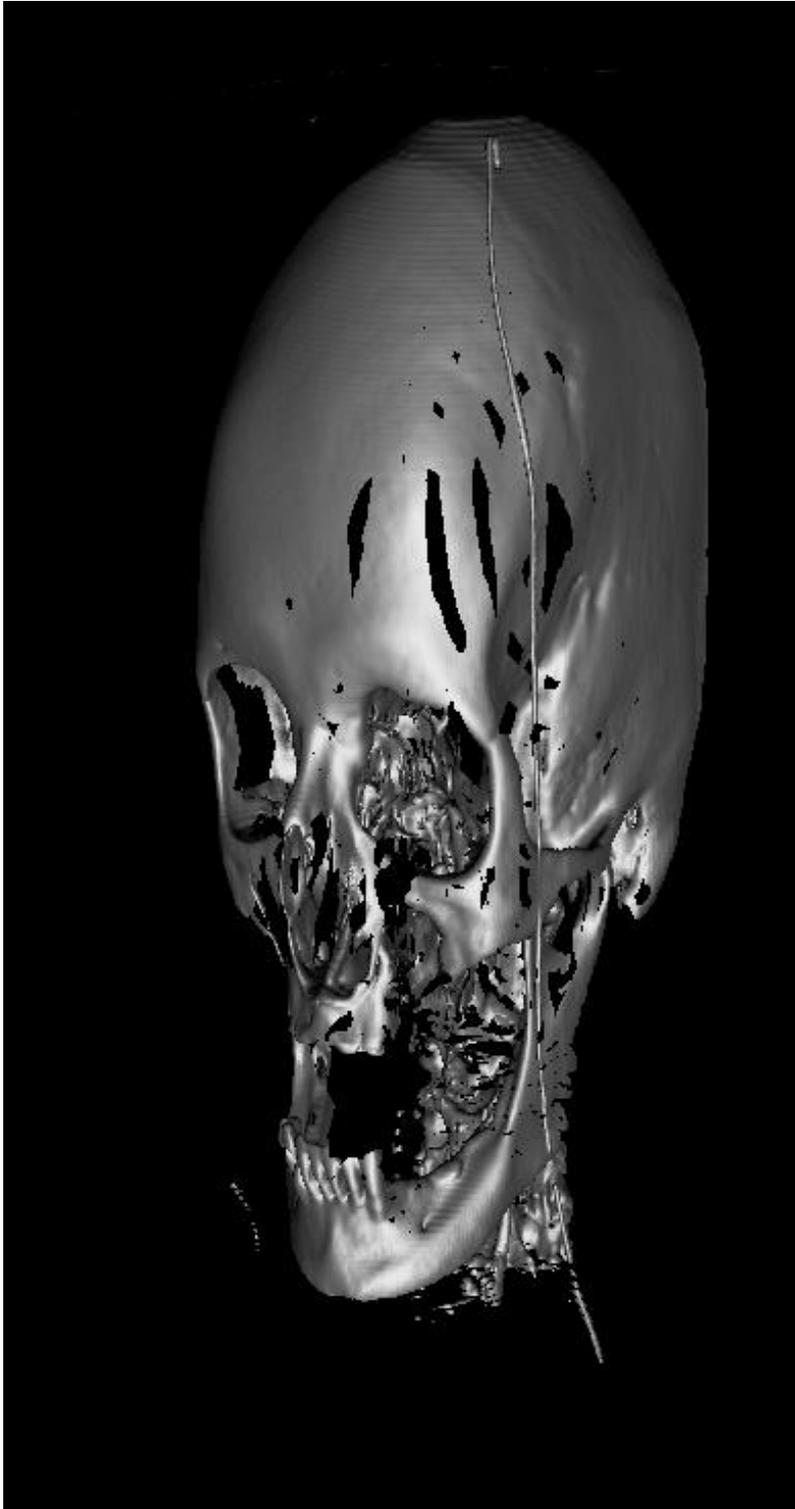


Figure 6-82. Example of too low EmptySpaceSkipping property value

### 6.14.9 Opacity

Specifies an **Accumulation Ray Function** behaviour. Low opacity makes an object more transparent.



Figure 6-83. Example of Accumulation Ray Function Opacity modification: 15% (left), 45(right)

## 6.14.10 Brightness and Darkness

The properties define the image's transfer function. Every change has its own transfer function. It is represented by the linear function:  $output = Brightness * input - Darkness$

## 6.15 Rectangle3D objects

**Rectangle3D** objects can be added in **View3D.Rectangles** list. They allow presenting a rectangle, turned to any angle, at any size, at any location. Rectangles can act also as planes by defining their size identical to **View3D.Dimensions** or more.

Set **Size** in 3D world dimensions (not X, Y or Z axis values) as Width and Height. Set the center point in **Center** property, it's defined in X, Y and Z axis values. Set rotation in degrees in **Rotation** property.

Fill settings can be modified in **Fill** property. Solid color and bitmap fills are available. To use bitmap fill, set the bitmap in **Image**, and enable **UseImage**. When setting **Fill.Layout = Stretch**, the bitmap stretches to fill the rectangle. By setting **Fill.Layout = Tile**, the same bitmap is tiled to fill the rectangle. The tile count can be set in **Fill.TileCountWidth** and **Fill.TileCountHeight** properties.

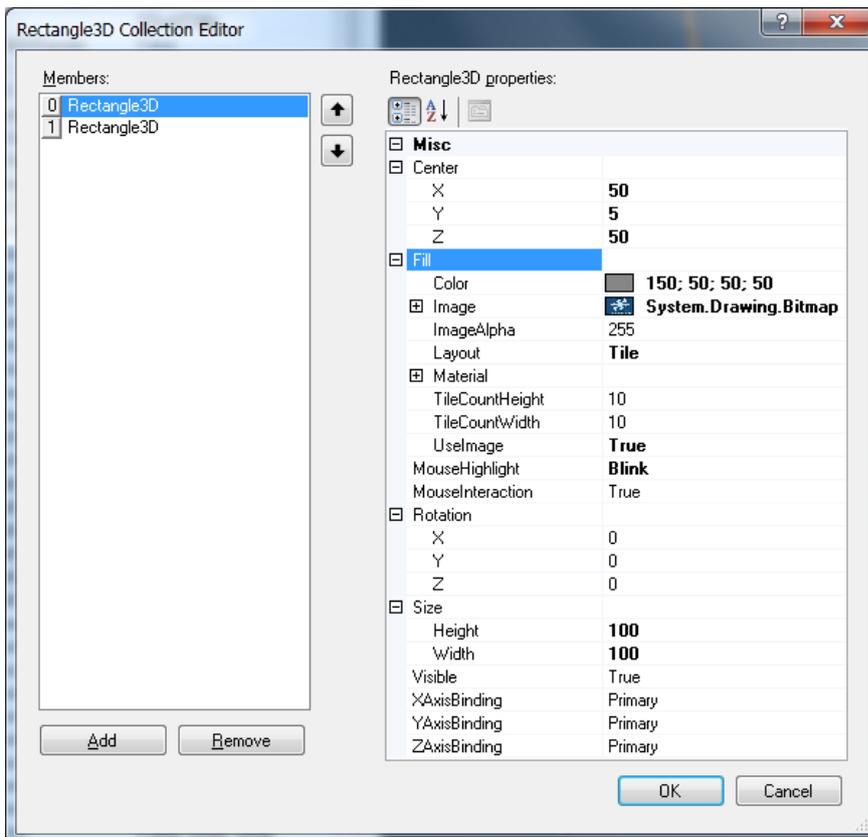


Figure 6-84. Properties of Rectangle3D objects.

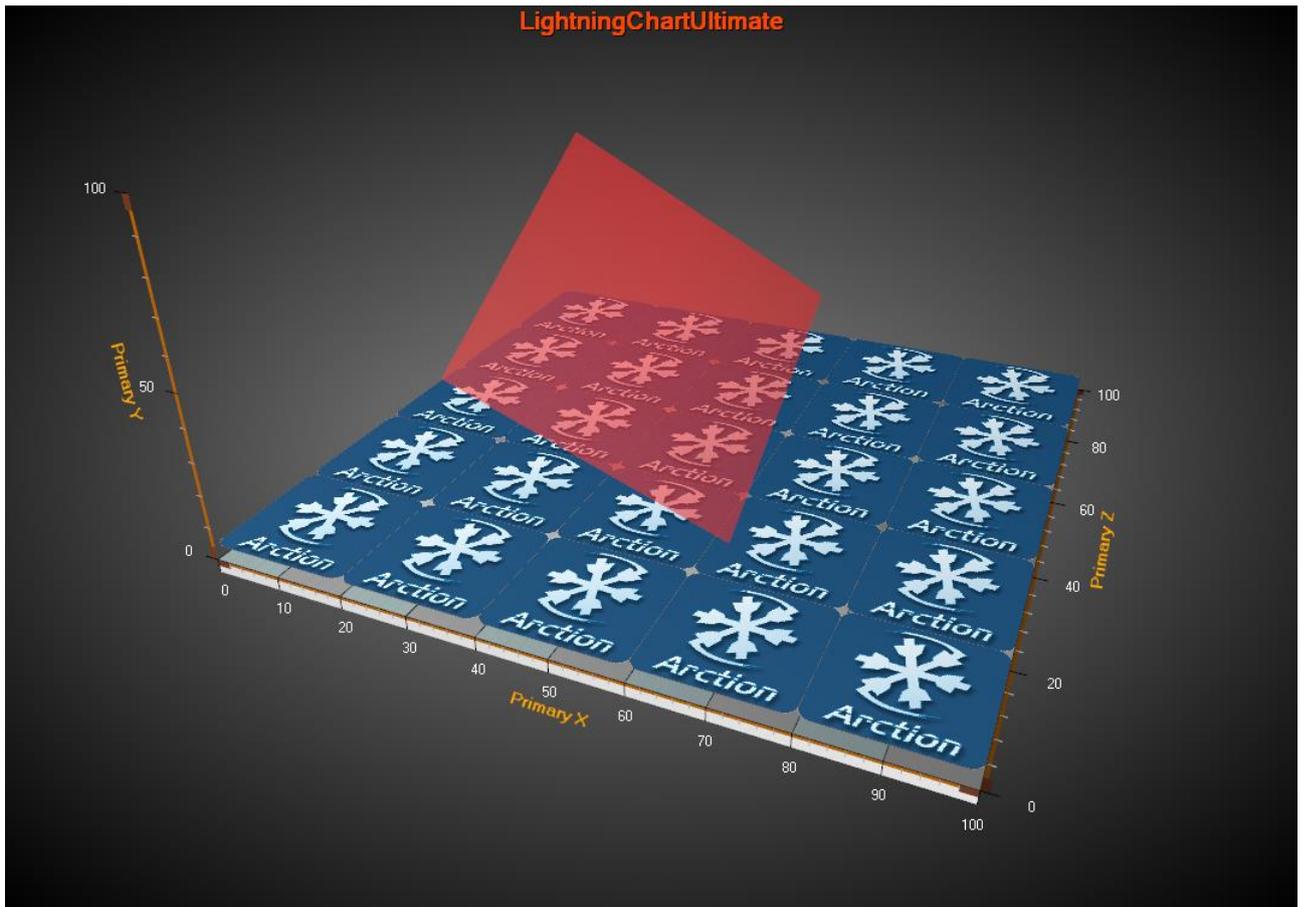


Figure 6-85. Two Rectangle3D objects in View3D. The blue one in the bottom shows a bitmap fill, with Layout = Tile. Red rotated rectangle on the top is configured with translucent color.

## 6.16 Polygon3D objects

**Polygon3D** objects can be added in **View3D.Polygons** list. They allow presenting a 2D polygon, stretched to given Y range.

Define the polygon path in X and Z axis values. Store the path in **Points** array. Set the Y range with **YMin** and **YMax** values.

Set the main color in **Material.Diffuse**. Rotate the polygon to another angle with **Rotation.X**, **Rotation.Y** and **Rotation.Z** in degrees.

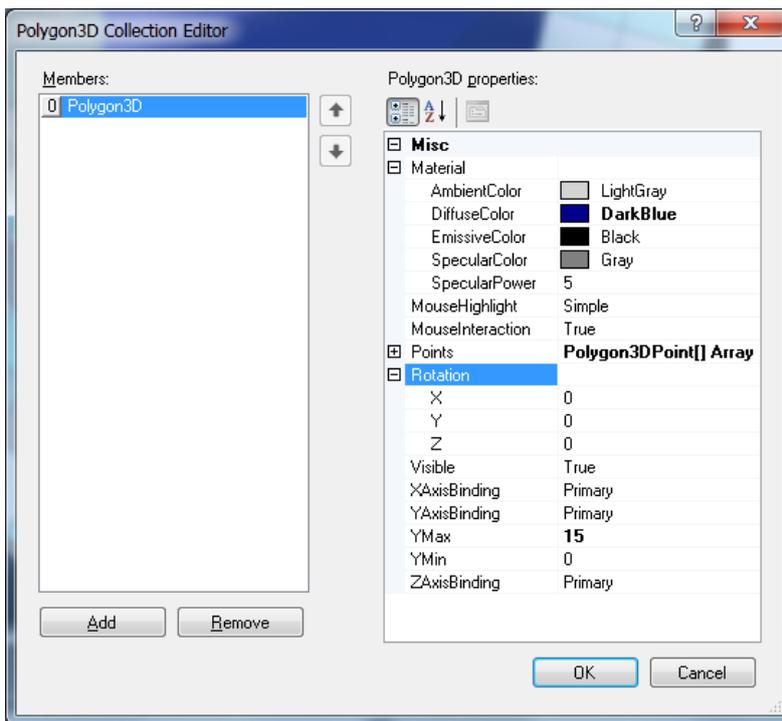


Figure 6-86. Properties of Polygon3D objects.

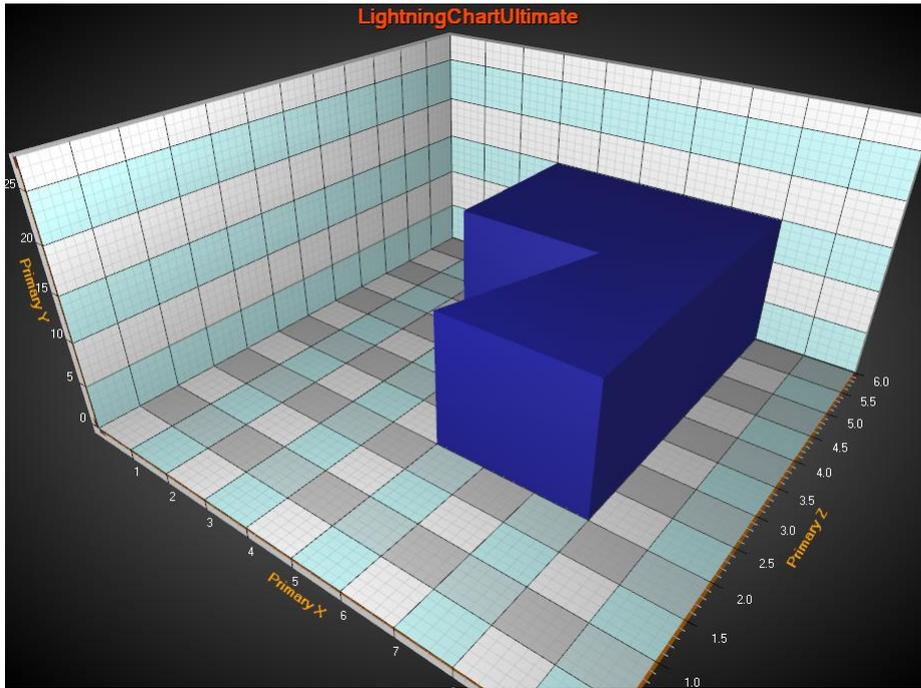


Figure 6-87. A 6-point polygon ranging from YMin = 0, YMax = 15.

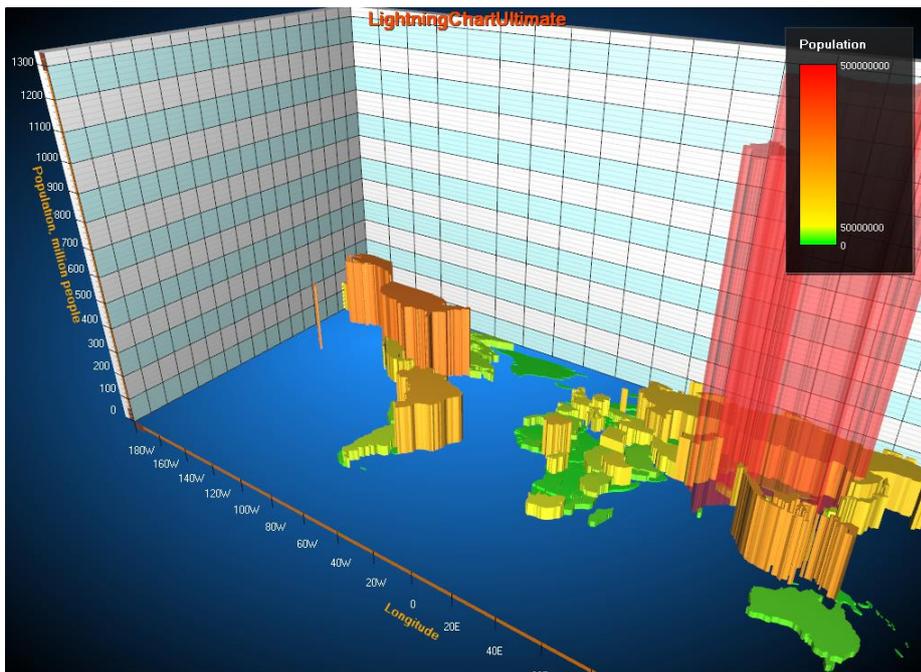


Figure 6-88. World population shown with Polygon3D objects. A Polygon3D object for each region of map data. Population value of a country is used to color the polygon and to set the YMax of it. China and India are shown with translucent colors because of high population of theirs.

## 6.17 Zooming, panning and rotating

Use **ZoomPanOptions** properties to control the zooming, panning and rotation settings.

ZoomPanOptions	
AutoFit	False
AxisMouseWheelAction	Pan
BoxZoomingOutCrossVisible	True
BoxZoomOutFactor	2
LeftMouseButtonAction	Rotate
LimitBoxZoomInsideGraph	True
MiddleMouseButtonAction	Pan
MouseWheelZoomEnabled	True
MouseWheelZoomFactor	1.1
MultiTouchPanEnabled	True
MultiTouchZoomEnabled	True
PanSensitivity	1
RightMouseButtonAction	Pan
RightToLeftZoomAction	ZoomOut
RotationSensitivity	1
WheelAreaThickness	2
ZoomBoxColor	<input type="color" value="#302550"/> 30, 255, 165, 0
▸ ZoomInBoxLineStyle	
▸ ZoomOutBoxLineStyle	

None
Pan
Rotate
PanPrimaryXZ
PanPrimaryXY
PanPrimaryYZ
ZoomXY
ZoomXZ
ZoomYZ
ZoomX
ZoomY
ZoomZ

Figure 6-89. ZoomPanOptions properties and sub-properties, with LeftMouseButton / MiddleMouseButtonAction / RightMouseButtonAction options on the right.

Zooming can be performed with mouse wheel or by touch screen pinching/spreading, or by painting a box on selected 3D plane. Panning, box zooming and rotating can be performed by left, middle or right mouse button, they are configurable. Panning can be made for the whole 3D chart, or so that primary axes are adjusted but 3D scene location stays same.

### 6.17.1 Mouse wheel zooming

To enable mouse wheel zooming, set **MouseWheelZoomEnabled** to **True**. To disable zooming, set it to **False**. Scroll mouse wheel up to zoom in, and down to zoom out. Use **MouseWheelZoomFactor** to adjust the amount of applied zoom with every mouse wheel event.

### 6.17.2 Box zooming

To enable box zooming assign the box zooming to a mouse button action property. E.g. **LeftMouseButtonAction = ZoomXZ** and then the box zoom applies to XZ plane. Y dimension is not affected. Set **ZoomXZ** or **ZoomYZ** respectively if you want to zoom other planes.

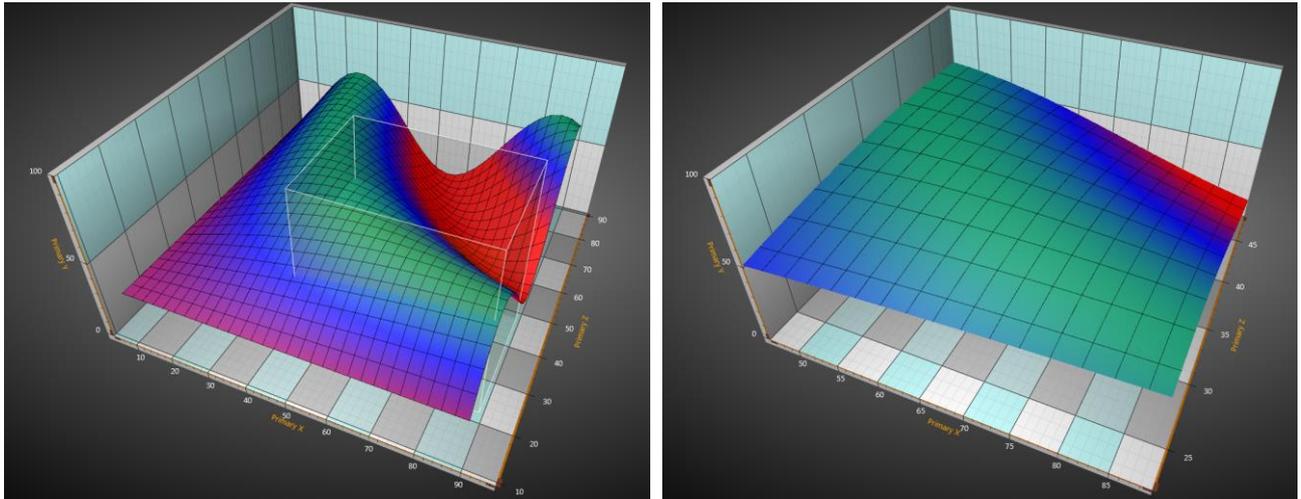


Figure 6-90. XZ-plane box zooming in progress and after mouse button released. X and Z axis ranges are modified, Y axis range not.

Zoom in by dragging box from left to right. The zoomed ranges are applied to axes related to the selected plane.

To zoom only specific dimension, X, Y or Z, select **ZoomX**, **ZoomY** or **ZoomZ**.

Zoom out by dragging box from the right to left. Zooming out is applied by factor set in **BoxZoomOutFactor**. Zooming out shows a cross in the front of the box, to disable that, set **BoxZoomingOutCrossVisible = False**.

### 6.17.3 Rotating and panning

Camera can be rotated around the 3D model by pressing the assigned mouse button down and by dragging horizontally or vertically. **RotationX**, **RotationY** and **RotationZ** properties are updated.

When a mouse button action is set to **Pan**, panning updates **Target** property of **Camera**. When mouse button action is set to **PanPrimaryXZ**, **PanPrimaryXY** or **PanPrimaryYZ**, the primary X, Y and Z axes ranges are adjusted. For example, **PanPrimaryXZ** adjusts X and Z axes when dragging with mouse. Secondary X, Y and Z axes are not altered.

Set **LeftMouseButtonAction** / **MiddleMouseButtonAction** / **RightMouseButtonAction** to **Pan/PanPrimaryXZ/PanPrimaryXY/PanPrimaryZ** to enable panning. Set it to **Rotate** to enable rotating. To disable panning and rotating from left mouse button, set it to **None**.

Use **PanSensitivity** to control the amount of applied panning. Respectively, use **RotationSensitivity** to control the amount of applied rotation.

#### 6.17.4 Zooming with touch screen

Set two fingers on the chart, and pinch your fingers closer to zoom out, or away to zoom in. To disable zooming with touch screen, set **MultiTouchZoomEnabled** to **False**.

#### 6.17.5 Panning with touch screen

Set two fingers on the chart, and move your fingers to same direction to apply panning. To disable panning with touch screen, set **MultiTouchPanEnabled** to **False**.

#### 6.17.6 Using mouse wheel over an axis

When mouse wheel is scrolled over an axis, the chart makes axis-specific zooming or panning. **WheelAreaThickness** to adjust how wide the mouse wheel sensitive area is, near axis. **AxisMouseWheelAction** can be used to select between zooming and panning.

#### 6.17.7 Zooming, rotating and panning by code

3D view is rotated by moving the **View3D.Camera** with **RotationX**, **RotationY** and **RotationZ** properties. When orthographic camera is not used, zoom can be done by setting **ViewDistance**. With orthographic camera, the **Dimensions** must be changed to achieve zoom. Panning is done by setting camera **Target**, as 3D model coordinates.

### 6.18 Clipping objects within axis ranges

By setting **ClipContents** property to **True**, series, rectangles and mesh models are clipped inside axis value ranges. The axes are always stretched for a dimension, so when clipping is enabled, it prevents rendering outside the walls.

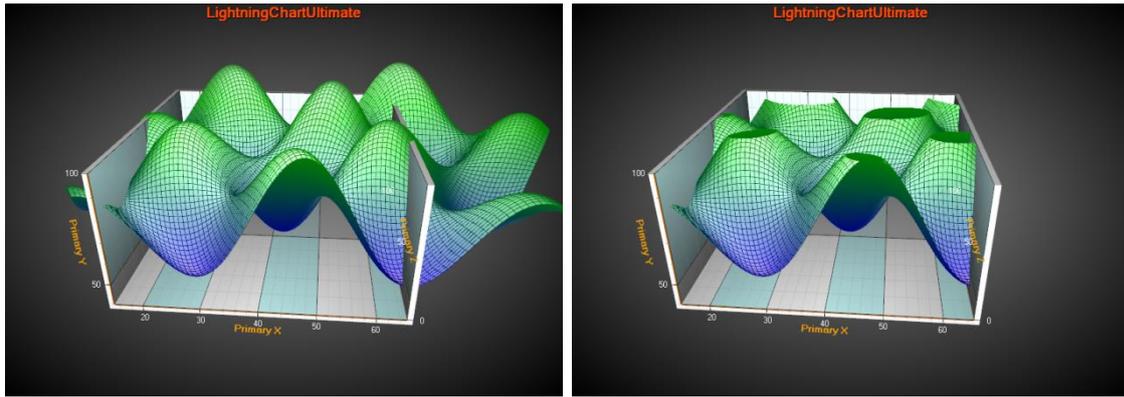


Figure 6-91. On the left, `ClipContents` is not used. Series render outside axis ranges. On the right, `ClipContents` is enabled,

Note that clipping does not modify the series data set itself. Clipping occurs only in rendering stage. Also mouse hit test will take effect also outside the walls for invisible, clipped objects.

When clipping is enabled, all lines in the chart are automatically set to line width of 1.

## 6.19 Annotations

Annotations collection allows adding annotations into the 3D scene. In general, they are very similar to `ViewXY's Annotations` (see section 5.19), but **Target** and **Location** control support 3D.

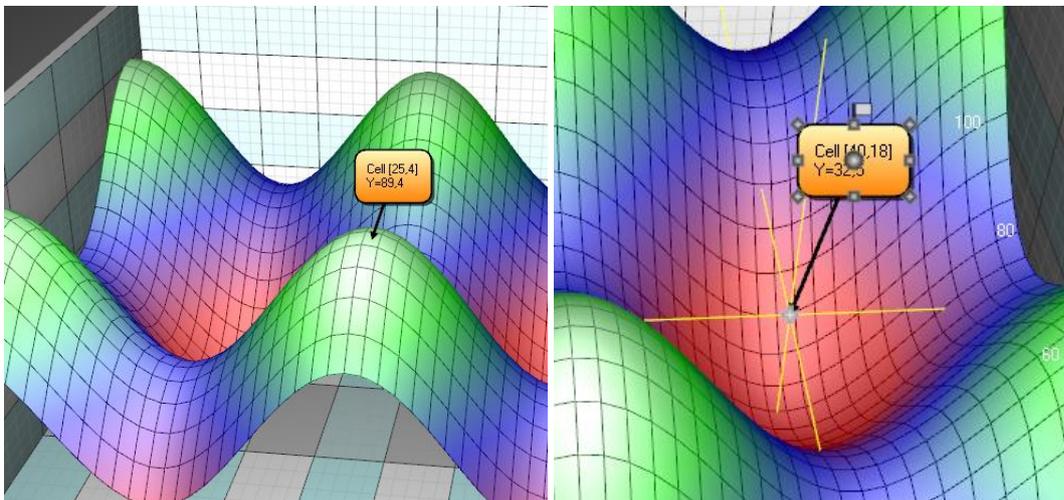


Figure 6-92. `Annotation3D` object displaying value of 3D series. Crosshair cursor can be used to aid the target movement.

**Target** can be moved by mouse in 3D. For aiding the movement, annotation shows cross-hair lines when mouse is over the **Target** node. Set `ShowTargetCrosshair` property to **Auto/On/Off**, and adjust the line style in `TargetCrosshairLineStyle`.

## 7. ViewPie3D

ViewPie3D presents data as pie and donut charts, in 3D.

ViewPie3D	3D pie/donut view (Collection)
Annotations	
▶ Camera	
DonutInnerPercents	50
ExplodePercents	10
▶ LegendBox3DPie	<b>LegendBoxPie3D</b>
LightingScheme	DirectionalFromCamera
Lights	<b>(Collection)</b>
▶ Material	
Rounding	<b>40</b>
StartAngle	0
Style	Pie
Thickness	25
TitlesNumberFormat	<b>0 USD</b>
TitlesStyle	<b>Values</b>
Values	<b>(Collection)</b>
▶ ZoomPanOptions	

Figure 7-1. ViewPie3D object tree.

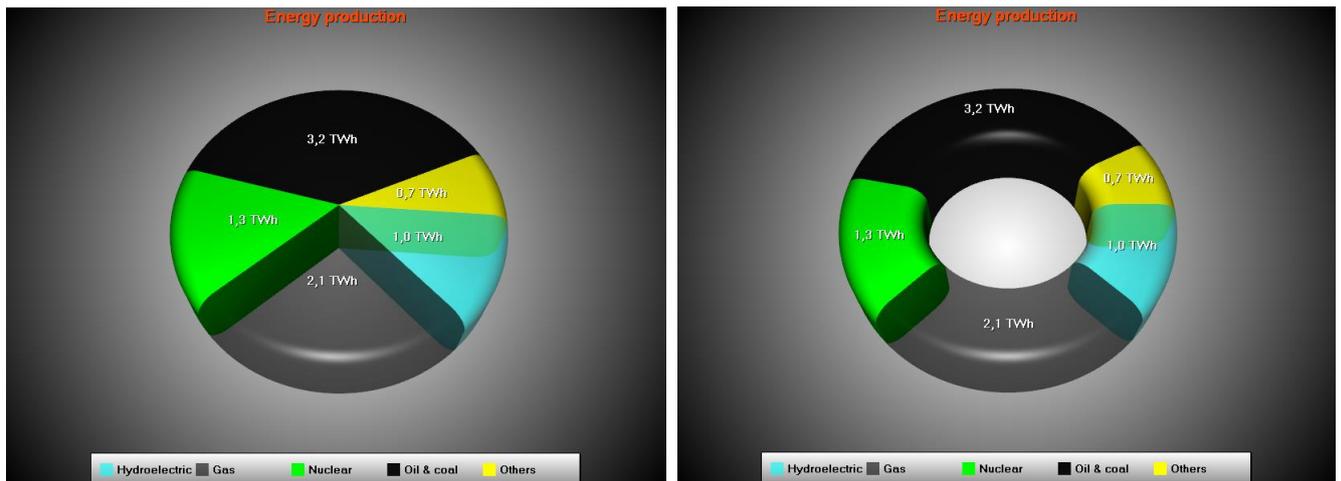


Figure 7-2. Example of Pie and Donut charts.

## 7.1 Properties

Select the chart type by using **Style** property, **Pie** or **Donut**. Control the zooming, panning and rotation with **ZoomPanOptions** property tree, much similar to View3D (see chapter 6).

**Camera** property controls the viewpoint, see section 6.4. Predefined lighting setup can be selected with **LightingScheme** property. Use Material property and its sub-properties to adjust general 3D surface appearance and shininess.

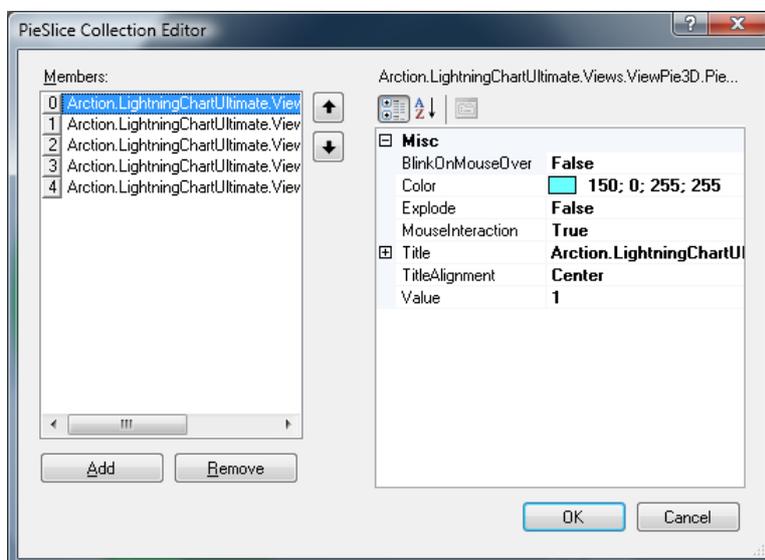
Use **DonutInnerPercents** to set the donut inner radius, **Rounding** to adjust edge rounding radius, **StartAngle** to rotate the pie, and **Thickness** to adjust pie thickness. **ExplodePercents** adjusts how far away the exploded pie slice is, when slice's **Explode** is true.

**TitlesStyle** sets the pie slice text of one of the following: **Titles**, **Values** or **Percents**. Edit **TitlesNumberFormat** for example to "0.0 TWh" to include units in the end.

**Annotations** can be used in same way than in View3D, but without axis value binding properties. See section 6.19.

## 7.2 Pie slices

The **Values** list editor looks like this:



Each item in the list is of type **PieSlice**. Edit the data value in **Value** property. Set title string into **Title.Text** property. By defining **TitleAlignment** = **Outside**, the title is drawn outside the pie.

## 7.3 Setting data by code

Data is stored in the **Values** list. Each item in the list is of type **PieSlice**.

```
//Add pie slice data
//By using last true as last parameter, the slice is automatically added to
chart.ViewPie3D.Values collection
PieSlice slice1 = new PieSlice("Hydroelectric",
Color.FromArgb(150, Color.Aqua), 1.0, chart.ViewPie3D, true);

PieSlice slice2 = new PieSlice("Gas",
Color.FromArgb(150, 0, 0, 0), 2.1, chart.ViewPie3D, true);

PieSlice slice3 = new PieSlice("Nuclear", Color.Lime, 1.3, chart.ViewPie3D,
true);

PieSlice slice4 = new PieSlice("Oil & coal", Color.FromArgb(240,0,0,0), 3.2,
chart.ViewPie3D, true);

PieSlice slice5 = new PieSlice("Others", Color.Yellow, 0.66,
chart.ViewPie3D, true);

slice3.Explode = true;
```

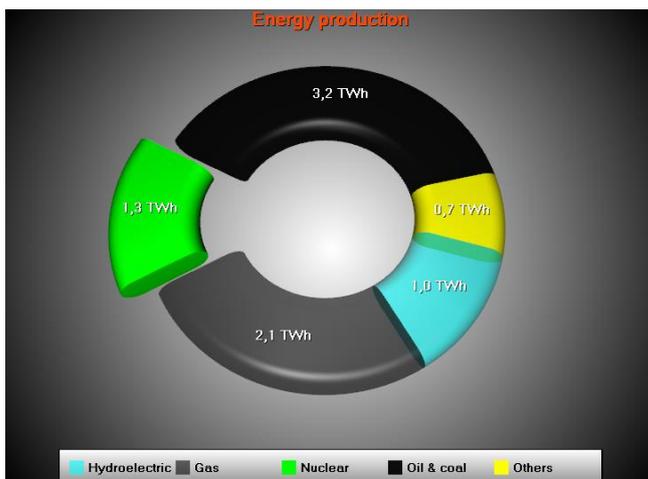


Figure 7-3. Data set into chart. Third slice is separated by using `slice3.Explode = true`.

## 7.4 Viewing pie chart in 2D

Set the camera as predefined camera from top.

```
chart.ViewPie3D.Camera.SetPredefinedCamera(PredefinedCamera.PieTop);
```

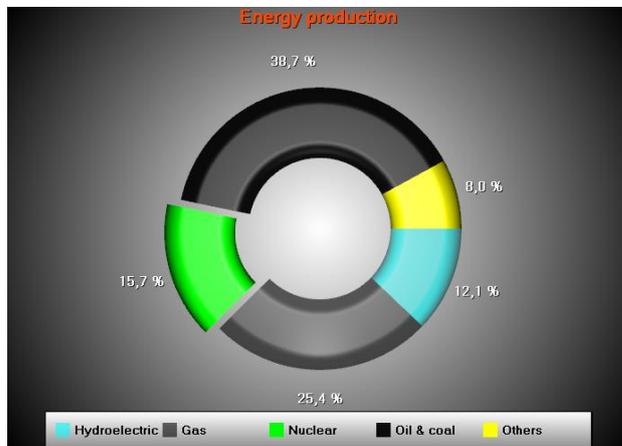


Figure 7-4. Pie chart shown as 2D, with a predefined camera from top.

## 8. ViewPolar

**ViewPolar** allows data visualization in a polar format. The data point position is determined by angular value and amplitude (compare angle as X and amplitude Y in **ViewXY**). Polar view has zooming and panning features.

<b>ViewPolar</b>	<b>Polar chart view</b>
Annotations	<b>(Collection)</b>
AreaSeries	<b>(Collection)</b>
Axes	<b>(Collection)</b>
▶ GraphBackground	
▶ LegendBox	<b>LegendBoxPolar</b>
▶ Margins	<b>10, 30, 10, 10</b>
Markers	<b>(Collection)</b>
PointLineSeries	<b>(Collection)</b>
Sectors	<b>(Collection)</b>
▶ ZoomCenter	<b>0;0</b>
▶ ZoomPanOptions	
ZoomScale	<b>1</b>

Figure 8-1. ViewPolar object tree.

## 8.1 Axes

You can define polar axes into **Axes** list property. Several axes can be used in same chart. Series can be assigned with any of these axes by setting **AssignPolarAxisIndex** property of a series. An axis represents **both angular** scale and **amplitude** scale. Otherwise, the polar axes are very similar to ViewXY axes, see section 5.2.

AmplitudeAxisAngle	0
AmplitudeAxisAngle Type	Relative
AmplitudeAxisLine Visible	True
AmplitudeLabelsAngle	<b>0</b>
AmplitudeLabels Visible	True
AmplitudeReversed	False
AngleOrigin	0
AngularAxisAutoDiv Spacing	True
AngularAxisCircle Visible	True
AngularAxisMajorDivCount	8
AngularLabels Visible	True
AngularReversed	False
AngularTicks Visible	True
AngularUnit Display	Degrees
AntiAliasing	True
AutoFormatLabels	True
AxisColor	 <b>Sienna</b>
AxisThickness	4
> GridAngular	
GridVisibilityOrder	<b>BehindSeries</b>
InnerCircleRadiusPercentage	0
KeepDivCountOnRangeChange	True
> LabelsFont	<b>Segoe UI, 9pt</b>
LabelTicksGap	5
MajorDiv	<b>6</b>
MajorDivCount	5
> MajorDiv Tick Style	
> MajorGrid	
MarginInner	5
MarginOuter	5
MaxAmplitude	<b>30</b>
MinAmplitude	0
MinorDivCount	5
> MinorDiv Tick Style	
> MinorGrid	
MouseDragSnapToDiv	False
MouseHighlight	Simple
MouseInteraction	True
MouseScaling	True
MouseScrolling	True
> ScaleNibs	
TickMarkLocation	Outside
> Title	<b>RoundAxis Title</b>
> Units	<b>RoundAxis Title</b>
UsePreviousAxisDiameter	False
Visible	True

Figure 8-2. AxisPolar property tree

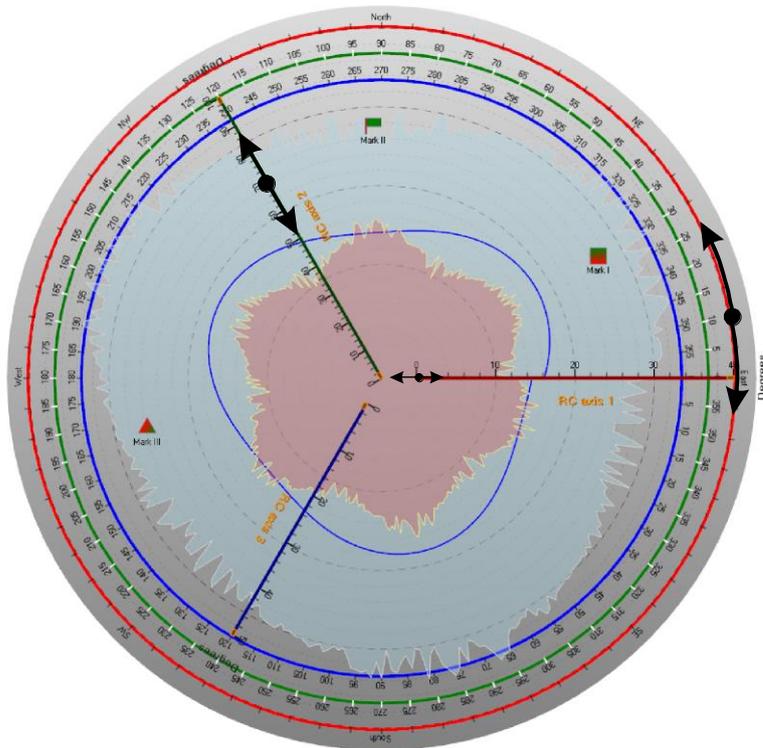


Figure 8-3. Three axes, first (red) in the outer circle, second (green) in the middle, and third (blue) closest to center. Axis AngleOrigin can be changed by dragging it over the axis circle. Amplitude range can be changed by dragging from the axis. Minimum or maximum of axis amplitude range can be changed by dragging from the small nib in the end of the amplitude scale.

### 8.1.1 Reversed axes

The axis can be reversed by amplitude, angle or both.

To reverse the angle scale, set **AngularReversed = True**. To reverse the amplitude scale, set **AmplitudeReversed = True**.

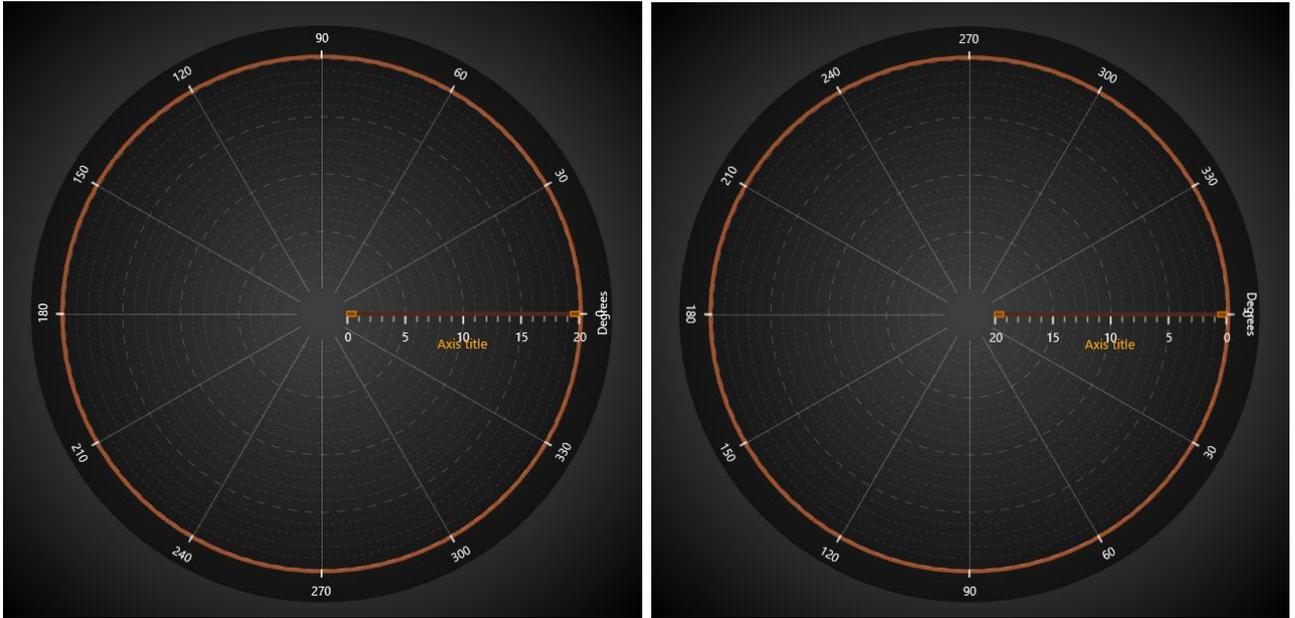


Figure 8-4. On the left, scales are not reversed. On the right, `AngularReversed = True` and `AmplitudeReversed = True`.

### 8.1.2 Setting rotation angles of the scales

Use `AngleOrigin` to set the rotation angle of angle scale.

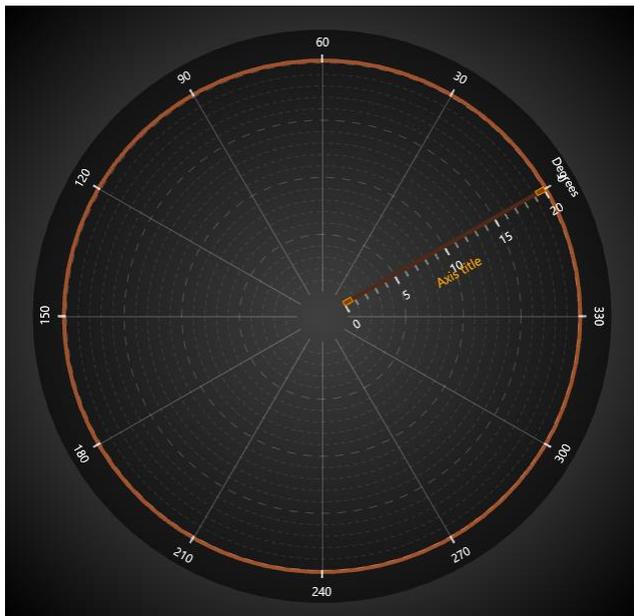
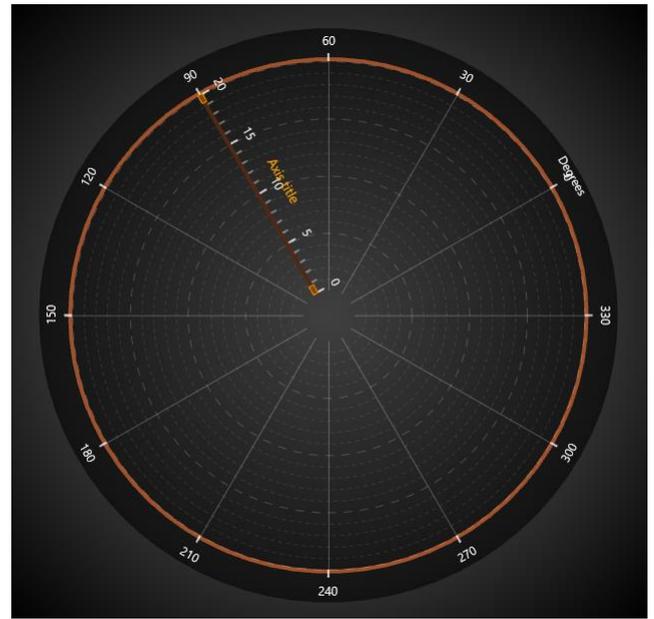
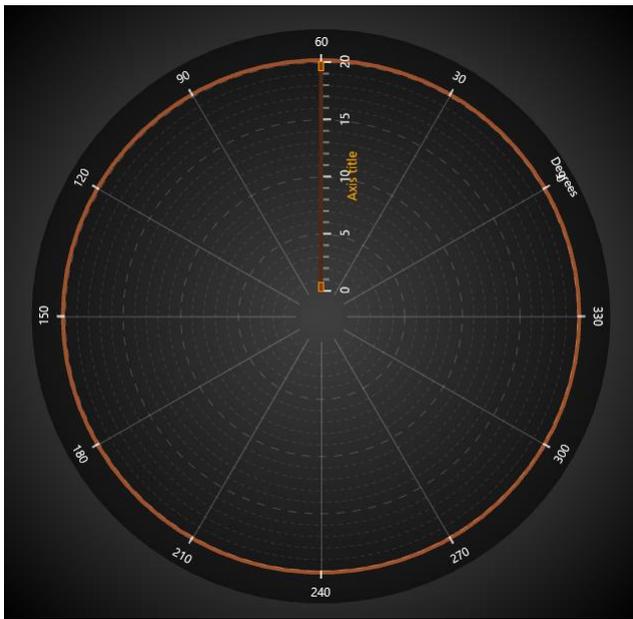


Figure 8-5. `AngleOrigin = 30`.

Use **AmplitudeAxisAngle** to rotate amplitude axis position. Amplitude scale angle can be set as absolute angle (**AmplitudeAxisAngleType = Absolute**), or relative to angle scale's angle.



(**AmplitudeAxisAngleType = Relative**).

Figure 8-6. **AngleOrigin = 30**. **AmplitudeAxisAngle = 90**. On the left, **AmplitudeAxisAngleType = Absolute**. On the right, **AmplitudeAxisAngleType = Relative**: Overall the amplitude scale rotates 120 degrees in this case.

### 8.1.3 Setting divisions

Set the amplitude division count with **MajorDivCount**, and division magnitude with **MajorDiv** property. The amplitude scale will adjust accordingly (updating **MaxAmplitude**). Set amplitude minor division count with **MinorDivCount**.

By default, the chart tries to include almost as many angular divisions as it can fit. To control the angular divisions, set **AngularAxisAutoDivSpacing** to False. Then the chart tries **AngularAxisMajorDivCount** count of divisions. If chart space is too small to render all the divisions and labels, it will use a lower division count that it can fit.

## 8.2 PointLineSeries

*PolarView's PointLineSeries* can be used to draw a line, group of points or point-line. Lots of line and point styles are available in *LineStyle* and *PointStyle* properties.

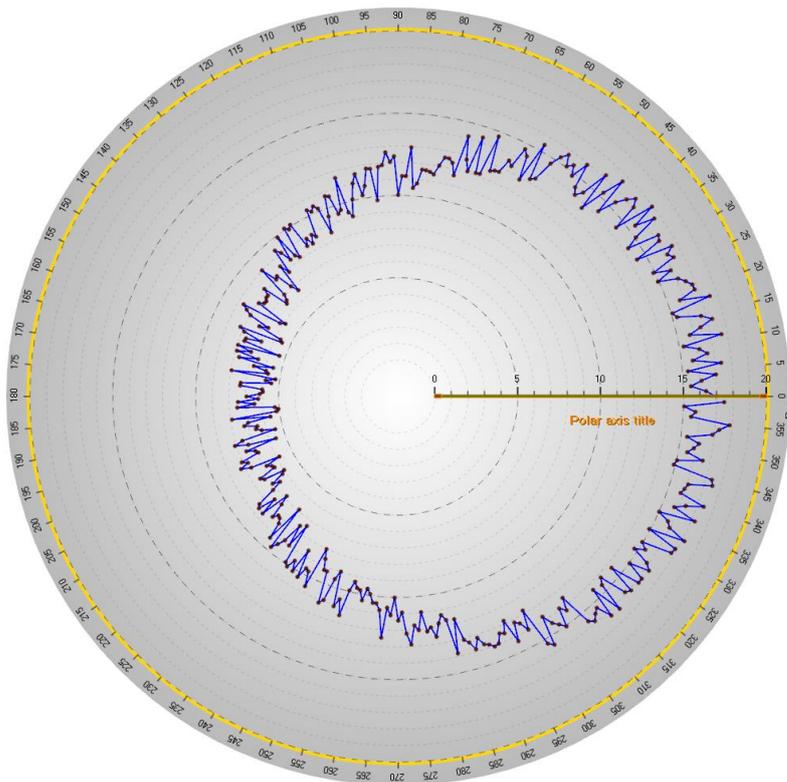


Figure 8-7. Some data presented with ViewPolar's PointLineSeries. Line and points are both visible.

### 8.2.1 Setting data

This code represents the data setting of previous figure.

```
int iCount = 360;
PolarSeriesPoint[] points = new PolarSeriesPoint[iCount];
Random rnd = new Random();

for (int i = 0; i < iCount; i++)
{
    points[i].Amplitude = 10.0 + 3.0 * rnd.NextDouble() + 5.0 *
        Math.Cos(AxisPolar.DegreesAsRadians((double)i * 1.0));
    points[i].Angle = (double)i;
}
chart.ViewPolar.PointLineSeries[0].Points = points;
```

## 8.2.2 Palette coloring

Line coloring supports palette.

Use **ColorStyle** property to select how the palette coloring is applied

- **LineStyle**: No palette fill. The color set in LineStyle.Color property applies
- **PalettedByAngle**: Data point **Angle** field determines the color
- **PalettedByAmplitude**: Data point **Amplitude** field determines the color
- **PalettedByValue**: Data point **Value** field determines the color

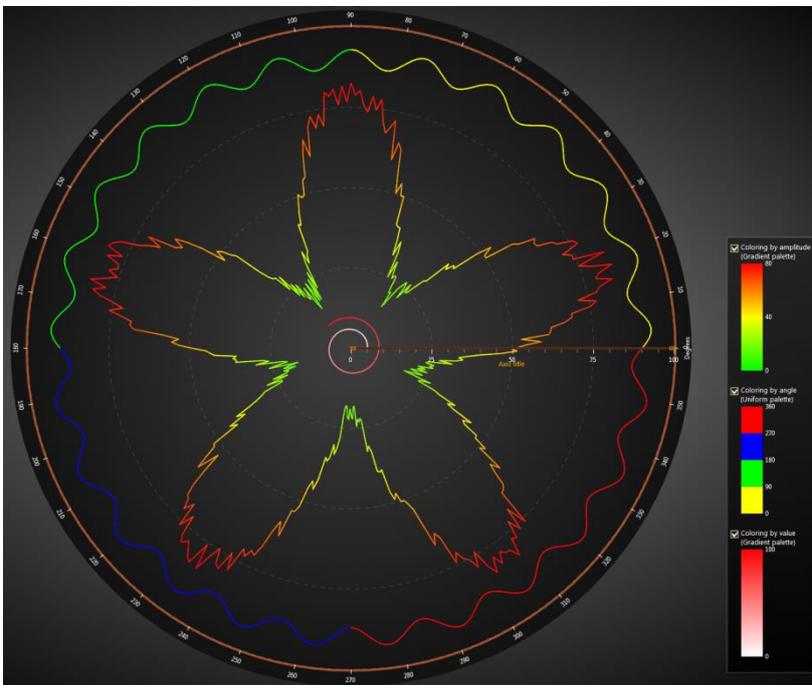


Figure 8-8. Palette coloring applied.

Use **ValueRangePalette** property to define the colors and value steps, it works in similar way than with ViewXY's and View3D's series.

## 8.2.3 Custom shaping and coloring with CustomLinePointColoringAndShaping event

Custom coloring and coordinate adjustment can be made with **CustomLinePointColoringAndShaping** event, which is called just before entering the rendering stage of the chart. It works in similar way than the **CustomLinePointColoringAndShaping** event in ViewXY's FreeformPointLineSeries, see 5.8.2.

## 8.3 AreaSeries

Area series allow data visualization in filled area style. The line style in the edge can be edited with **LineStyle** property. Fill can be changed with **FillColor** property.

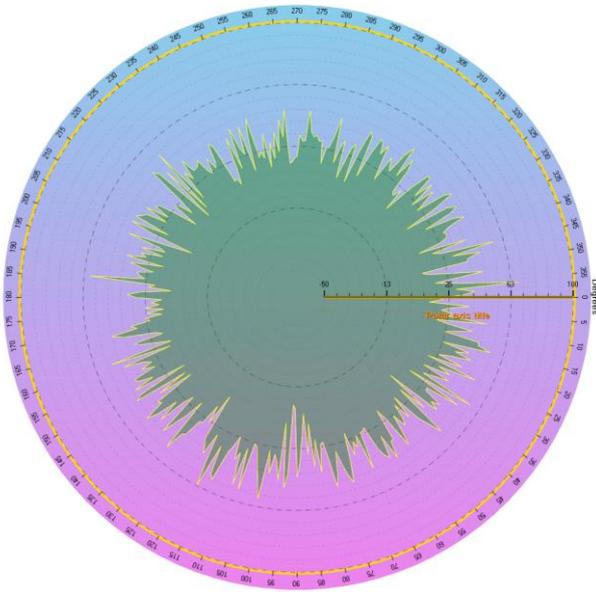


Figure 8-9. Some data presented with ViewPolar's AreaSeries.

### 8.3.1 Setting data

This code represents the data setting of previous figure.

```
int iCount = 360;
PolarSeriesPoint[] points = new PolarSeriesPoint[iCount];
Random rnd = new Random();

for (int i = 0; i < iCount; i++)
{
    points [i].Amplitude = 30f + rnd.NextDouble() * 5f *
        Math.Sin((double)i / 50f);
    points [i].Angle = (double)i;
}
chart.ViewPolar.AreaSeries[0].Points = points;
```

## 8.4 Sectors

**Sectors** can be defined to indicate some angular or amplitude range. Define amplitude range with **MinAmplitude** and **MaxAmplitude** properties. Define angular range with **BeginAngle** and **EndAngle**. You can move a sector by dragging it with mouse.

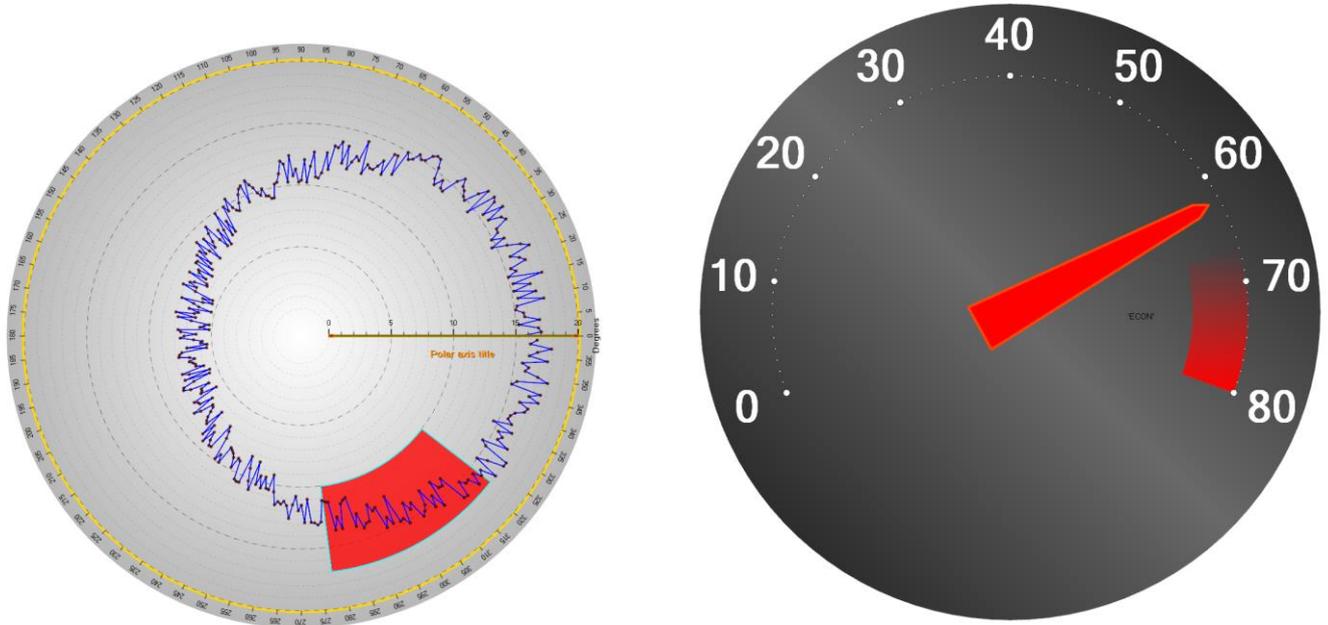


Figure 8-10. A couple of examples sectors are used in. In the first figure, there's a PointLineSeries and sector. In the second figure, a dial is made with AreaSeries and a sector represents RPM meter red zone.

## 8.5 Annotations

**Annotations** are almost similar to ViewXY's **Annotations**, see section 5.19, but **Target** is Location are defined in Polar axis values. Sizing by axis values is not suitable and therefore **Sizing** property has only values **Automatic** and **ScreenCoordinates**.

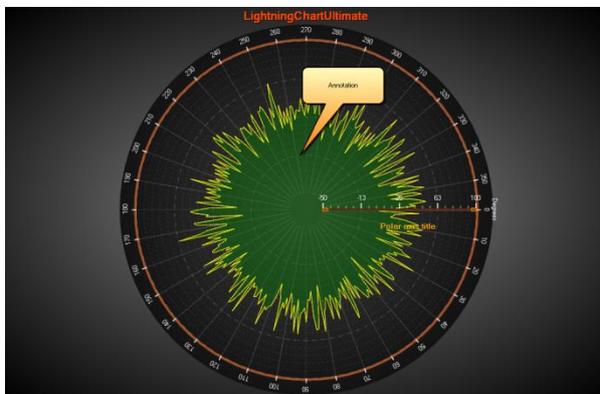


Figure 8-11. An annotation in Polar view.

## 8.6 Markers

Markers can be used to assign a specific data value at certain position. Assign the marker with a preferred axis by setting its **AssignPolarAxisIndex**. Define **Amplitude** and **AngleValue** properties to put it into place. Edit **Symbol** to have the appearance you prefer, and define text with **Label** property.

Markers can be moved by dragging it with mouse. Set **SnapToClosestPoint** to **Selected** or **All** to enable nearest data point snapping when dragging it. **Selected** tracks only the series this marker is set to snap to with **SetSnapSeries()** method. **All** tracks all series.

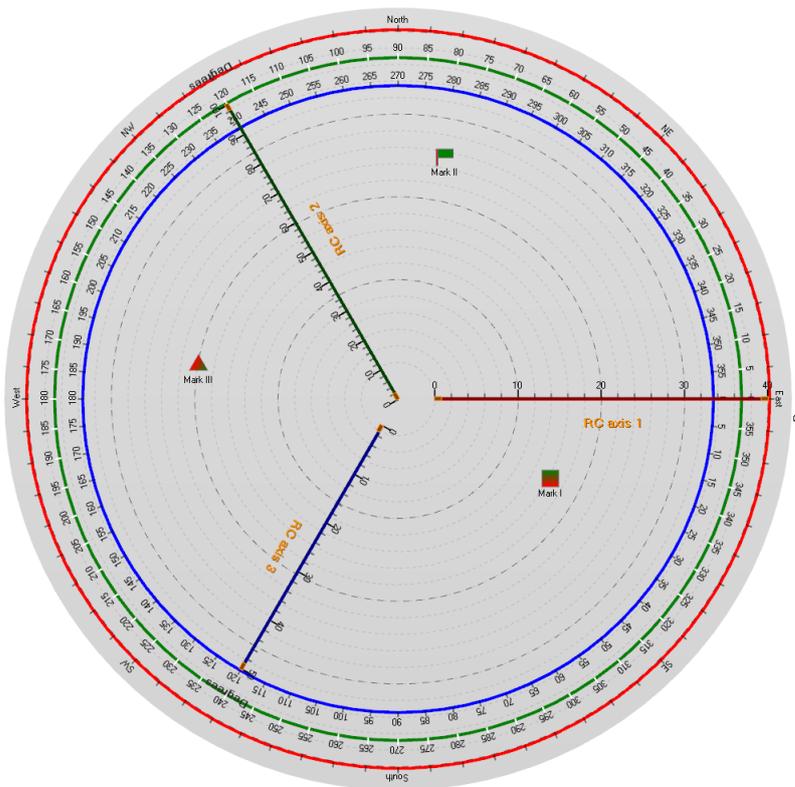


Figure 8-12. A couple of markers in a polar chart.

## 8.7 Zooming and panning

Zooming can be applied by code, by setting **ZoomCenter** and **ZoomScale** properties. **ZoomCenter** is defined as relative X-Y ranges.

X = -1: polar view's left edge at center of chart area  
X = 0: polar view's center at center of chart area  
X = 1: polar view's right edge at center of chart area

Y = -1: polar view's bottom edge at center of chart area  
Y = 0: polar view's center at center of chart area  
Y = 1: polar view's top edge at center of chart area

**ZoomScale** is the magnifying factor. E.g. 2 makes the chart appear twice as large in both X and Y direction than with 1.

Mouse-zooming features can be configured in **ZoomPanOptions** property tree.

ZoomPanOptions	
AxisMouseWheelAction	Pan
LeftMouseButtonAction	Zoom
MiddleMouseButtonAction	Pan
MousePanThreshold	5
MouseWheelRotateAction	Rotate
MouseWheelZooming	True
RectangleZoomAboutOrigin	False
▶ RectangleZoomingThreshold	
RightMouseButtonAction	Pan
RightToLeftZoomAction	DefaultView
ZoomFactor	2
▶ ZoomOutRectFill	
▶ ZoomOutRectLine	
▶ ZoomRectFill	
▶ ZoomRectLine	

Figure 8-13. ViewPolar's ZoomPanOptions.

## 9. ViewSmith

Smith charts are generally used in electronics in impedance measurements and impedance matching applications.

Smith chart plots the data in real in imaginary values ( $R + jX$ ) .

<b>Terms</b>
Impedance = $Z = R + jX$
R = Resistance, Real part
X = Reactance, Imaginary part
X > 0: Capacitive
X < 0: Inductive

The data position is determined on 2D-plot by angular on circular Real and Imaginary log-log scales.

ViewSmith	<b>Smith chart view</b>
Annotations	<b>(Collection)</b>
> Axis	<b>AxisSmithBase: Axis title</b>
> GraphBackground	
> LegendBox	<b>LegendBoxSmith</b>
> Margins	<b>80; 30; 30; 60</b>
Markers	<b>(Collection)</b>
PointLineSeries	<b>(Collection)</b>
> ZoomCenter	<b>17.8140703517588;2.181</b>
> ZoomPanOptions	
ZoomScale	<b>1</b>

Figure 9-1. ViewSmith property tree.

### 9.1 Axis

The Smith chart has only one real axis, which can be configured via extended property tree **Axis**, see figure below.

Axis	<b>AxisSmithBase: Ohm</b>
AngularAxisCircleVisible	True
AngularLabelsVisible	True
AngularTickStyle	
AngularUnitDisplay	<b>Radians</b>
AntiAliasing	True
AutoFormatLabels	True
AxisColor	 <b>Cyan</b>
AxisThickness	5
ClipGridInsideGraph	True
GridAngular	
GridDivCount	<b>12</b>
GridDivSpacing	80
GridImg	
GridReal	
GridType	<b>DivCount</b>
LabelsFont	<b>Microsoft Sans Serif; 9pt</b>
LabelTicksGap	5
MarginOuter	5
MouseHighlight	Simple
MouseInteraction	True
MouseScaling	True
RealAxisLineVisible	True
ReferenceValue	<b>200</b>
ScaleNibs	
ShowAbsoluteValues	True
TickMarkLocation	Outside
Title	<b>RoundAxis Title</b>
Units	<b>RoundAxis Title</b>
Visible	True

Figure 9-2. AxisSmith property tree.

Most of the properties are identical to the PolarView's axes and ViewXY's axes to customize and make it more attractive, there are added advanced properties especially for SmithView adjustment, e.g. **GridDivCount**, **GridImg** and **GridReal**, **RealAxisLineVisible**, **ShowAbsoluteValues**, **ClipGridInsideGraph**.

**GridDivCount** defines amount of circular grid lines on Real Axes and logarithmic grid lines on Imaginary scale.

**GridImg** and **GridReal** each of these properties is responsible for customizing the grid lines either on **Real** or **Imaginary** scales. In addition, there is a property to hide the grid, thus a user may hide one of them and continue to work with another.

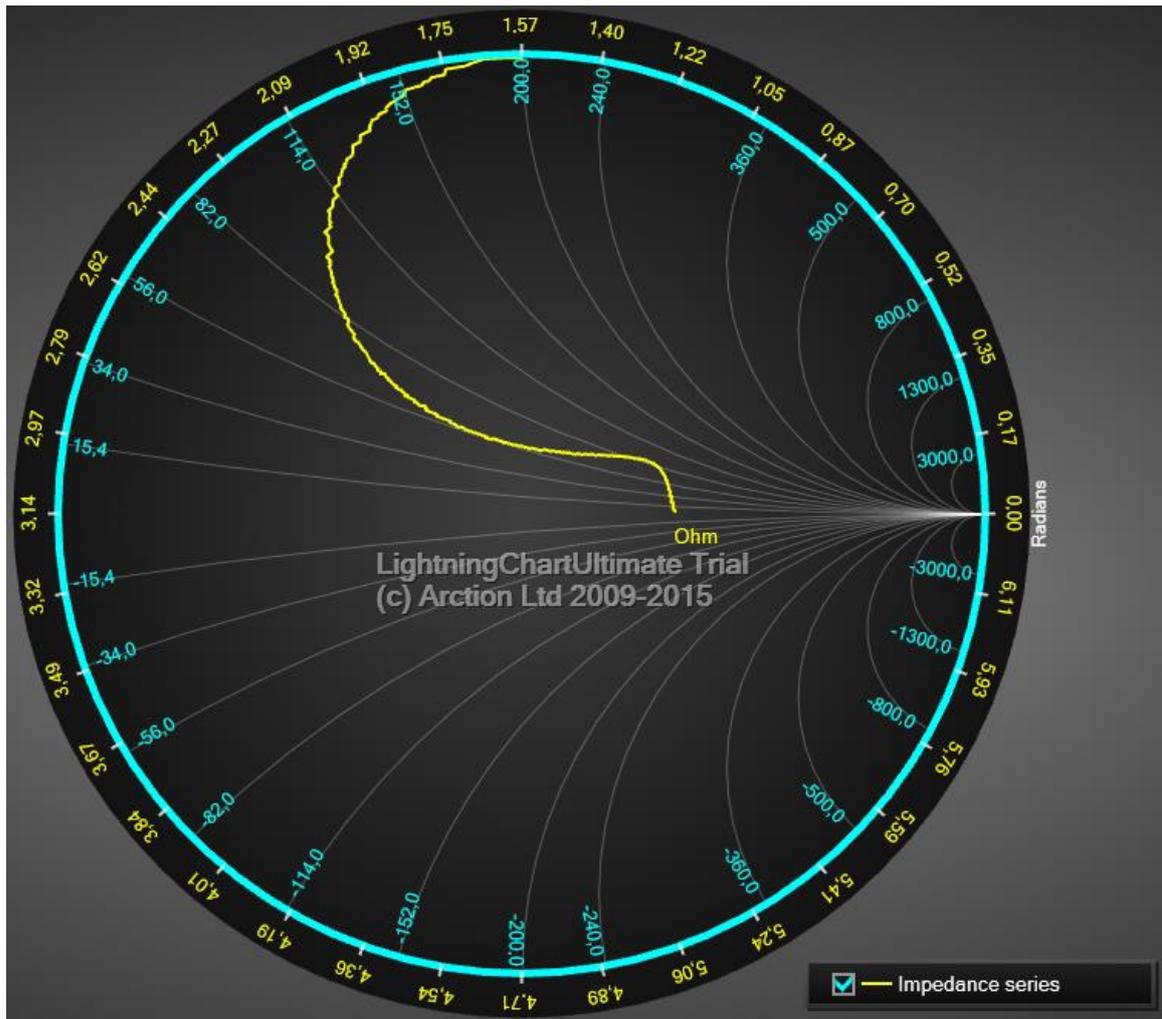


Figure 9-3. Real grid lines are hidden, Imaginary lines are visible.

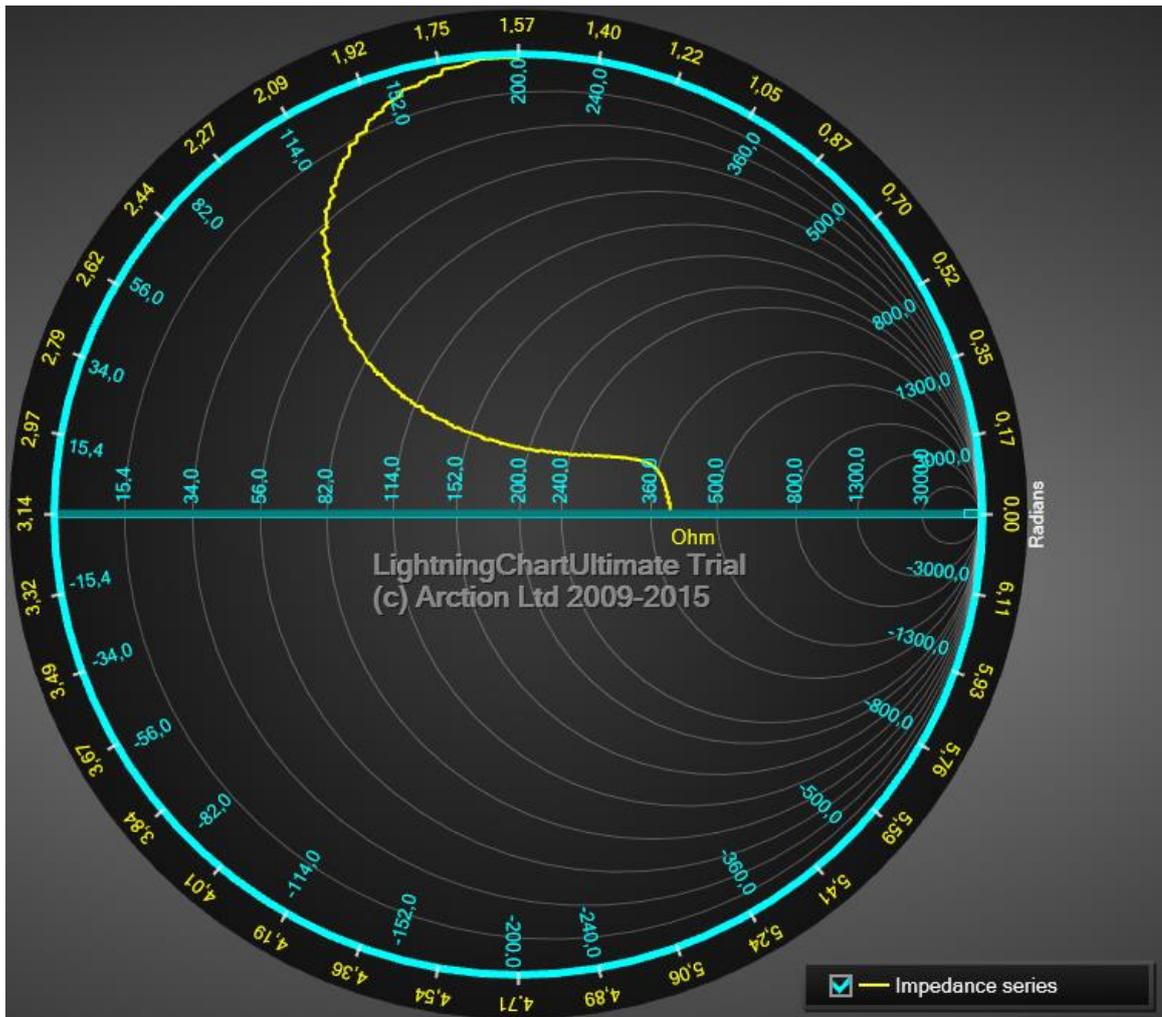


Figure 9-4. Imaginary grid lines are hidden, real lines are visible.

***RealAxisLineVisible*** this property hides the axis line, see Figure 9-3.

***ShowAbsoluteValues*** this property defines which values will be on scales (absolute or normalised).

***ClipGridInsideGraph***. Gridlines are visible outside the chart circle.

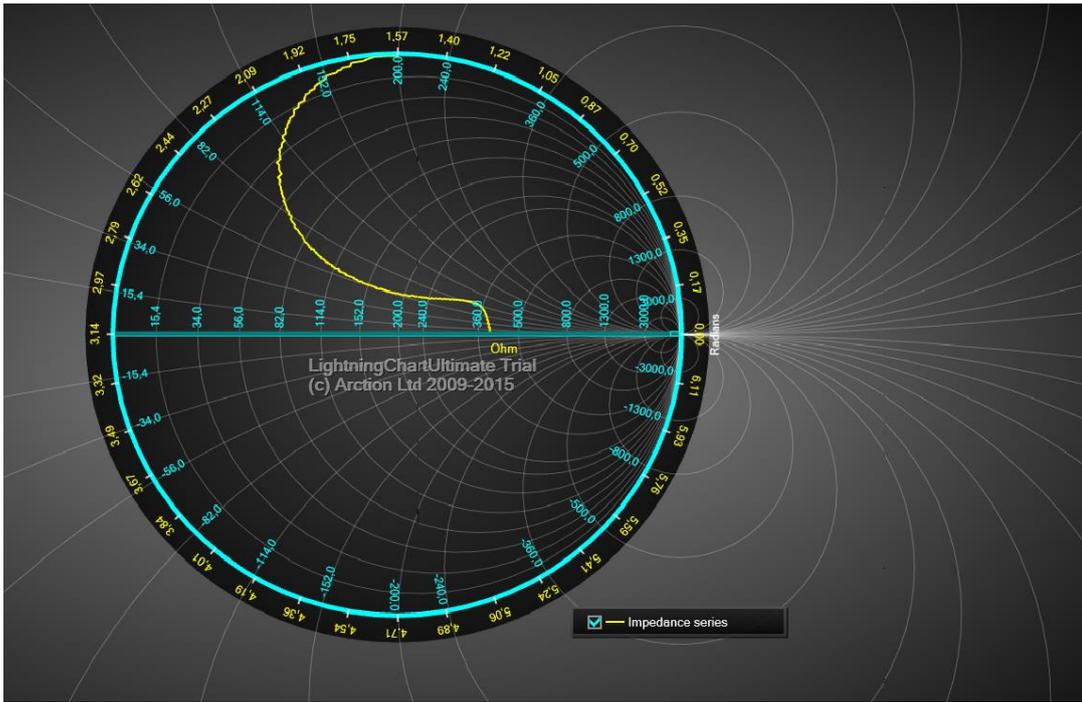


Figure 9-5. ClipGridInsideGraph = False.

The fully customized Smith chart, you can see below.

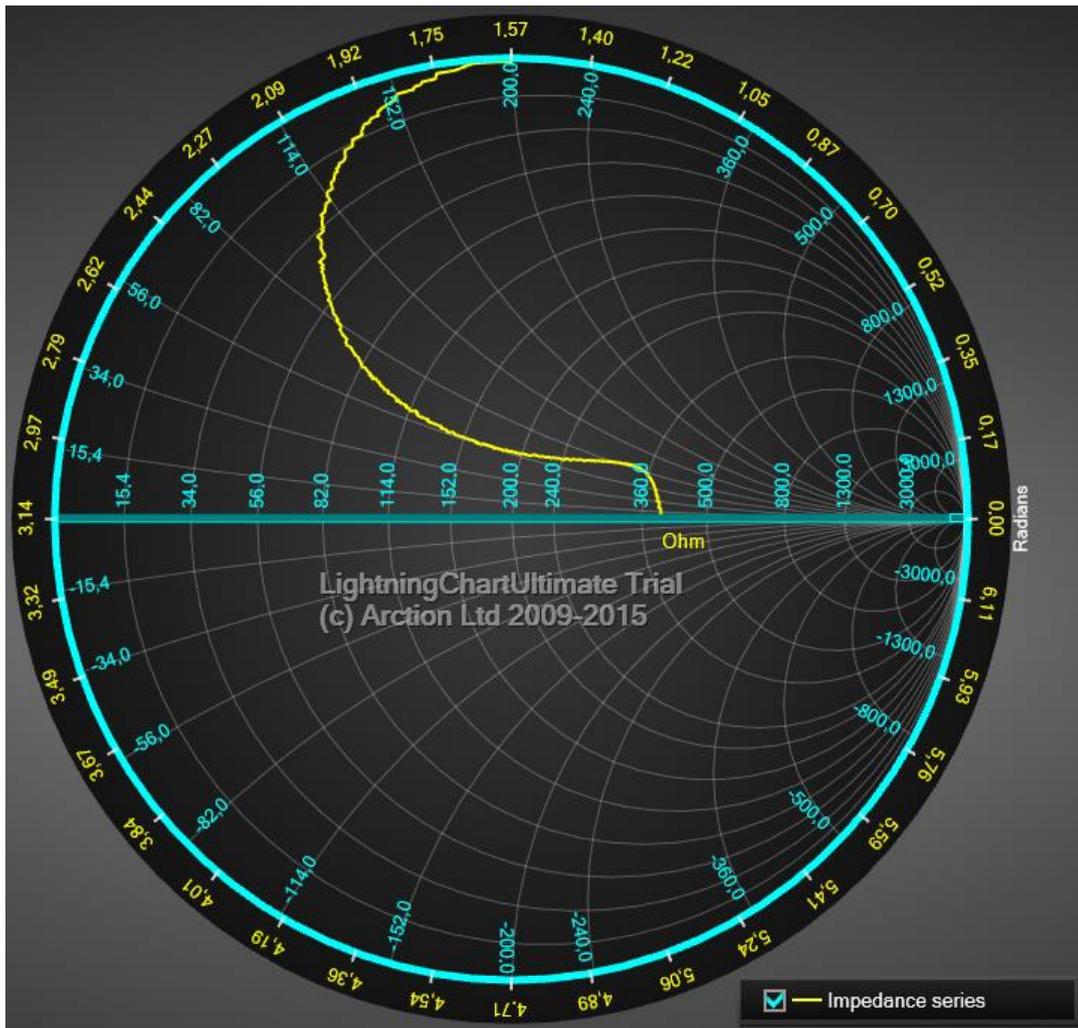


Figure 9-6. Customized Smith chart.

## 9.2 PointLineSeries

*SmithView's PointLineSeries* can be also used to draw a line, group of points or point-line as in *PolarView*. Lots of line and point styles are available in *LineStyle* and *PointStyle* properties.

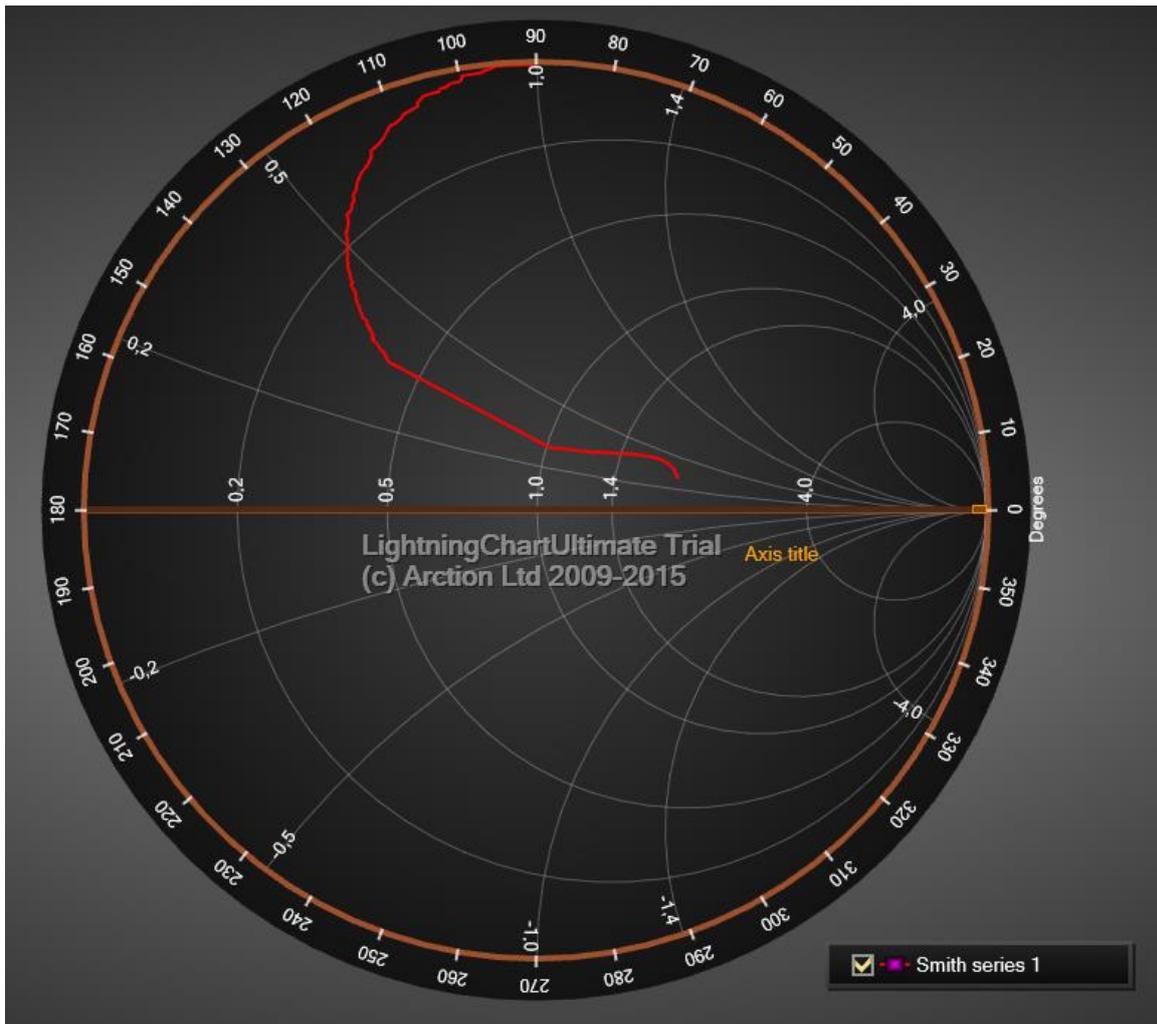


Figure 9-7. Smith data series.

### 9.3 Setting data

The code below, will add one set of data points to the collection of the Smith chart.

```
SmithSeriesPoint[] m_aPoints;
PointLineSeriesSmith Series = new PointLineSeriesSmith(m_chart.ViewSmith, axis);
//Create data for series
m_iCount = 5000;
m_aPoints = new SmithSeriesPoint[m_iCount];
for (int i = 0; i < m_iCount; i++)
{
    // Sine from left to right
    m_aPoints[i].RealValue = i * (MaxReal / m_iCount);
    m_aPoints[i].ImgValue = Math.Sin(0.01 * i)/Math.PI * MaxReal;
}
Series.Points = m_aPoints;
//Add series to chart
m_chart.ViewSmith.PointLineSeries.Add(Series);
```

## 9.4 Annotations

**Annotations** are identical to ViewPolar's **Annotations**, see section 8.5. **Target** is Location are defined in Polar axis values. **Sizing** property has only values **Automatic** and **ScreenCoordinates**.

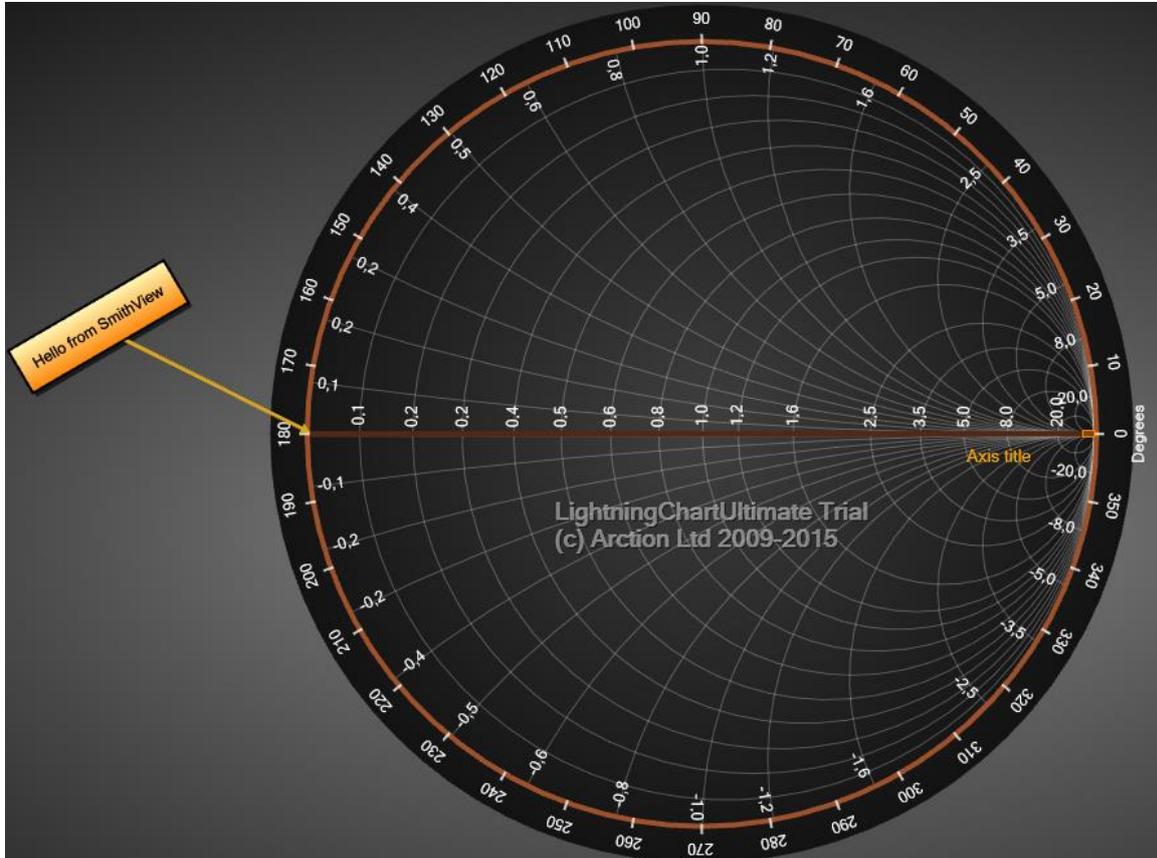


Figure 9-8. An annotation in Smith view.

## 9.5 Markers

Markers can be used a specific data value at certain position. Markers can be moved by dragging it with mouse. This property has identical definition with ViewPolar's markers, see section 8.6.

Assign the marker with a preferred axis by setting its **AssignPolarAxisIndex**. Define **Amplitude** and **AngleValue** properties to put it into place. Edit **Symbol** to have the appearance you want, and define text with **Label** property.

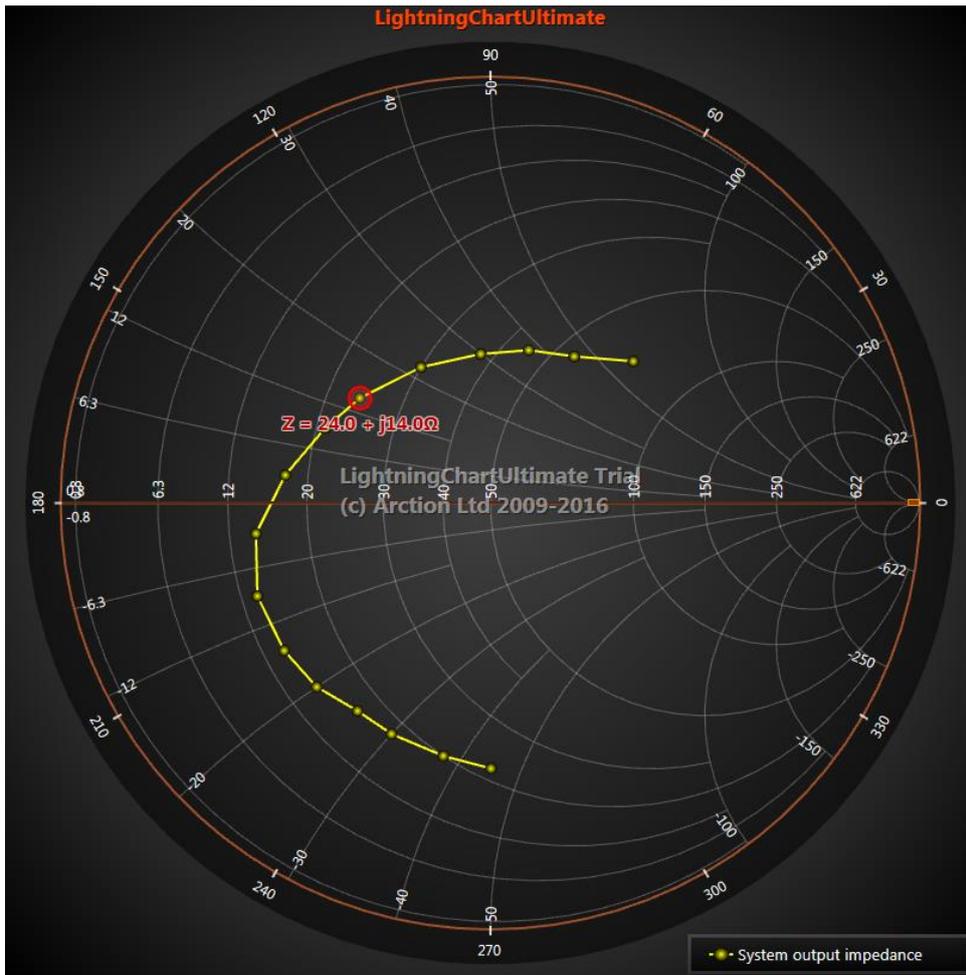


Figure 9-9. A marker tracking a series in Smith view.

## 10. Setting color theme

Set the color theme with **ColorTheme** property. Setting the theme will override majority of the object colors in the chart you've created. It is advised to first set the **ColorTheme** and then the object colors you want to be different.

**Note!** By setting the color theme, you will lose the colors that you've manually assigned in the Visual Studio property grid, without a warning.



Figure 10-1. Different color themes in use. On the left, default Dark theme with some custom colors. On the right, LightBlue theme set.

## 11. Scrollbars

Add one or more scrollbars in **HorizontalScrollBars** or **VerticalScrollBars** collection property. The appearance is fully customizable, allowing you to define even oval shaped buttons and scroll box. You can put for example a bitmap in the as button icon. Scrollbars can be used with all views, but the most apparent usage is in **ViewXY**.

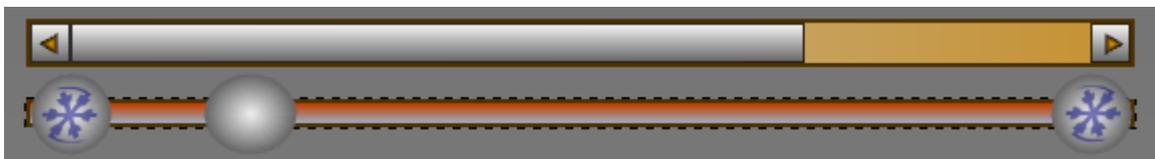


Figure 11-1. A couple of different looking scrollbars

**HorizontalScrollBar** can be aligned to fit the width of the graph by setting **Alignment** property to **BelowGraph**, **AboveGraph** or **GraphCenter**. By setting **None** as **Alignment**, position the scroll bar freely with **Offset** property, and size it with **Size** property.

**VerticalScrollBar** can be aligned to fit the height of the graph by setting **Alignment** property to **LeftToGraph**, **GraphCenter** or **RightToGraph**. By setting **None** as **Alignment**, position the scroll bar freely with **Offset** property, and size it with **Size** property.

The scrollbars have 64-bit unsigned integer value, instead of usual 32-bit signed integer value range. **Value** is the current position, **Minimum** is the minimum range value and **Maximum** is the maximum range value. This way, it can directly support long measurements with high sampling frequency. When **SampleDataSeries** are used in the measurement, set the sample index directly as scrollbar value. **Minimum** value represents the first sample index, and **Maximum** represents the last sample index.

**SmallChange** property is the amount of increment or decrement, when a scroll button is clicked. If **KeyControlEnabled** is active, you can use also arrow keys to change the **Value** by **SmallChange** amount. **LargeChange** represents a page change, which occurs when the scrollbar is clicked outside scroll box or scroll buttons. Use **PageUp** and **PageDown** keys to change the **Value** respectively. **MouseWheelChange** sets the change value when the mouse wheel is scrolled over the scroll bar.

In your code, use **Scroll** event handler to react to scrollbar value changes. Alternatively, you can use **ValueChanged** event handler. Scroll event handler provides more information of how the scroll has been done.

## 12. Export and printing

### 12.1.1 Bitmap image export

The chart can be exported as .PNG, .BMP and .JPG file with **SaveToFile()** method. **SaveToFile(...)** method allows exporting image files with resolution decrement and smoothing/anti-alias options. To export to a stream, use **SaveToStream()** method instead.

### 12.1.2 Vector image export

ViewXY, ViewPolar and ViewSmith can be also exported as .WMF, .EMF and .SVG formats. View3D and ViewPie3D don't support it. Use **SaveToFile** or **SaveToStream** method with selected vector file format.

**NOTE!** The vector output is simplified and all details, such as complex point styles, may be presented as a plain color and simple shape. The vector output may also contain some bitmap elements.

### 12.1.3 Copy to clipboard

The chart can be copied to clipboard by calling **CopyToClipboard(...)**. ViewXY, ViewPolar and ViewSmith can be copied with **CopyToClipboardAsEmf()** method in vector format.

### 12.1.4 Capturing to byte array

The chart has ***CaptureToByteArray*** method, to get as fast raw image data copy to external components or further processing of data.

#### **Usage**

```
int width;
int height;
byte[] aData = _chart.CaptureToByteArray(out width, out height);

Bitmap bitmap = new Bitmap(width, height,
    System.Drawing.Imaging.PixelFormat.Format32bppArgb);

System.Drawing.Imaging.BitmapData bitmapData = bitmap.LockBits(new
    System.Drawing.Rectangle(0, 0, width, height),
    System.Drawing.Imaging.ImageLockMode.ReadOnly,
    System.Drawing.Imaging.PixelFormat.Format32bppArgb);

IntPtr ipDst = bitmapData.Scan0;
int iRowByteCount = width * 4;
int iSrcIndex = 0;
for (int iY = 0; iY < height; iY++)
{
    Marshal.Copy(aData, iSrcIndex, ipDst, iRowByteCount);
    ipDst = new IntPtr(ipDst.ToInt64() + bitmapData.Stride);
    iSrcIndex += iRowByteCount;
}

bitmap.UnlockBits(bitmapData);
```

### 12.1.5 Setting output stream for continuous frame writing

Use chart.***OutputStream*** property set a stream where chart will write it's rendered frames into.

This property is intended as the fastest way to capture continuous frames from the chart, especially on ***Headless*** mode. (see Chapter 21.)

The stream is a raw byte stream, with each pixel described with 4 bytes, one byte per channel. The order of the channels depends on the renderer and its settings.

Use ***GetLastOutputStreamFormat*** and ***GetLastOutputStreamSize*** methods to find out the format and output size of the last written image.

Produced image size should be of the size of the chart, in pixels.

**NOTE!** On the contrary to the other properties of the LightningChart, set stream is NOT disposed on chart's dispose.

**NOTE!** On the contrary to the other properties, setting this property will not cause new frame to be rendered.

## 12.1.6 Printing

Call **PrintPreview()** method to open a print preview dialog or **Print()** to directly print with default settings. Call **Print(...)** to print with manual settings. Print ViewXY, ViewPolar and ViewSmith support vector printing as well. Supply **Raster** or **Vector** format to parameter to Print method.

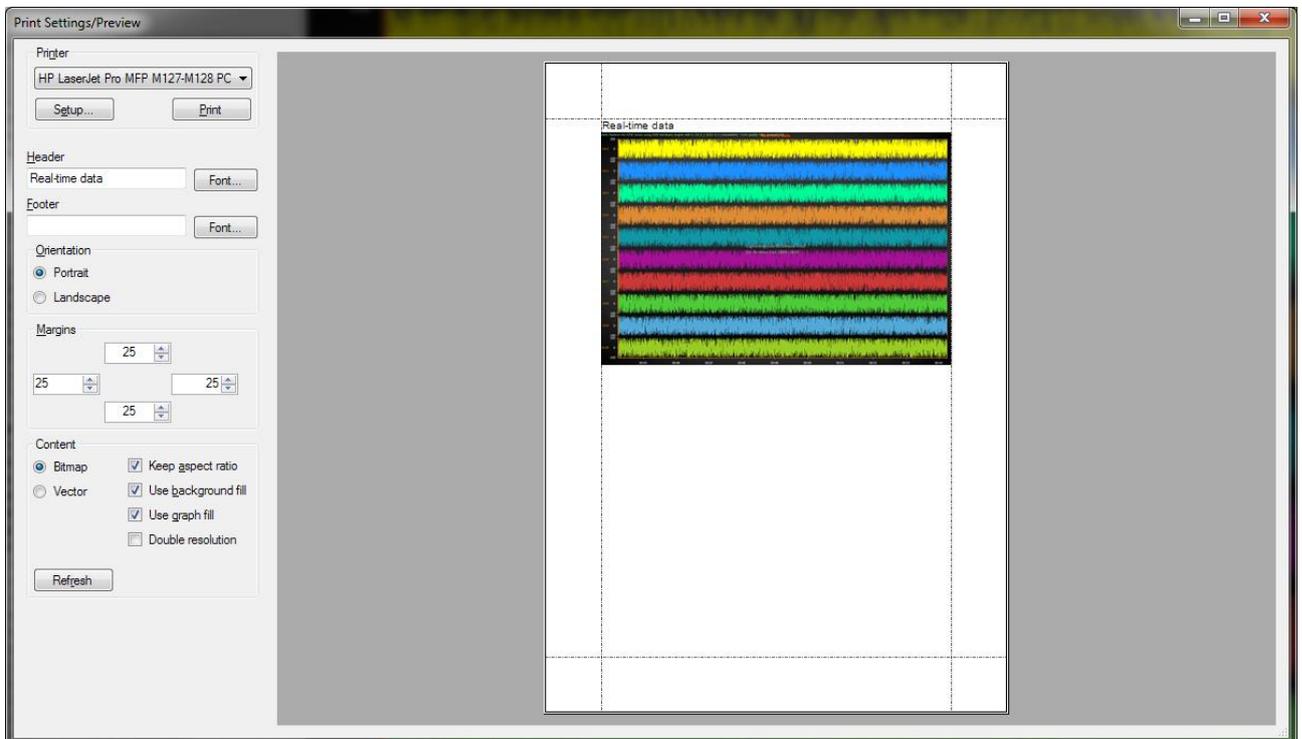


Figure 12-1. Print preview dialog.

## 13. LightningChart performance

### 13.1 Selecting the correct API edition

Select the chart edition instructed in section 1.1. Do not use series data binding features if not necessary.

### 13.2 Set the rendering options correctly

LightningChart's DirectX9 rendering engine maybe slightly faster than DirectX11 engine in specific applications, but generally, leaving DirectX11 as preferred renderer is a good choice. DirectX11 in the also gives better appearance.

Fonts quality setting is important as well.

See section 4.6.

### 13.3 Updating chart data or properties

Every property or series data value change will cause the LightningChart control to be redrawn. Every redraw will cause CPU and display adapter overhead. If you programmatically change more than one property at same time, you should make the property changes between **BeginUpdate()** and **EndUpdate()** method calls, as a batch. **BeginUpdate()** will stop drawing the control until **EndUpdate()** is called. There's internal counter for pending **BeginUpdate()** calls and when equal amount of **EndUpdate()** calls have been reached, **EndUpdate()** redraws the control. The following example demonstrates how to update chart with minimal load to the computer.

```
chart.BeginUpdate() ;//Disable redraws

//Add data to series
chart.ViewXY.SampleDataSeries[0].AddSamples(multiChannelSampleStream[0],
false);
chart.ViewXY.SampleDataSeries[1].AddSamples(multiChannelSampleStream[1],
false);
chart.ViewXY.SampleDataSeries[2].AddSamples(multiChannelSampleStream[2],
false);

//Update point counter bar
chart.ViewXY.BarSeries[0].SetValue(0,1,(double)totalPointsCollected,"",
false);
```

```

//Update point counter label
chart.Title.Text = totalPointsCollected.ToString();

//Set monitoring scroll position to latest x
newestX = firstSampleTimeStamp + (double)(pointsLen - 1) / genSampFreq;
chart.ViewXY.XAxes[0].ScrollPosition = newestX;

chart.EndUpdate() ;//Enable redraws and redraw

```

The internal counter allows using nesting updates as follows:

```

void MainMethod()
{
    chart.BeginUpdate() ;
    chart.Title.Text = "My title";
    chart.ViewXY.XAxes[0].AxisColor = Colors.Red;

    UpdateSeriesColors();

    chart.EndUpdate() ;
    //repaints only once.
}

private void buttonCreate_Click(object sender, EventArgs e)
{
    UpdateSeriesColors(); //repaints only once
}

void UpdateSeriesColors()
{
    chart.BeginUpdate() ;

    foreach(PointLineSeries series in chart.ViewXY.PointLineSeries)
    {
        series.LineStyle.Color = Color.Yellow;
    }

    chart.EndUpdate() ;
}

```

## 13.4 Line series tips

- When using a line series, use **SampleDataSeries**, if it is suitable for your application. It is drawn fastest and it doesn't need as much memory as other line series types.
- Set **PointsVisible** property false, if you don't need the points to be visible.
- Set line width to 1 with **LineStyle.Width** property.
- Use solid line style by setting **LineStyle.Pattern** to **Solid**.
- Disable anti-aliasing by setting line series **LineStyle.AntiAliasing** to **None** and set chart's **AntiAliasLevel** to 0.
- Disable all mouse interactivity, by setting **MouseInteraction** of a series to false. Alternatively, disable whole chart's mouse interactivity by setting chart's **chart.Options.MouseInteraction** to false.

## 13.5 Intensity series tips

Applies to: *IntensityGridSeries*, *IntensityMeshSeries*

- Change the **Optimization** property of the series to **StaticData**, if the data won't be updated continuously. **DynamicData** is better choice if data is changed many times per second.
- Use **Optimization: DynamicValuesData** to update only the **Value** fields of **Data** array's **IntensityPoint** structures, and call **InvalidateValuesDataOnly** method to update the chart. This way, the update is much faster as the geometry of the series is not recalculated. This is only intended to be used for applications where the data X and Y values of the nodes stay in the same place, like thermal imaging solutions.

Applies to: *IntensityGridSeries*

- For high-resolution thermal imaging applications, enable **PixelRendering** for *IntensityGridSeries*.
- For rapidly updating data sets, use **SetValuesData** and **SetColorsData** methods instead of using **Data** property, to save memory and to improve performance.

## 13.6 3D surface series tips

Applies to: *SurfaceGridSeries3D*, *SurfaceMeshSeries3D*, *WaterfallSeries3D*

- Disable lighting if light reflections and shading are not needed, by setting **SuppressLighting** to false.
- If contour lines are used, use **FastColorZones** or **FastPalettedZones** instead of **ColorLines** or **PalettedColorLines**.

Applies to: *SurfaceGridSeries3D*

- With scrolling data (like 3D spectrum or spectrogram), use **InsertRowBackAndScroll** and **InsertColumnBackAndScroll** methods to update data and axis ranges.

## 13.7 Maps tips

Applies to: **ViewXY.Maps**

- Set **ViewXY.Maps.Optimization** to **CombinedLayers**, when you typically keep the X and Y axis ranges same, and present other data over the maps. This way, the map layers are rendered into same buffer image, resulting into more efficient rendering.
- Set **ViewXY.Maps.Optimization** to **None**, if you need to display the map titles over *IntensityGrid* or *IntensityMesh* series.

## 13.8 Hardware

When getting the absolute maximum performance for your application, the computer hardware must be powerful. In many applications, display adapter power is more important than CPU power. Use as modern display adapter as possible. DirectX 9.0c level display adapters work. 'c' comes from DirectX *Shader model 3*, which is required by some effects.

You can find out if some feature is not supported by the display adapter used, by calling ***GetRenderDeviceInfo()*** method. Especially if returned information states that ***FastVertexFormat*** is not supported, it is a bad thing for performance.

Note! LightningChart is GPU hardware accelerated chart. Without a good GPU, the performance may be much lower than in optimal case. A good resource to compare the performance of different GPUs is **PassMark's Video Card Benchmarks** ([http://www.videocardbenchmark.net/gpu\\_list.php](http://www.videocardbenchmark.net/gpu_list.php)). A video card having 10x better score than other, can be also 10 times faster in LightningChart use, but overall difference in refresh rate is rarely that great, since other computer hardware may become bottleneck.

## 14. LightningChart error and exception handling

Enable ***chart.Options.ThrowChartExceptions*** property if you want the errors to be thrown as exceptions. Alternatively, define an error event handler for ***ChartError*** event. The exceptions are thrown as ***ChartException*** objects, with an enumerated ***ErrorType*** value and a string description of the error. ***ErrorType*** can also be ***Information***, which is not an error and should not be treated as one. You can convert error type to string with ***ChartTools.ErrorTypeToString(...)*** method.

When having any mystic problem with the chart, ensure you have a working error/exception handler in your application which brings those errors visible for the user and tells the reason.

## 15. ChartManager component

Add ChartManager control to your form. Then, assign the manager control to **ChartManager** property of all LightningChartUltimate controls.

### 15.1 Chart interoperation, drag-drop

ChartManager control can be used to coordinate interoperation of several LightningChartUltimate controls. It enables series drag-drop from chart to another, in WinForms. For WPF it's not usable for technical reasons.

Series have **DisableDragToAnotherAxis** property, and it must be set to **False** to enable the dragging. It is **True** by default.

Axes have **AllowSeriesDragDrop** property which can be set to **False** to prevent dragging over specific axis. Default is **True**.

Move mouse over the series to be dragged, press left mouse button down to start dragging.

Dragging over Y axis: Drag the series over Y axis of other chart and release the button. The other chart takes the ownership of the series and the series is assigned to the target Y axis. It also assigns **the first X axis** for the series.

Dragging over X axis: Drag the series over X axis of other chart and release the button. The other chart takes the ownership of the series and the series is assigned to the target X axis. It also assigns **the first Y axis** for the series.

### 15.2 Memory management enhancement

In some extreme real-time monitoring applications the .NET garbage collector does not free unused memory nicely enough, if the application is run with high CPU load. Garbage collector frees the memory all at once, causing a visible 'freeze' or 'pause' in chart updating. To make the chart updates smoother, enable ChartManager's **MemoryGarbageCollecting** property. By doing that, a separate thread is used to free the memory more often regardless of the CPU load. Using **MemoryGarbageCollecting** is recommended to be used with multi-core processors, as the thread running will slightly load the CPU.

## 16. SignalGenerator component

**SignalGenerator** component can be used to generate real-time signal. The signal is produced as sum of different waveforms. Several **SignalGenerator** components can be connected together with master-slave relationship, to produce a synchronized, multi-channel output. **SignalGenerator** is very useful when developing signal monitoring or data acquisition software with LightningChart.

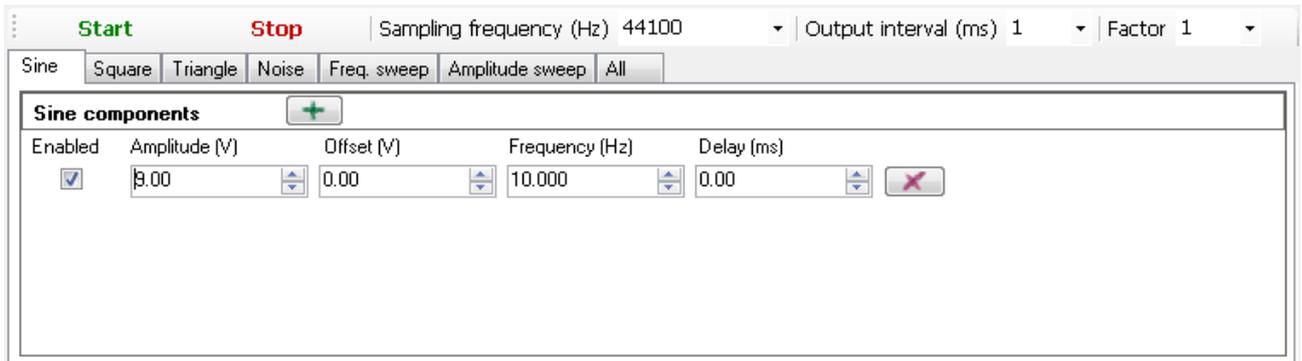


Figure 16-1. Signal generator component with Sine page selected.

The waveforms are divided into following categories: **Sine**, **Square**, **Triangle**, **Noise**, **Frequency sweep**, and **Amplitude sweep**. You notice the tab pages for them in the component. In Sine page, you can add the sine waveforms. In Square page, you can add the square waveforms. In Triangle page, you can add the triangle waveforms. In Noise page, you can add random noise waveforms. In Frequency sweep page, you can add frequency sweeps. In Amplitude sweep page, you can add amplitude sweeps. In **All** page, you can set all waveforms in stacked view.

### 16.1 Sampling frequency, Output interval and Factor

**Sampling frequency** tells how many signal points are generated per second. Higher sampling frequency produces more accurate signal, but with cost of increased dataflow and overhead. With high sampling frequency, you can present signals containing high frequencies. Sampling frequency must be more than twice the maximum signal frequency, to fulfill Nyquist sampling theorem.

**Output interval** sets the preferred interval of calculated output samples, in milliseconds. For example, if you set here 100, you will get a bundle of samples 10 times per second, after every 100 ms period. Using lower values here will give smoother real-time monitoring output. Note that output interval is not accurate, and may vary with computer load. The output data stream automatically gives more samples out if the period has been longer than expected, and the data stream will get in shape also with high data rates and under heavy computer overhead.

**Factor** multiplies the output samples with selected value. For example, to generate mV signal instead of V signal, set Factor to 1E-3.

## 16.2 Sine waveforms

Sine waveform is constructed with **Amplitude**, **Offset**, **Frequency** and **Delay** parameters. **Amplitude** is the maximum voltage difference from zero level. Note that the total range is bipolar. Peak-to-peak value will be  $2 * \text{Amplitude}$ . **Offset** is DC level added to the signal, positive values shift the signal up and negative values shift it down in the value range. **Frequency** tells the signal cycle count in Hertz. One cycle per second is frequency of 1 Hertz. **Delay** delays in the signal in milliseconds.

A simple sine waveform with the following settings

Sine components				
Enabled	Amplitude (V)	Offset (V)	Frequency (Hz)	Delay (ms)
<input checked="" type="checkbox"/>	40,00	-5,00	20,000	0,00

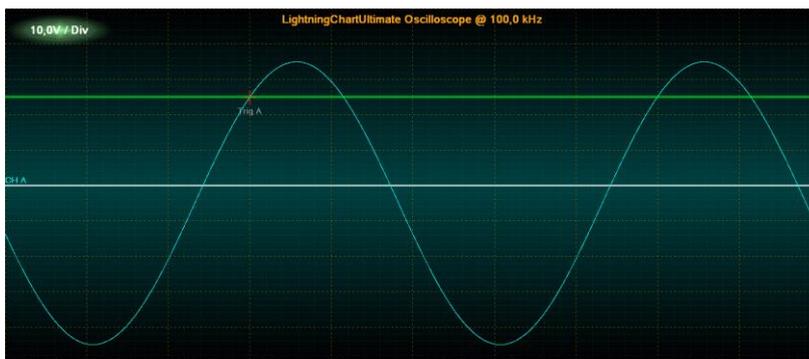


Figure 16-2. One sine waveform signal.

Respectively, two sine waveforms with the following settings

Sine components					
Enabled	Amplitude (V)	Offset (V)	Frequency (Hz)	Delay (ms)	
<input checked="" type="checkbox"/>	40,00	-5,00	20,000	0,00	<input type="checkbox"/>
<input checked="" type="checkbox"/>	10,00	0,00	60,000	0,00	<input type="checkbox"/>

produces signal like the following figure.

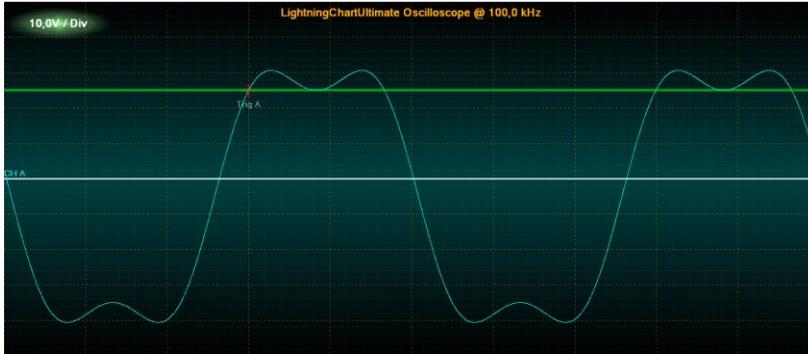


Figure 16-3. Two sine waveforms signal.

### 16.3 Square waveforms

Square waveform has one more parameter, **Symmetry**. The range for symmetry is 0...1. **Symmetry** tells how long the signal is in high state, related to cycle period. With Symmetry of 0.5 the low and high states of the signal are of equal lengths.

Square waveform with the following settings

Square components					
Enabled	Amplitude (V)	Offset (V)	Frequency (Hz)	Delay (ms)	Symmetry
<input checked="" type="checkbox"/>	35,00	0,00	30,000	0,00	0,80

produces a signal like the following.

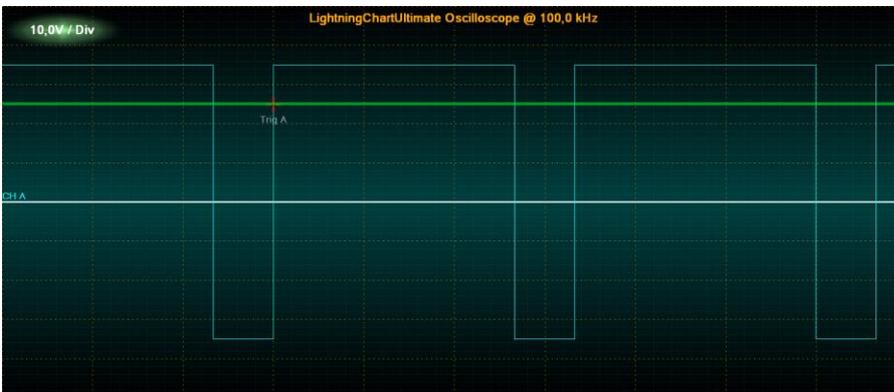


Figure 16-4. One square waveform signal, with symmetry of 0.8.

## 16.4 Triangle waveforms

Triangle waveform has also **Symmetry** parameter. It controls side and amount the triangle leans. 0.5 is the value for symmetrical triangle. Values under 0.5 lean left and values over it lean right.

Triangle waveform with the following settings

Triangle components					
Enabled	Amplitude (V)	Offset (V)	Frequency (Hz)	Delay (ms)	Symmetry
<input checked="" type="checkbox"/>	40,00	0,00	25,000	0,00	0,70

produces a signal like the following

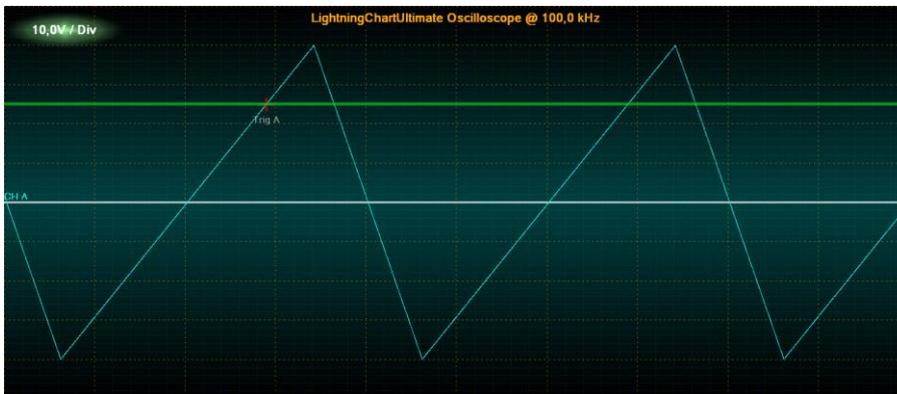


Figure 16-5. One triangle waveform signal, with symmetry of 0.7.

## 16.5 Noise waveforms

**Noise** waveform is a randomly generated signal. Points get randomized between  $-Amplitude$  and  $+Amplitude$ .

Noise waveform with the following settings

Noise components		
Enabled	Amplitude (V)	Offset (V)
<input checked="" type="checkbox"/>	30,00	0,00

produces a signal like the following

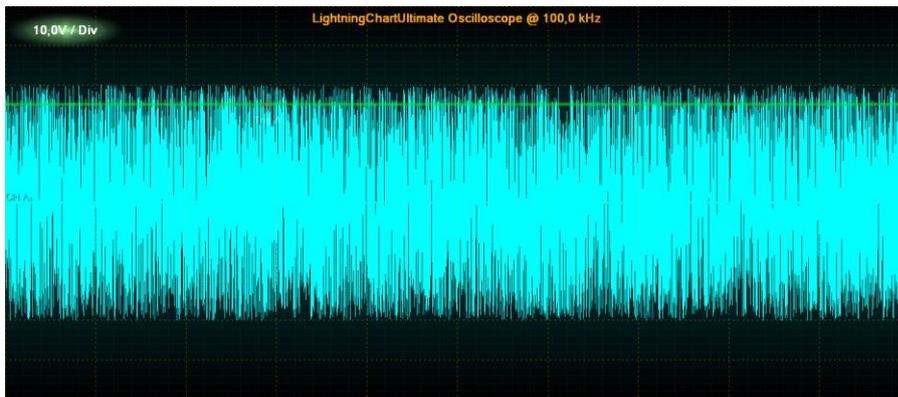


Figure 16-6. Noise waveform signal.

## 16.6 Frequency sweeps

**Frequency sine sweep** runs from frequency 1 to frequency 2 in given time period, with constant amplitude. Use **Amplitude** to set the constant amplitude, **FrequencyFrom** to set start frequency, **FrequencyTo** to set end frequency and **DurationMs** to set the duration in milliseconds.

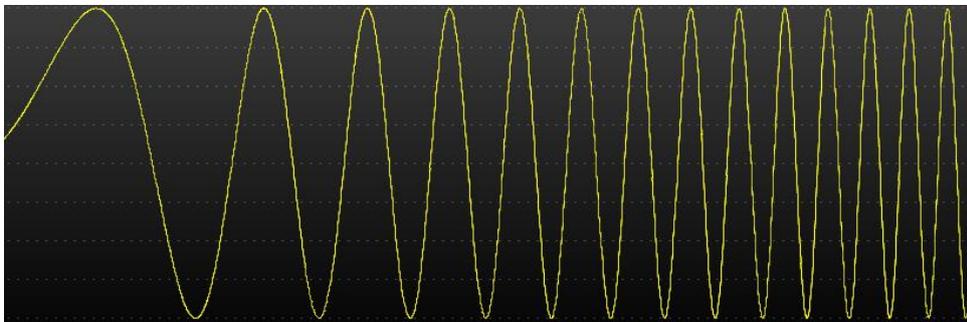


Figure 16-7. Frequency sweep.

## 16.7 Amplitude sweeps

**Amplitude sine sweep** runs from amplitude 1 to amplitude 2 in given time period, with constant frequency. Use **Frequency** to set the constant frequency, **AmplitudeFrom** to set start amplitude, **AmplitudeTo** to set end amplitude and **DurationMs** to set the duration in milliseconds.



Figure 16-8. Amplitude sweep.

## 16.8 Starting and stopping

Start the generator by pressing Start button or calling **Start** method. Stop the generator by pressing Stop button or calling **StopRequest** method. **Stopped** event will fire when stopping is complete.

## 16.9 Multi-channel generator with master-slave configuration

Several **SignalGenerator** components can be connected together to produce a synchronized, multi-channel output.

**Master generator** controls sampling frequency, start, stop and output of all generators. Master generator produces the first channel in the output data stream.

**Slave generator** is connected to master generator by assigning its **MasterGenerator** property. Define the signal waveforms freely. Slave generator is started and stopped by the master generator. Slave generator gets output data stream channel index in connection order. The slave generators must be connected before starting the master generator.

## 16.10 Output data stream

The output is a two-dimensional array, obtained with a ***NewSignalPointsGenerated*** event handler. The event is raised approximately after every **Output interval**.

The event handler obtains reference to samples array, and the time stamp for first samples bundle of received this round. First dimension of samples array represents channels and second dimensions samples for each channel. All channels have equal sample count.

The event raised is like the following:

```
private void m_signalGenerator_DataGenerated(DataGeneratedEventArgs args)
```

To investigate the channel count of the data stream, get the length of first dimension

```
channelCount = args.Samples.Length;
```

To get the sample count for channels

```
sampleBundleCount = args.Samples[0].Length;
```

To forward this data directly to ***SampleDataSeries*** list of LightningChart, and update real-time monitoring scroll position, use code like the following:

```
private void m_signalGenerator_DataGenerated(DataGeneratedEventArgs args)
{
    chart.BeginUpdate();
    int channelIndex = 0;
    int sampleBundleCount = args.Samples[0].Length;
    foreach (SampleDataSeries series in chart.ViewXY.SampleDataSeries)
    {
        series.AddSamples(args.Samples[channelIndex++], false);
    }

    //Set latest scroll position x
    newestX = args.FirstSampleTimeStamp + (double)(sampleBundleCount - 1) /
    generatorSamplingFrequency;
    chart.ViewXY.XAxes[0].ScrollPosition = newestX;
    chart.EndUpdate();
}
```

Note that with `args.Samples[0]` you can access the master generator's data. `args.Samples[1]` gives access to first slave generator data, `args.Samples[2]` to second slave etc.

## 17. SignalReader component

**SignalReader** component allows reading data from a signal source file, and playing it back with selected rate. **SignalReader** output data stream format is similar to **SignalGenerator**, see chapter 16.10.

SignalReader component currently supports a couple of formats, wav and sid.

### 17.1 Key properties

**FileName** defines the file to opened, like "c:\\wavedata\\audioclip1.wav"

**Factor** sets the output factor. Raw signal samples are multiplied with this value.

**OutputInterval** is similar to **SignalGenerator's** property, see section 16.1.

**IsLooping** allows file read to jump to the beginning of the file, when the end of file has been reached.

After the file has been opened, the following properties can be used to get information of the file:

**ChannelCount**: the channel count of the file.

**SamplingFrequency**: sampling frequency in Hz.

**FileSize**: File size in bytes.

**Length**: Sample count for each channel. It may not be exact for all signal file formats.

**IsReaderEnabled**: Status telling is the component started and reading data. If **Looping** is set to false and end of file is reached, **IsReaderEnabled** will change to false.

### 17.2 Opening file quickly for playback

Call **OpenFile(...)** method supplied with a file name. File name must have an extension of supported formats. Then, call **Start()** method

```
signalReader.OpenFile("c:\\wavedata\\audioclip1.wav");  
signalReader.Start();
```

It will start playback of a PCM-formatted WAV file.

Playback can be stopped by calling **StopRequest()** method.

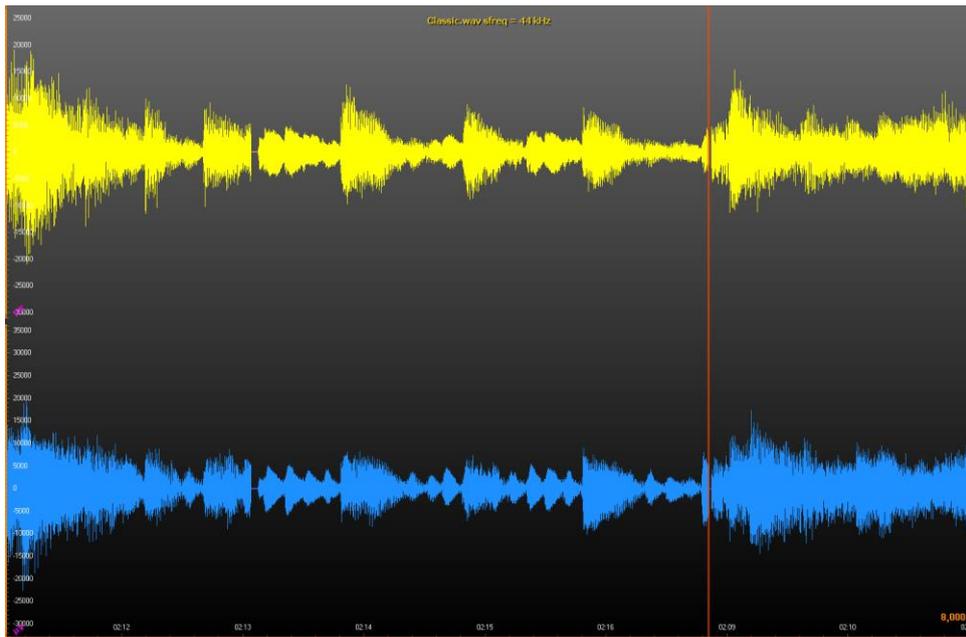


Figure 17-1. SignalReader reads a wav file and LightningChart SampleDataSeries draw the signal. A cursor line is used to mark current reading position and X-axis scroll position.

## 18. AudioInput component

**AudioInput** component allows user to capture signal from Windows' recording device to System.Double values. These values can then be rendered on LightningChart Ultimate, send to an AudioOutput component, saved to a file etc...

### 18.1 Properties

**BitsPerSample** – Gets or sets how many bits are allocated per sample. Supported values are 8 and 16. If other value is used, 16 is used instead. Is settable when **IsInputEnabled** is false.

**IsInputEnabled** – Gets or sets the state of this instance (i.e. starts or stops it). Setting this property true is the same as calling **Start** method where false is the same as calling **Stop** method.

**IsStereo** – Gets or sets whether to use two channels (stereo) or just one (mono). Is settable when **IsInputEnabled** is false.

**LicenseKey** – Gets or sets license key in normal or encrypted format.

**RecordingDevice** – Gets or sets the current recording device. Is settable when **IsInputEnabled** is false. By setting this property null, Windows' default recording device is used.

**SamplesPerSecond** – Gets or sets sampling frequency. Is settable when **IsInputEnabled** is false.

**ThreadInvoking** – Gets or sets whether this instance automatically synchronizes its events to the main UI thread hence eliminating the need to call Control.Invoke method on caller's side.

**Volume** – Gets or sets volume (0-100). Is settable when **IsInputEnabled** is false.

### 18.2 Methods

**GetRecordingDevices** – Use this static method to get a list of available Windows recording devices.

**RequestStop** – Signals this AudioInput instance to stop. Stop does not occur immediately after exiting this method. By subscribing to Stopped event, caller is notified when everything has stopped.

**Start** – Starts reading audio from selected recording device. Started event is triggered when internal thread is about to start.

## 18.3 Events

**DataGenerated** – Occurs when a new set of audio data has been generated. Data and its first sample's time stamp can be read from a *DataGeneratedEventArgs* object that is provided as a parameter.

**Started** – Occurs when audio input has been started. *StartedEventArgs* object that is provided as a parameter, contains three public fields: *BitsPerSample*, *ChannelCount* and *SamplesPerSecond*.

**Stopped** – Occurs when audio input has been stopped.

## 18.4 Usage (WinForms)

This chapter describes the usage of WinForms version of *AudioInput* class. WPF version will be handled in chapter 17.5.

### 18.4.1 Creation

Create a new **AudioInput** instance either by manually in your source code or by dragging and dropping it from Visual Studio's toolbox on to your form, user control etc.

If you do not need to show the GUI (i.e. you use your own or you control **AudioInput** object from your source code) then set **Visible** property false. Parent property is always recommended to set so that when the parent control is disposed of, **AudioInput** instance gets disposed of automatically. If there is no parent then do not forget to call *Dispose* method when you are done with **AudioInput** instance. Note, if you create a new **AudioInput** instance via Visual Studio's toolbox then **Parent** is automatically set.

It is recommended to set **LicenseKey** property so that your **AudioInput** instance uses an explicit license key instead of trying to find one from Windows' registry. Note, if you are using a trial version/license then you can leave **LicenseKey** property to its default value.

### 18.4.2 Event handling

To get new samples from **AudioInput** instance you need to subscribe at least to **DataGenerated** event. When **DataGenerated** event is triggered you can get the new samples and the first sample time stamp from a *DataGeneratedEventArgs* object provided as a parameter.

If you want to know when **AudioInput** instance has started its audio sampling task you subscribe to Started event. You can get the number of bits per sample, is audio mono or stereo and how many samples per second are generated from **StartedEventArgs** object that is provided as a parameter.

If you want to know when **AudioInput** instance has stopped you subscribe to **Stopped** event. It has no parameters and its sole purpose is to tell user when everything has been stopped.

### 18.4.3 Configuring

Set **ThreadInvoking** true if you want **AudioInput** instance to synchronize its events to the main UI thread automatically but make sure that AudioInput instance has a valid parent control.

**ThreadInvoking** is false by default so do not forget to call **Control.Invoke** method if you update GUI in your **DataGenerated** event handler.

If you want to use other Windows' recording device than the default one, set **RecordingDevice** property. You can get all available recording devices by using AudioInput's static method **GetRecordingDevices**.

Volume can be controlled through Volume property. Valid values are from 0 to 100 where 0 means mute and 100 maximum volume. You can set volume also when **AudioInput** instance is enabled (i.e. generating samples).

If you want to use difference sampling rate than the default (44100 Hz) then set **SamplesPerSecond** property. Setting this property while **AudioInput** instance is enabled has no effect.

To use mono audio instead of stereo (default), set **IsStereo** to false. Setting this property while **AudioInput** instance is enabled has no effect.

If you prefer 8 bits per sample rather than 16 (default), set **BitsPerSample** property 8. Valid values are 8 and 16 (default). This limitation comes from PCM wave format. Setting this property while **AudioInput** instance is enabled has no effect.

### 18.4.4 Starting

To start **AudioInput** instance you can either set **IsInputEnabled** property true or by calling **Start** method. When **DataGenerated** event provides you with a new set of audio samples you can e.g. render them using **LightningChartUltimate** instance.

## 18.4.5 Stopping

When you want to stop ***AudioInput*** instance, set ***IsInputEnabled*** false or call ***RequestStop*** method. ***RequestStop*** method does not stop instantly. It just signals ***AudioInput*** instance to stop as soon as it is possible. If you need to wait until everything has been stopped then wait for ***Stopped*** event to get triggered. You need to have subscribed to it.

## 18.5 Usage (WPF)

This chapter describes the usage of WPF version of ***AudioInput*** class. WPF version of ***AudioInput*** works basically the same way as WinForms version. However, there are a couple of things that a WPF user is good to know and these things are described in this chapter.

### 18.5.1 Creation

Create a new ***AudioInput*** instance either manually in code-behind or by dragging and dropping it from Visual Studio's toolbox on to your window, user control etc.

If you do not need to show GUI (i.e. you use your own or you control ***AudioInput*** object from your source code) then use ***AudioInput*** from ***Arction.WPF.SignalTools*** namespace. This particular class is derived from ***FrameworkElement*** and all its properties are bindable. For your convenience, after you have installed LightningChart Ultimate SDK, ***Arction.WPF.SignalTools.AudioInput*** can also be found from Visual Studio's toolbox so you can drop it on your windows, user control etc. and then move the element wherever you need it in your XAML code. Necessary XML namespace will be added automatically this way.

There is also a ready-made GUI for ***AudioInput***. It can be found from ***Arction.WPF.SignalTools.GUI*** namespace. You can find it also from Visual Studio's toolbox after you have installed LightningChart Ultimate SDK. Note that this is just a GUI for ***Arction.WPF.SignalTools.AudioInput*** class but it contains an instance of ***Arction.WPF.SignalTools.AudioInput*** class and you can access it through ***Input*** property. I.e. you do not have to create a new separate ***Arction.WPF.SignalTools.AudioInput*** instance.

It is recommended to set ***LicenseKey*** property so that your ***AudioInput*** instance uses an explicit license key instead of trying to find one from Windows' registry. Note, if you are using a trial version/license then you can leave ***LicenseKey*** property to its default value.

## 19. AudioOutput component

**AudioOutput** component allows user to convert System.Double signal data into an audio stream which is then played back through speakers, or sent to Line-out connector of sound device.

### 19.1 Properties

**Balance** – Gets or sets audio playback balance. Valid values are between -100 to 100. -100 means that audio is played only through the left speaker. 0 means that both speakers output audio. 100 means that audio is played only through the right speaker.

**BitsPerSample** – Gets or sets how many bits are allocated per sample. Supported values are 8 and 16. If other value is used, 16 is used instead. Is settable when IsOutputEnabled is false.

**IsOutputEnabled** – Gets or sets the state of this instance (i.e. starts or stops it). Setting this property true is the same as calling Start method where false is the same as calling Stop method.

**IsStereo** – Gets or sets whether to use two channels (stereo) or just one (mono). Is settable when IsOutputEnabled is false.

**LicenseKey** – Gets or sets license key string in normal or encrypted format.

**PlaybackDevice** – Gets or sets the current playback device. Is settable when IsOutputEnabled is false. By setting this property null, Windows' default playback device is used.

**SamplesPerSecond** – Gets or sets sampling frequency. Is settable when IsOutputEnabled is false.

**Volume** – Gets or sets volume (0-100). Is settable when IsOutputEnabled is false.

## 20. SpectrumCalculator component

*SpectrumCalculator* component allows conversion between time domain and frequency domain.

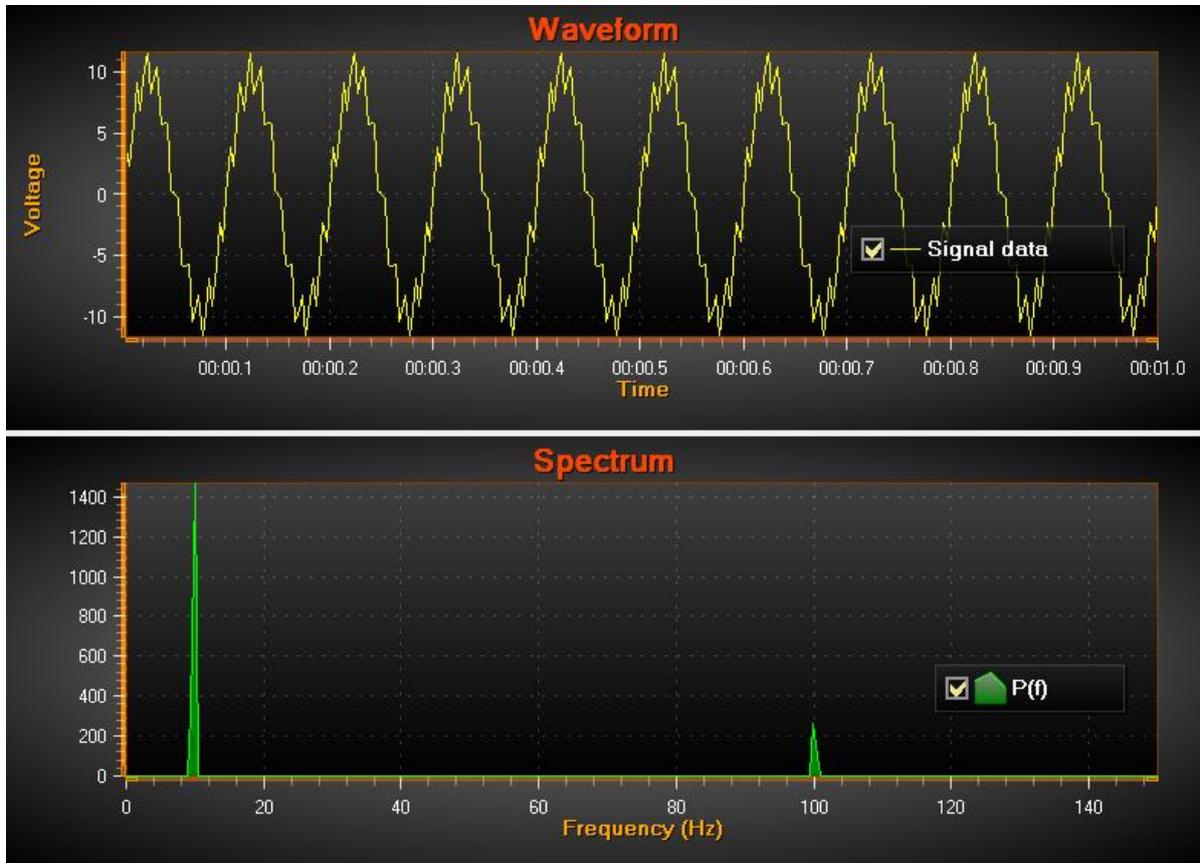


Figure 20-1. Example of source signal data (top) converted to frequency domain (bottom). Signal sampling frequency = 300 Hz, thus frequency scale is  $300/2 = 150$  Hz. The strong sine base line is 10 Hz (10 cycles / sec). Smaller signal of 100 Hz is added as noise. Both spikes are found in the power spectrum.

The following public methods are available:

- ***CalculateForward(double[] samples, out double[] fftData)***: Converts time domain signal data to frequency domain by using FFT. Output *fftData* contains also negative values. Input and output data arrays must be of equal length. The length is the resolution of the data, spreading from 0 Hz to sampling frequency / 2 with equal frequency interval between output values.
- ***CalculateForward(float[] samples, out float[] fftData)***: like the previous method, but for single accuracy floating point values.
- ***CalculateBackward(double[] fftData, out double[] samples)***: Converts frequency domain data to time domain. Makes signal samples from FFT data, sample count equals input *fftData* length.

- **CalculateBackward**(float[] fftData, out float[] samples): like the previous method, but for single accuracy floating point values.
- **PowerSpectrum**(double[] samples, out double[] fftData): Calculates power spectrum of signal data. Same as CalculateForward, but with absolute output values.
- **PowerSpectrum**(float[] samples, out float[] fftData): like the previous method, but for single accuracy floating point values.
- **PowerSpectrumOverlapped**(double[] samples, int fftWindowLength, double overlapPercent, out double[] fftData, out int processedSampleCount): calculates power spectrum by shifting the calculation windows inside source signal samples data, by overlap percent. Signal data must be longer than given FFT window length. The output FFT data is length of fftWindowLength, so not necessarily same than the length of source data. The output data is absolute values.
- **PowerSpectrumOverlapped**(float[] samples, int fftWindowLength, double overlapPercent, out float[] fftData, out int processedSampleCount)

## 21. Headless mode

Headless mode is a software capability of working on a device without an access to a Graphical User Interface (GUI). The term "headless" is also used when software does not require the presence of peripheral devices (like display, keyboard, mouse) and access to them. The absence of peripherals does not cause the failure of initialization or execution processes. However, in this case, the software may receive inputs and provide output via other communication interfaces – i.e., network or serial port.

### 21.1.1 Headless Rendering

Headless configuration allows running LightningChart in a headless/server environment. Expected scenarios include background rendering in software applications without User Interface (UI) and generation of a bitmap image from the chart content. The image then can be passed to the *headful* system for further rendering.

#### Basic usage

```
var chart = new LightningChartUltimate(new RenderingSettings()
{
    HeadlessMode = true
});
```

Headless mode can be activated by setting the flag to true. The property can be accessed via ***chart.ChartRenderOptions*** (for WPF) or ***chart.RenderOptions*** (for WinForms). LightningChart automatically detects its usage in the application of type Windows Service and there is no need to specify the mode.

#### 21.1.1.1 Additional initialization options

The initialized instance of LightningChart with a missing UI and visual parent will not receive any rendering requests, like layout size or when to render a frame. Furthermore, WPF chart uses these signals to initialize a rendering engine, when WinForms does engine initialization during the creation time. Thus, the following operations and configurations must be applied to the chart by the user:

- Define size using ***chart.Width*** and ***chart.Height*** properties.
- Request rendering engine initialization by calling ***chart.InitializeRenderingDevice(true)***. (Only for WPF)
- Subscribe to the ***chart.AfterRendering*** event for implementing the logic of exporting the images.

The chart still reacts to property changes. The rendering of a new frame can be queried by consecutive ***BeginUpdate()*** and ***EndUpdate()*** call, if it is needed.

### 21.1.1.2 Capturing images

The rendered frame can be exported (see Chapter 12) with these ways

- **OutputStream** property
- **SaveToStream** method
- **CopyToClipboard** method
- **CaptureToByArray** method
- **SaveToFile** method

In general, bitmap stream is preferred. ViewXY chart supports EMF, WMF, SVG in headless mode too, in **SaveToStream** and **SaveToFile** methods.

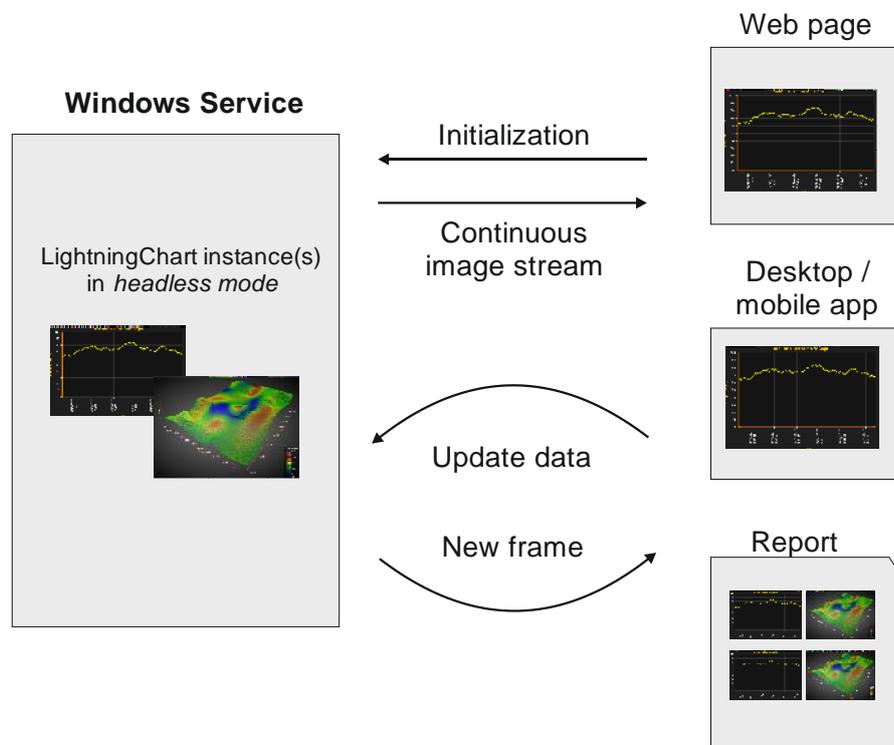


Figure 21-1. Diagram of example usages.

## 21.1.2 Limitations and Requirements

### 21.1.2.1 Threads

Headless configuration allows using LightningChart for a background work without placing it inside a visual parent and without access to the chart from the foreground thread (GUI thread). During the creation of the LightningChart instance, the properties of a chart must be updated within the same thread that has created the chart.

- The COM threading model, called “**apartment**”, must be STA (Single Thread Apartment), not MTA (Multi Thread Apartment). For an ordinary UI application, STA is the default model, whereas MTA state is default for Windows Services.
- All access, i.e. update, creation, disposing, must be made via that thread. UI must be touched only from GUI thread. Thus, if there are interaction operations required, they should be moved from chart’s thread to GUI thread. **Note!** LightningChart can be run on GUI thread.
- The thread must have valid and active message queue pump. For example, run *Application.Run* on the thread.

### 21.1.2.2 Chart Update

LightningChart uses a single buffer on rendering, thus exporting of a new image will be handled after the exporting of the previous image was finished. The synchronous configuration (*ChartUpdateType.Sync*) provides a rendering of an image straight after the receiving a request to update the properties of a chart. Sync mode should be enabled for the headless mode to perform faster and uninterruptedly.

### 21.1.2.3 Engine support

Both DirectX 9 & 11 engines work in a headless mode. However, only DirectX 11 can be used in the applications type of Windows Services due to the limitations of MS DirectX.

### 21.1.2.4 Licensing

By default, Windows Service executes in the security context of a system user account. **Installation of a trial and development license is impossible.** For this reason, the service application **must** contain a valid **Deployment Key** or be running with credentials of a normal user with an active license (trial / development).

### 21.1.3 Example solution

LightningChart SDK comes with an example Visual Studio solution (*DemoService.sln*) containing:

- Service
- Console application
- Client application for WPF

DemoService.sln can be found in *c:\program files (x86)\Arction\LightningChart Ultimate SDK v.8\DemoService* folder.

When starting up the WPF client, it shows a frame container in the middle of the window.

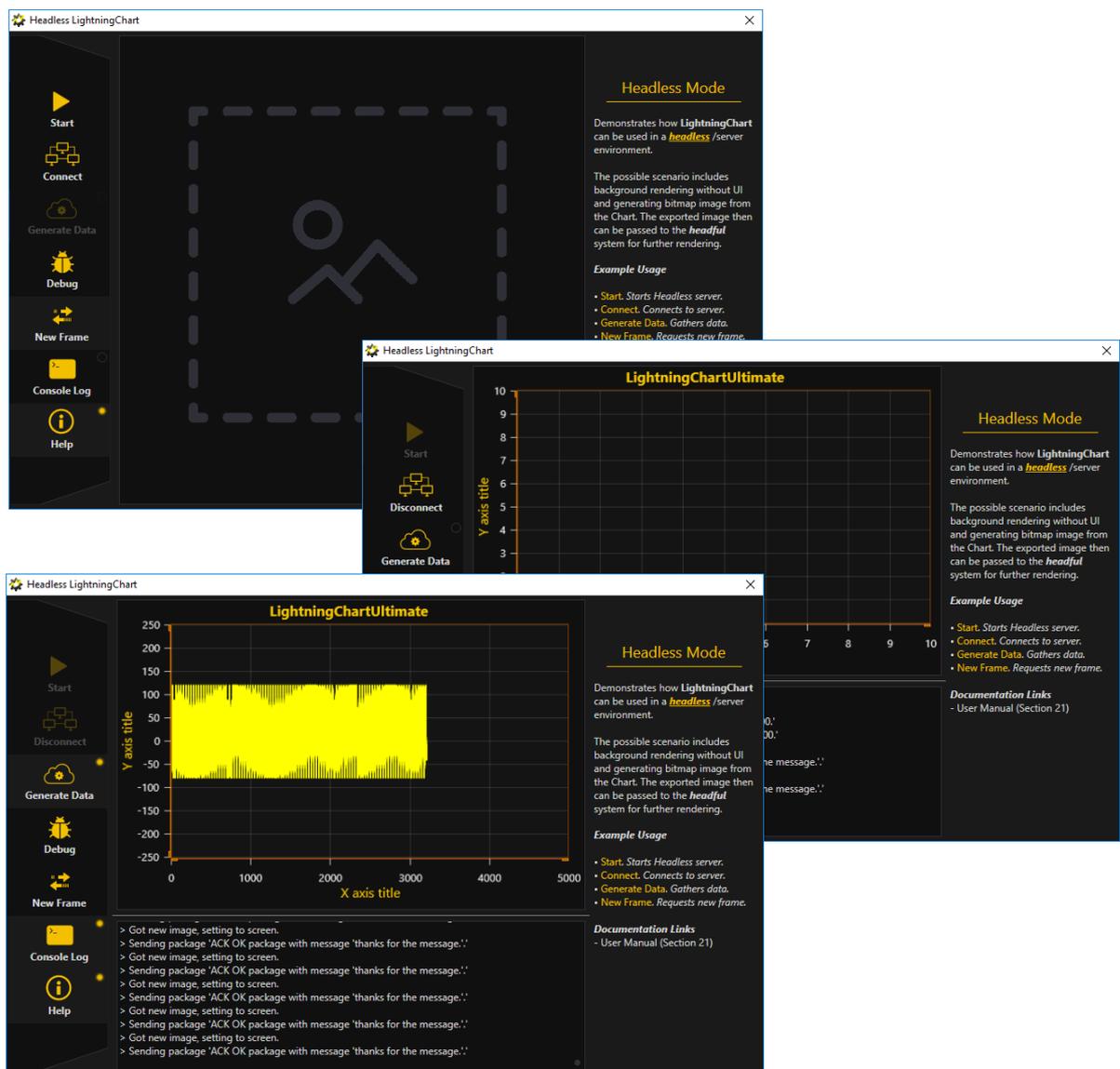


Figure 21-2. WPF client app. After Start, Connect and Generate Data, it shows continuously updating image stream.

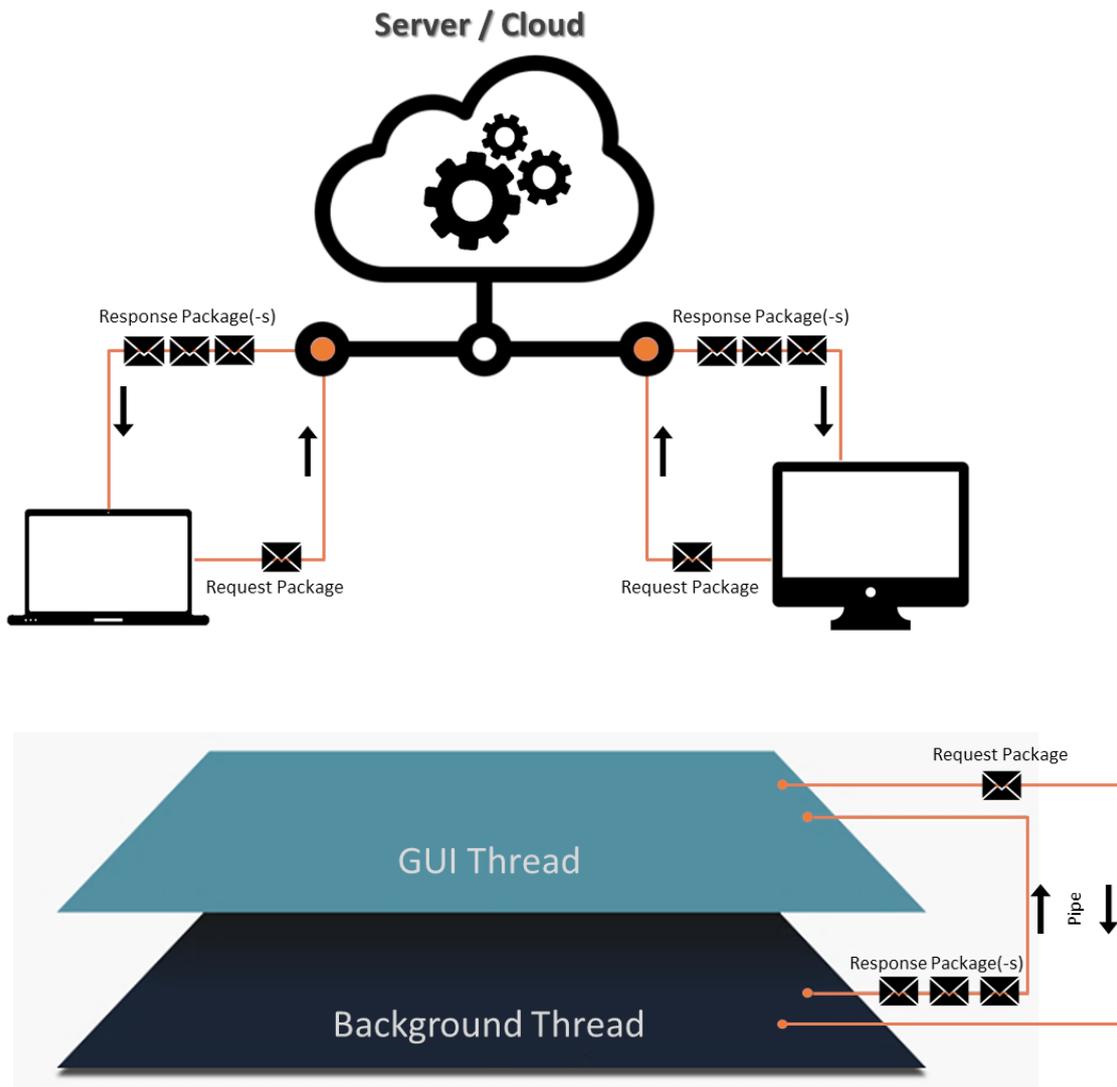


Figure 21-3. Headless demo service – client internal operation of messages with named pipes and background thread illustrated.

## 22. Using Windows Forms chart in WPF application

**Note:** LightningChart has 3 WPF API's available. Consider using WinForms chart API in WPF application only in special cases.

### 22.1 How about using Arction Windows Forms controls in WPF?

In WPF, you can use Windows Forms components by adding **Arction.WinForms.Charting.LightningChartUltimate.dll** and **Arction.WinForms.SignalProcessing.SignalTools.dll** as reference to your project, and creating them by code. **LightningChartUltimate** control and most of other controls have a built-in UI. Use **WindowsFormsHost** as parent container to these. These controls can be used also without UI, with their methods and properties.

### 22.2 Should I use Arction.WinForms.LightningChartUltimate in WPF?

Using WPF chart assemblies is recommended over WinForms chart in WPF applications, because it doesn't need the **WindowsFormsHost** control, and thus does not have the generic "airspace" problem of **WindowsFormsHost** control. Another advantage is that the WPF chart can have transparent background and the charts can be placed one over another.

Using **WindowsFormsHost** control with WinForms chart control can be considered to be used when absolutely maximum performance is required. **WindowsFormsHost** + *WinForms chart* rendering is slightly faster.

If you choose to use the WinForms chart in WPF application, it must be placed inside **WindowsFormsHost** control. Add a **WindowsFormsHost** control (found in the Visual Studio WPF Toolbox) into your WPF form.

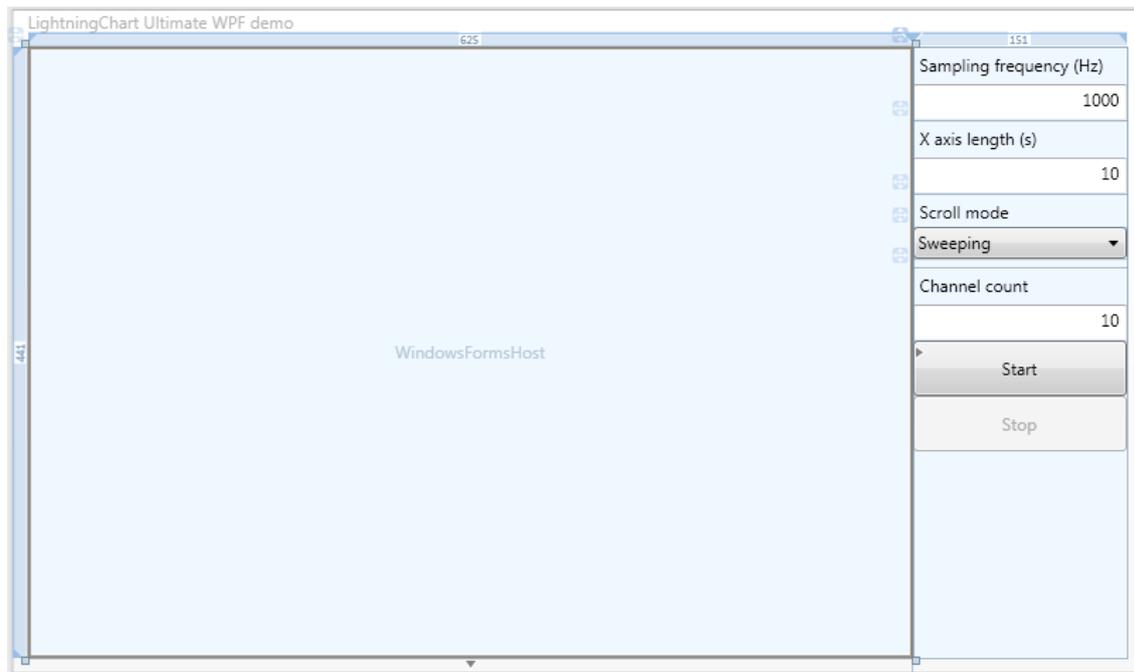


Figure 22-1. WPF application in designer. **WindowsFormsHost** control keeps the **LightningChartUltimate** object inside when the application is executed.

Create a **LightningChartUltimate** object and place it inside the **WinFormsHost** object in code. Open the form xaml.cs file and create the chart in the form constructor:

```
public WindowMain()
{
    InitializeComponent();

    CreateChart();
}

private LightningChartUltimate m_chart = null;

void CreateChart()
{
    m_chart = new LightningChartUltimate();

    //Set the chart object as child to the WindowsFormsHost control
    windowsFormsHost1.Child = m_chart;
}
}
```

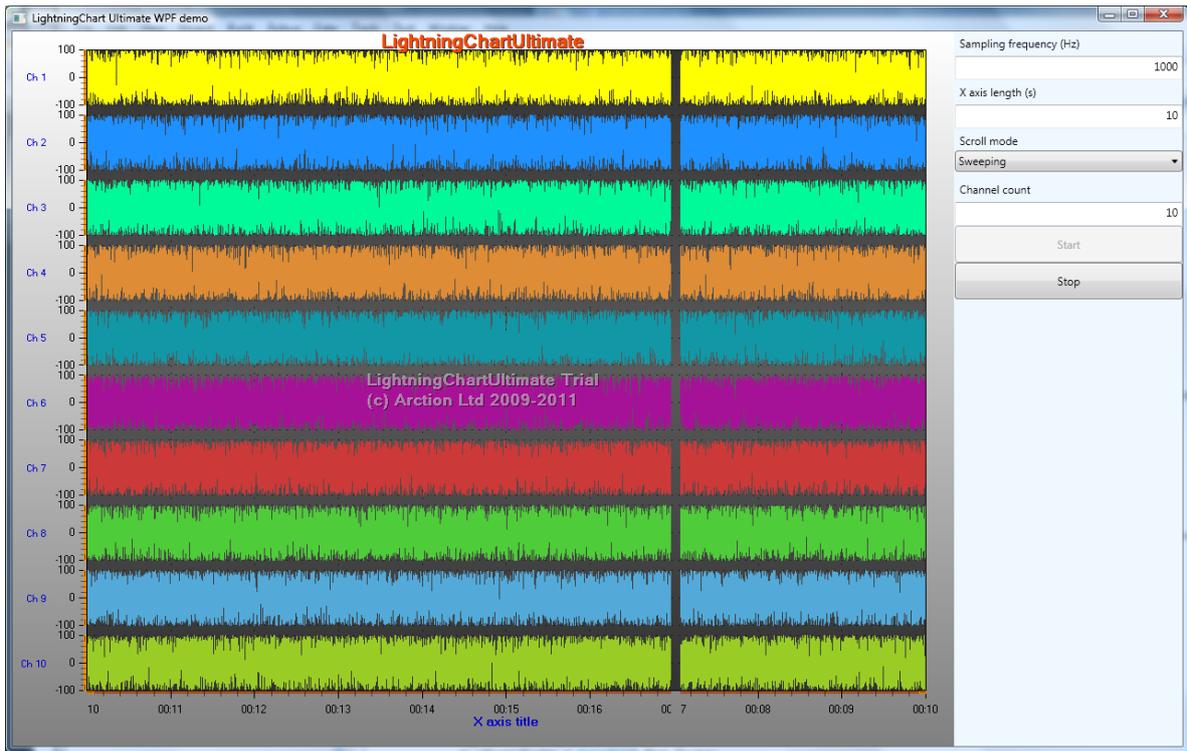


Figure 22-2. WinForms chart in WPF application.

## 23. Using LightningChart in C++ applications

LightningChart is a .NET library and it can be used with C# and VB.NET languages most fluently. But it can be used in C++ Win32 applications, including MFC applications, too. The application using LightningChart must be compiled with **Common Language Runtime Support (/clr)** option. If you want to create a Windows Form Project using C++, then please refer to the detailed step by step tutorial below.

### 23.1 Install required C++/CLR packages

Make sure your Visual Studio have installed C++ package with C++/CLR. For example, run Visual Studio (2017) Installer, and select update/modify button. From Individual components select **C++/CLI support**. From Workloads select **Desktop development with C++**.

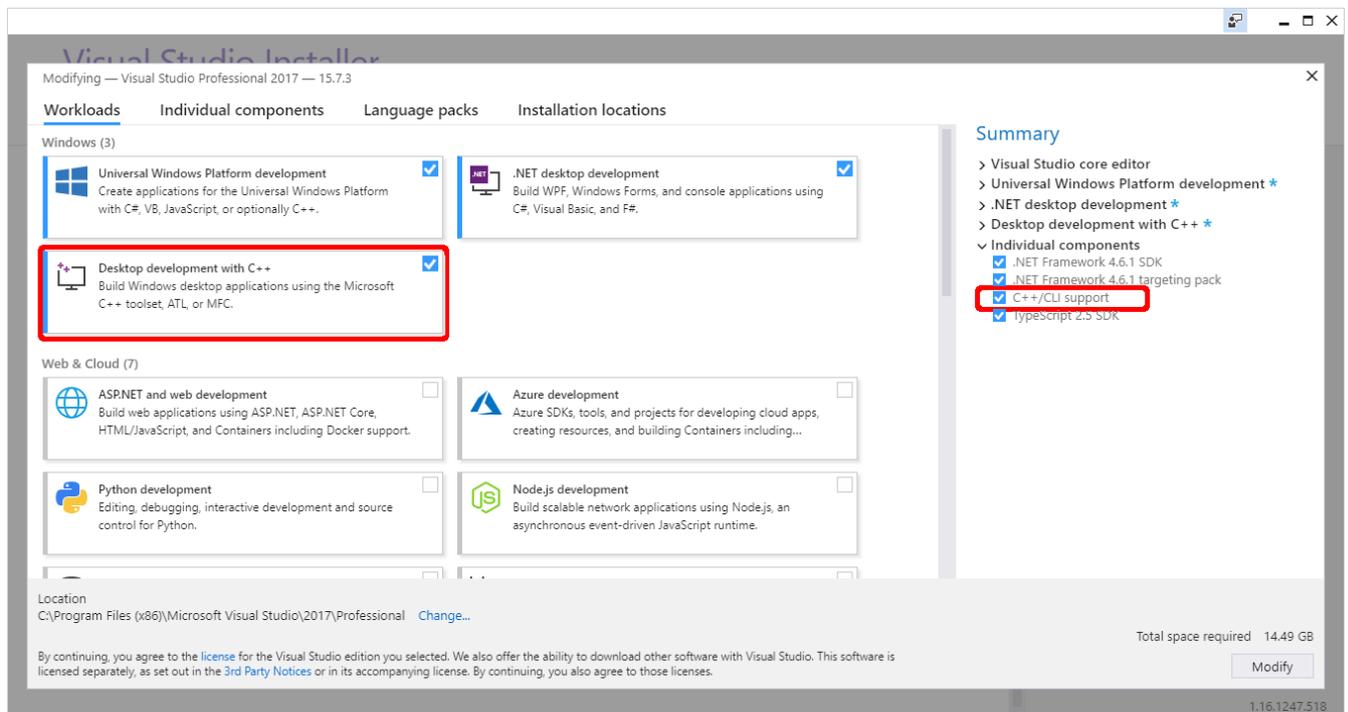


Figure 23-1. Visual Studio Installer selections for C++ project.

## 23.2 Setting Visual Studio project

Open Visual Studio 2017 and create a new project. If you have all required packages and components (as described above), then you should see the following selection when creating new project (**Templates -> Visual C++ -> CLR -> CLR Empty project**).

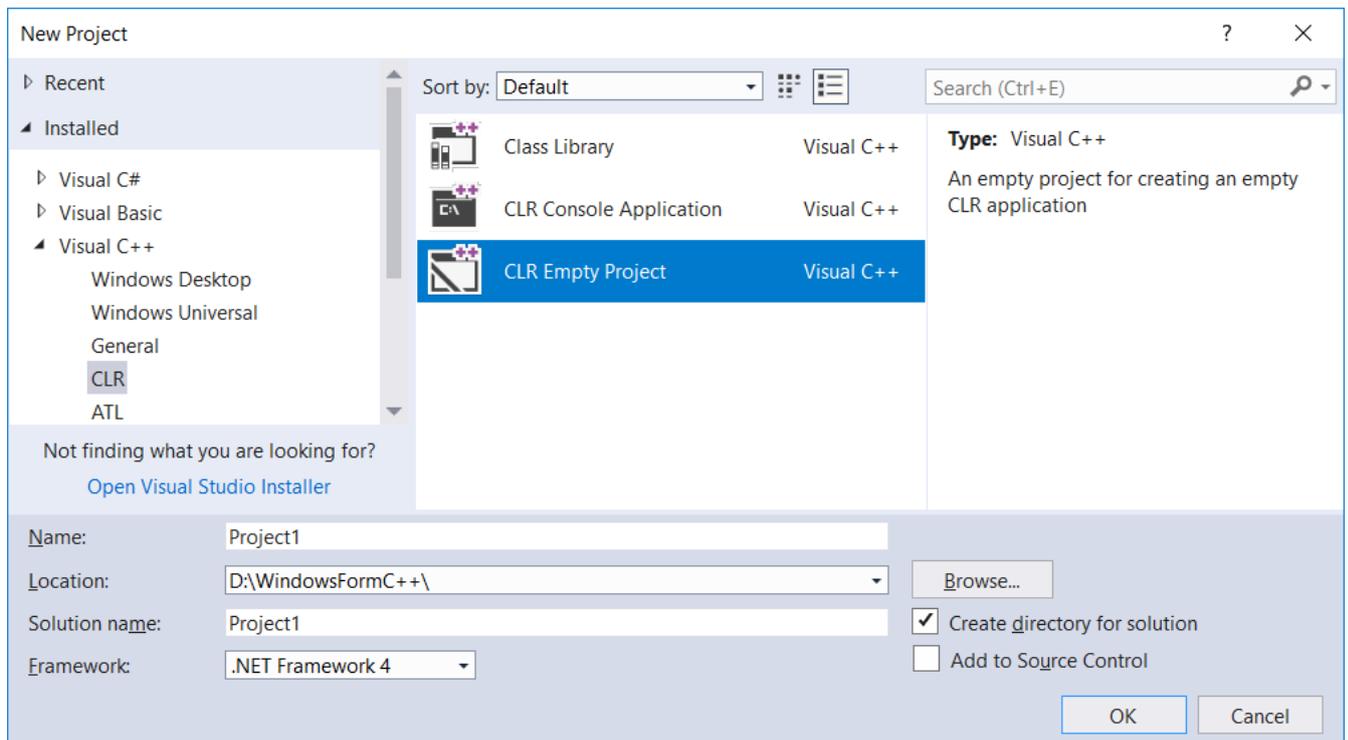


Figure 23-2. Windows Forms C++ project template.

Right click on the Project1 and choose Properties option. Modify **Configuration Properties -> Linker -> System -> SubSystem** and **Linker -> Advanced -> Entry point** as show in the figure.

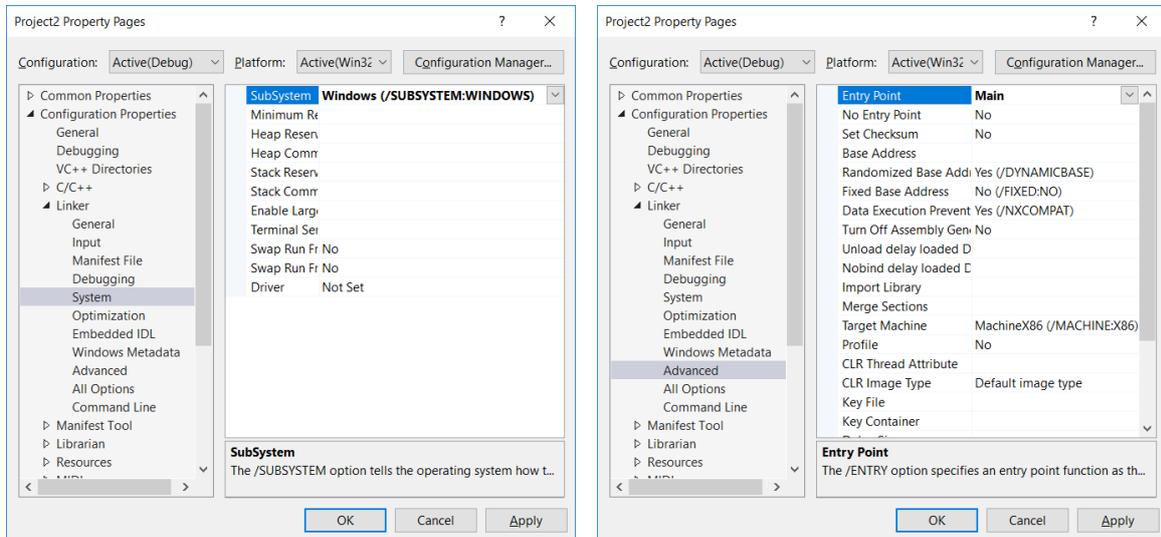


Figure 23-3. C++ project property pages.

Now add *Windows Form* Item in the project: right click on the Project1 and choose **Add -> New Item...** -> and select Windows Form as shown in the figure. Don't worry if you will see the error message **The data necessary to complete this operation is not yet available. (Exception from HRESULT: 0x8000000A)**. Just close it and proceed to the next step.

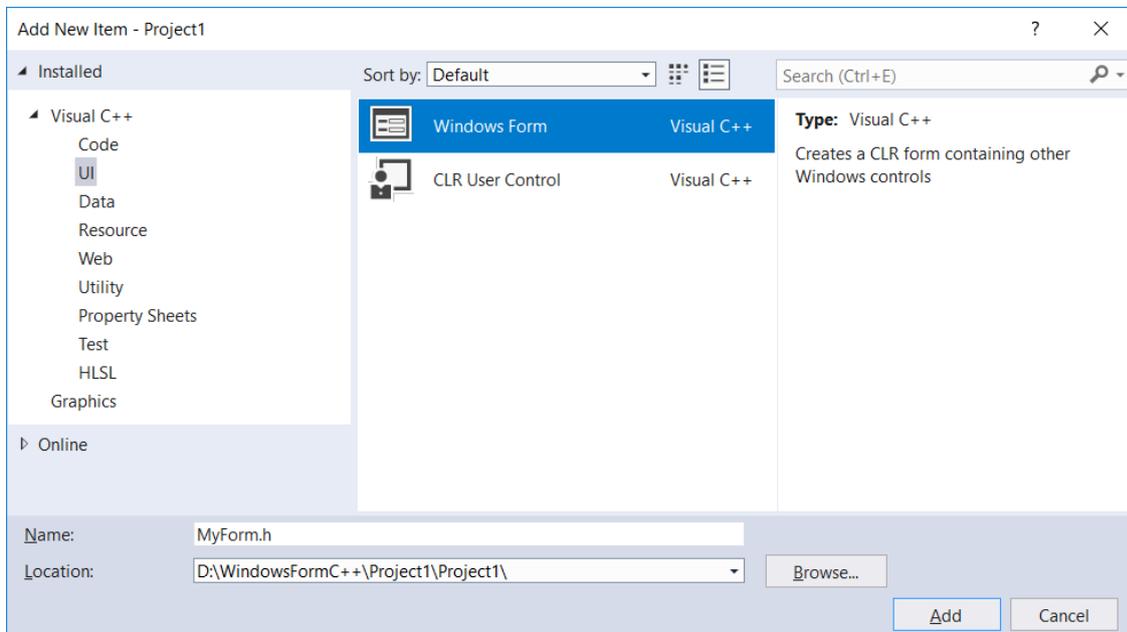


Figure 23-4. Add new Windows Forms item in C++ project.

Add the following code to the MyForm.cpp, then save it and close the Visual Studio 2017.

```
#include "MyForm.h"

using namespace System;
using namespace System::Windows::Forms;

[STAThreadAttribute]
void Main(array<String^>^ args) {
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);
    Project1::MyForm form;
    Application::Run(%form);
}
```

Project is ready to be built for the first time. Reopen your Project1 and select menu **Build -> Rebuild Solution**. Then you run project, you should see an empty *Windows Form*.

### 23.3 Creating LightningChart application in C++ project

Now you can add your components in the form by editing **MyForm.h** file. Below is simple example how to create chart with **PointLineSeries** on it. Include Arction's WinForms DLL in references list and add relevant namespaces in **MyForm.h** code file.

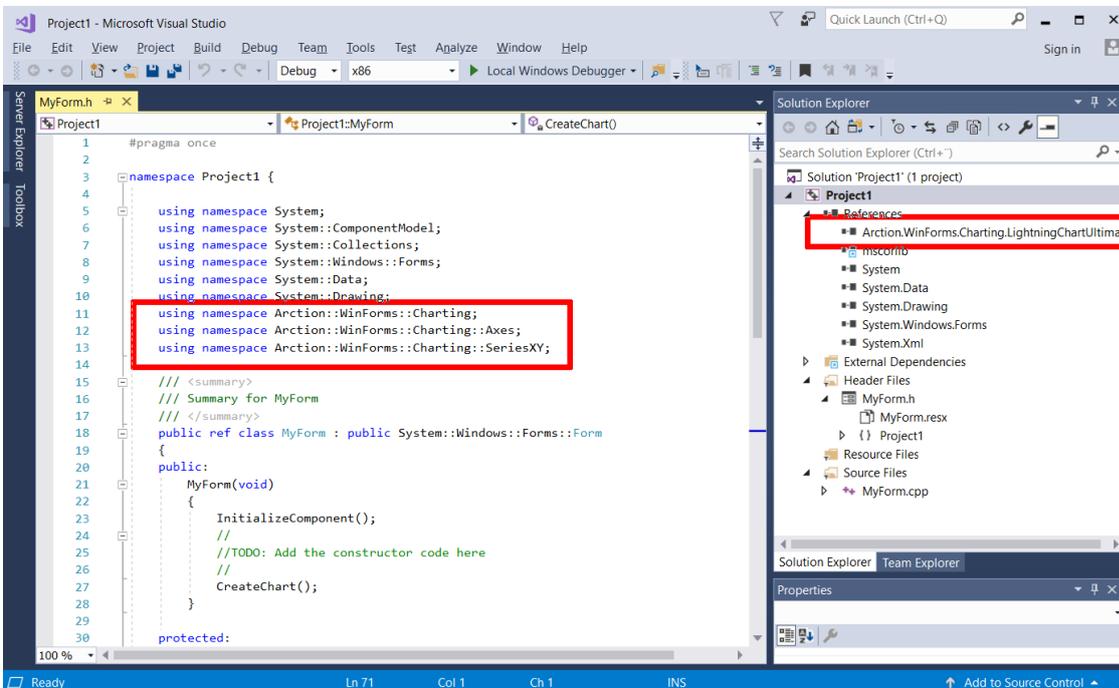


Figure 23-5. Include Arction.WinForms.Charting.LightningChartUltimate.dll in references list and add relevant namespaces in the project.

Declare a 'chart' variable and set its properties. Chart creation method can be like the following:

```
protected:
LightningChartUltimate ^ _chart;

void CreateChart()
{
    _chart = gcnew LightningChartUltimate();

    //Disable repaints for every property change
    _chart->BeginUpdate();

    //Set parent window by window handle
    _chart->Parent = this;

    //Fill the form area
    _chart->Dock = DockStyle::Fill;

    _chart->ActiveView = ActiveView::ViewXY;

    // Configure x-axis.
    AxisX^ axisX = _chart->ViewXY->XAxes[0];
    axisX->SetRange(0, 20);
    axisX->ScrollMode = XAxisScrollMode::None;
    axisX->ValueType = AxisValueType::Number;

    // Configure y-axis.
    AxisY^ axisY = _chart->ViewXY->YAxes[0];
    axisY->SetRange(0, 100);

    PointLineSeries^ pls1 = gcnew PointLineSeries(_chart->ViewXY, axisX, axisY);
    pls1->LineStyle->Color = Color::Yellow;
    pls1->Title->Text = "New Title";
    pls1->PointsVisible = true;
    pls1->LineVisible = true;
    _chart->ViewXY->PointLineSeries->Add(pls1);

    // Generate random data.
    Random rand;
    int pointCount = 21;

    array<SeriesPoint> ^ points = gcnew array<SeriesPoint>(pointCount);
    for (int point = 0; point < pointCount; point++)
    {
        points[point].X = (double)point;
        points[point].Y = 100.0 * rand.NextDouble();
    }

    pls1->Points = points;

    // Allow chart rendering.
    _chart->EndUpdate();
}
```

The resulting application of this example looks like the following, when compiled and executed:

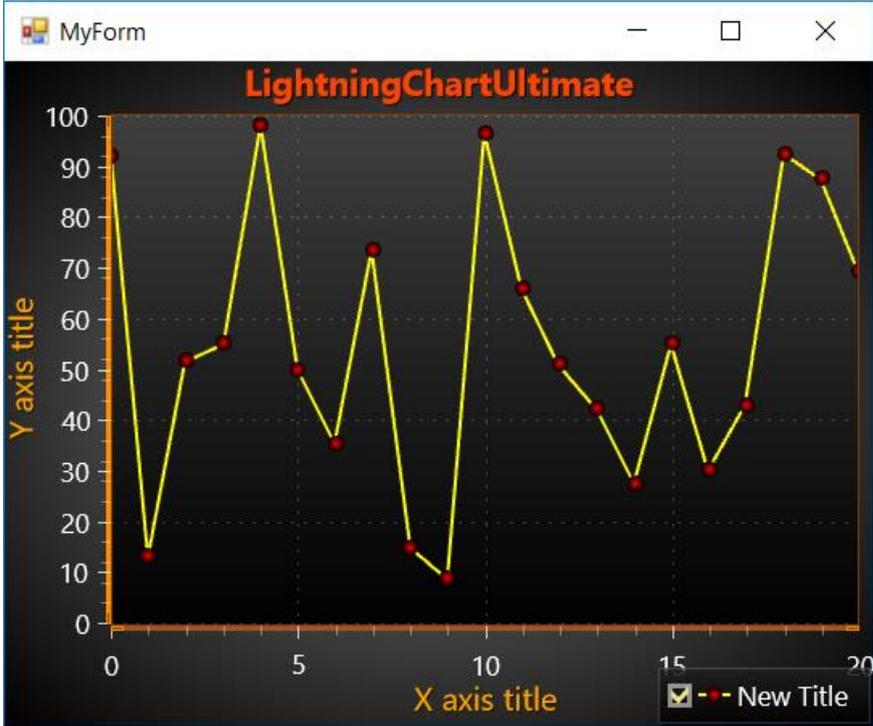


Figure 23-6. Example application executed.

## 24. Dispose pattern

### 24.1 Chart created in code

#### 24.1.1 Chart disposing

When you have created the chart in code, and don't need that chart anymore, call

```
chart.Dispose();
```

to free the memory of the chart and all its objects, such as series, markers, palette steps...

#### 24.1.2 Objects disposing

If you create objects on the fly, and want to free their memory before exiting the application or disposing the whole chart [with *chart.Dispose()*], remove the object from the collection it has been added to, and then call *Dispose()* for the object.

E.g. Disposing a series from *chart.ViewXY.PointLineSeries* collection:

```
//Do cleanup... Remove and dispose 3 series

_chart.BeginUpdate();

List<PointLineSeries> listSeriesToBeRemoved = new List<PointLineSeries>();
listSeriesToBeRemoved.Add(_chart.ViewXY.PointLineSeries[1]);
listSeriesToBeRemoved.Add(_chart.ViewXY.PointLineSeries[3]);
listSeriesToBeRemoved.Add(_chart.ViewXY.PointLineSeries[4]);

foreach (PointLineSeries pls in listSeriesToBeRemoved)
{
    _chart.ViewXY.PointLineSeries.Remove(pls);
    pls.Dispose();
}

_chart.EndUpdate();
```

When *LightningChart's* objects are not needed anymore it's good practice to *Dispose* them to prevent memory leaking.

## 25. Object model notes

### 25.1 Sharing objects between other objects

LightningChart object model is tree-based. Every class has its parent object, and list of child objects. With aid of this tree, child object notifies parent of its changes, so that parent can respond to it respectively, and parent notifies its parent chain until reaching the root node, LightningChartUltimate itself, so it knows how to refresh accordingly.

**Chart takes ownership of all the objects given to it, and it will dispose the objects when it does no longer need them.** This includes situations where a new replacing object is given to chart, and when the objects parent is disposed. User must be aware of this, as otherwise they might end up using disposed objects.

If object is shared between other .NET component and LightningChart, and LightningChart disposes the object, the other .NET component is left with a disposed object. LightningChart can't detect parent sharing between LightningChart and other components.

**Sharing objects between other objects in the same chart, or other chart instances, is forbidden.**

Example 1 of wrong usage:

```
AnnotationXY annotation1 = new Annotation();  
chart.ViewXY.Annotations.Add(annotation1);
```

```
AnnotationXY annotation2 = new Annotation();  
annotation2.Fill = annotation1.Fill;  
chart.ViewXY.Annotations.Add(annotation2);
```

It is wrong because same Fill object can't be shared between multiple objects.

Correct way: Only copy properties if they are of ValueType (e.g. Integer, Double, Color)

Example 2 of wrong usage:

```
SeriesEventMarker marker = new SeriesEventMarker();
```

```
chart.ViewXY.PointLineSeries[0].SeriesEventMarkers.Add(marker);
```

```
chart.ViewXY.PointLineSeries[1].SeriesEventMarkers.Add(marker);
```

Same object shouldn't be added to collection of multiple collections.

Correct way: Create own marker for both series.

**Remember to subscribe to ChartError event handler, it reports - in many cases - errors of invalid object sharing cases, see chapter 14.**

## 26. Deployment / distribution of LightningChart assemblies

### 26.1 Referenced assemblies

Deliver Arction dlls with your executable, next to your executable, Global assembly cache, or other folder where .NET assembly resolving system can find them. LightningChart supports **ClickOnce** deployment too.

#### WinForms:

- Arction.WinForms.Charting.LightningChartUltimate.dll
- Arction.Licensing.dll
- Arction.DirectX.dll
- Arction.RenderingDefinitions.dll
- Arction.RenderingEngine.dll
- Arction.RenderingEngine9.dll
- Arction.RenderingEngine11.dll
- Arction.DirectXInit.dll
- Arction.DirectXFiles.dll

and if using SignalTools

- Arction.WinForms.SignalProcessing.SignalTools.dll
- Arction.MathCore.dll

#### WPF:

- Arction.Wpf.Charting.LightningChartUltimate.dll **(if using Non-bindable WPF chart)**
- Arction.Wpf.SemibindableCharting.LightningChartUltimate.dll **(if using semi-bindable WPF chart)**
- Arction.Wpf.BindableCharting.LightningChartUltimate.dll **(if using fully bindable WPF chart)**
- Arction.Licensing.dll
- Arction.DirectX.dll
- Arction.RenderingDefinitions.dll
- Arction.RenderingEngine.dll
- Arction.RenderingEngine9.dll
- Arction.RenderingEngine11.dll
- Arction.DirectXInit.dll
- Arction.DirectXFiles.dll

and if using SignalTools

- Arction.Wpf.SignalProcessing.SignalTools.dll
- Arction.MathCore.dll

## 26.2 License key

Remember to assign use static **SetDeploymentKey** method for all components. Otherwise the chart enters in trial mode and works only 30 days, with trial nag on it. *For making a DeploymentKey and detailed license keys management, see chapter 3.*

## 26.3 Obfuscating your application code

It is **mandatory** to obfuscate your application code, so that LightningChart license keys are not visible to .NET disassembler tools. Leaking your license keys may lead into license termination, legal actions and claim of damage.

## 26.4 Obfuscating LightningChart code

If you are a LightningChart source code subscriber, you get access to source code of LightningChart libraries. It is **mandatory** to obfuscate the assemblies you build from LightningChart source code, to prevent Arction's intellectual property rights and code leaking. Distributing unobfuscated LightningChart libraries are a violation of EULA, and may lead into license termination, legal actions and claim of damage.

## 26.5 XML files of Arction assemblies

XML files

- Arction.WinForms.Charting.LightningChartUltimate.xml
- Arction.Wpf.BindableCharting.LightningChartUltimate.xml
- Arction.Wpf.Charting.LightningChartUltimate.xml
- Arction.Wpf.SemibindableCharting.LightningChartUltimate.xml
- Arction.Wpf.SignalProcessing.SignalTools.xml
- Arction.WinForms.SignalProcessing.SignalTools.xml

**deployment is forbidden.**

The files provided by Arction only for helping with the application development. They are used mainly to show code parameters and property tips. If you rebuild LightningChart assemblies from source code, ensure you are not deploying XML files mentioned above. **Distributing them is strictly forbidden**, as they will reveal too much info for .NET disassembler and reverse-engineering applications.

## 27. Troubleshooting

### 27.1 Updating from older version

LightningChart components API may have changed from older version you are using, and your project may not load or use the new version automatically. Follow this section for instructions on how to set the new version assemblies as a reference to your project and fix the properties that were unable to de-serialize in the Visual Studio form editor.

In order to update chart, you need to first remove the reference to old version and add reference to new version. In some cases you may need to fix \*.Designer.cs and \*.resx files as they may contain properties, which are binary incompatible.

#### Removing old reference from project References

1. Go to Solution Explorer.
2. Open References folder.
3. Select Arction assemblies and remove them by pressing **Delete** button or right-click and select **Remove**.

#### Adding reference to other version

1. Go to Solution Explorer.
2. Open References folder.
3. Add reference to new chart. Right-click on References folder. Select **Add Reference...** and select new Arction DLL file.

As the API may have changed, you may need to fix source code on changed properties.

If chart is totally incompatible (i.e. Visual Studio can't load UI on form editor), you need to remove LightningChart property setters from \*.Designer.cs and \*.resx files.

#### Removing property setters from \*.Designer.cs file

1. Open \*.Designer.cs file on text editor (use other editor than Visual Studio if possible).
2. Locate and delete rows containing setters for LightningChartUltimate. E.g.  
this.m\_chart.Background =  
(Arction.LightningChartUltimate.Fill)(resources.GetObject("m\_chart.Background"));

There is no need to remove inherited properties, like Location and Size. Remove properties, which are read from resource by a method like above "NN = ((...)(resources.GetObject("...")));".

#### Removing serialized items from \*.resx file

1. Open \*.resx file on text editor.

2. Find xml tags containing Arction objects (they are identified with chart member name. e.g. "m\_chart" or "lightningChartUltimate1").
3. Remove the lines including chart <data> tag to the end of xml object (</data> tag).  
E.g. chart Background is serialized as following xml object and you should remove all the following lines from the \*.resx file:

```
<data name="m_chart.Background" mimetype="application/x-  
microsoft.net.object.binary.base64">  
<value>  
    AAEAAAD/////AQAAAAAAAAAMAgAAAGRBCmN0aW9uLkxpZ2h0bmluZ0NoYXJ0VWx0aW1hd  
    GUsIFZlcnNp  
    b249NC42LjEuMjAwMSwgQ3VsdHVyZT1uZXV0cmFsLCBQdWJsaWNlZX1Ub2t1bj03MmY1N  
    WZiZDY5MDFm  
    ... lots of encoded stuff ...  
    YXlvdXQBAAAAB3ZhbHVlX18ACAIAAAAAAAAAACw==  
</value>  
</data>
```

Note that some objects may be very large, e.g. for Title row count may be like 200 lines and views are usually much larger (e.g. View3D about 2000 lines).

If you have several charts, all their serialized properties need to be removed. Editor search is handy tool to find the chart objects.

After you removed the objects from \*.resx file and related setters from \*.Designer.cs file, you should be able to open the project successfully in Visual Studio form editor.

## 27.2 Web support

See [www.arction.com/support](http://www.arction.com/support) for support options.

Discussion forums are available at <http://www.arction.com/forum>

## 27.3 Running in Virtual Machine platforms

LightningChart comes with DirectX10/11 WARP rendering for systems that don't give access to graphics hardware. WARP rendering takes place in CPU, performance reduction is to be expected when compared to hardware rendering. This needs an operating system with support for DirectX11.

For systems that don't support DirectX11, Lightning falls back to DirectX9 Reference Rasterizer mode. Performance is very poor, only small fraction of WARP's performance.

For automatic fallback to WARP and DirectX9, keep the RenderDevice set to **Auto**, **AutoPreferD9** or **AutoPreferD11**. See section 4.6.

## 28. Credits

### 28.1 Intel Math Kernel library

LightningChart Ultimate SDK uses Intel Math Kernel Library in some parts, like Fast Fourier Transform methods. Arction assemblies contain some native DLL files built from this library. Arction Ltd is licensed to use Intel Math Kernel Library.

### 28.2 Open-source projects

We present thanks to the following open-source projects and material providers:

#### **DirectX library for .NET**

LightningChart uses SharpDX-derived DirectX .NET DLLs with Arction-made extensions, <http://www.sharpx.org/>

### **Map sources**

LightningChart Ultimate maps have been imported from the map providers as follows:

*World, North America, Europe:* Natural Earth, <http://www.naturalearthdata.com/>  
*Australia:* Australian Bureau of Statistics, <http://www.abs.gov.au/>  
*Roads of USA:* National Atlas of the United States, <http://www.nationalatlas.gov>

### **Scalable Vector Graphics output**

LightningChart SVG export is using partially SvgNet project code by RiskCare Ltd.

### **Polynomial regression**

Polynomial regression calculation code is partially based on Math.Net library,  
<http://www.mathdotnet.com/>

The modified source code parts are available per request free-of-charge, from Arction Support ([support@arction.com](mailto:support@arction.com)).

For copyrights notices of open-source projects, see **Readme for LightningChartUltimate.txt** in LightningChart SDK install folder.