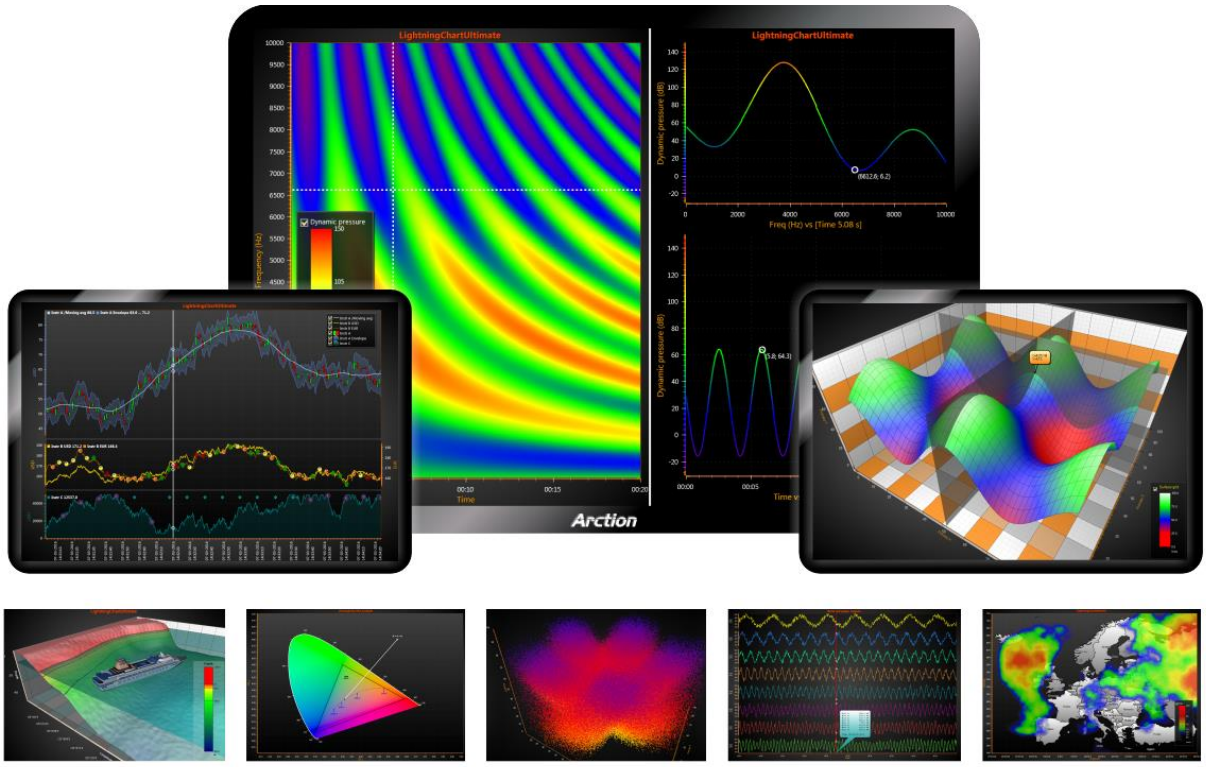


LightningChart Ultimate SDK

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) if you have any question!

Applies to LightningChart Ultimate edition, v.7.2

www.arction.com

www.lightningchart.com

1. Contents

2. Overview 14
2.1 Chart editions 14
2.2 Components 15
2.3 Namespaces 16
3. Installation 17
3.1 Before installing 17
3.2 Run the setup wizard 17
3.3 Adding Arction components manually to Visual Studio Toolbox 17
3.4 Configuring Visual Studio 2010-2015 help manually 18
3.4.1 Visual Studio 2010 18
3.4.2 Visual Studio 2012-2015 19
3.5 Code parameters and tips by Visual Studio IntelliSense 19
3.6 Selecting target framework 19
3.7 Updating from older version 21
4. License management 23
4.1 Licenses and adding LightningChart or other control to your application 23
4.1.1 Extracting license keys from LicenseManager 23
4.1.2 License storage when adding an Arction component from Toolbox into a form 24
4.1.3 License storage when creating a control by code 24
4.2 Trial period 25
5. LightningChartUltimate component 26
5.1 Adding from toolbox into Windows Forms project 26
5.1.1 Properties 26
5.1.2 Event handlers 27
5.1.3 Best practices, version updates in mind 27
5.2 Adding from toolbox into WPF project 27
5.2.1 Properties 27
5.2.2 Event handlers 28
5.3 Adding into Blend WPF project 28
5.3.1 Best practices, version updates in mind 29
5.3.2 Preventing blurring of the chart 29
5.4 Using Windows Forms chart in WPF application 30
How about using Arction Windows Forms controls in WPF? 30
Should I use Arction.WinForms.LightningChartUltimate in WPF? 30
5.5 Object model 33

5.5.1 Differences between Windows forms and WPF 34
5.6 Configuring appearance / performance settings 35
6. ViewXY 37
6.1 Axis layout options 39
6.1.1 Automatic margins 40
6.1.2 Setting how axes are placed 40
X axis automatic placement 40
Y axis automatic placement 42
6.1.3 Graph segments and Y axes placement in them 44
Layered 44
Stacked 44
Segmented 45
6.1.4 Axis grid strips 46
6.1.5 Other AxisLayout options 47
6.2 Y axes 48
6.2.1 AxisY class properties 48
6.2.2 Tick value labels formatting 48
6.2.3 Value type 49
6.2.4 Range setting 50
6.2.5 Restoring range 50
6.2.6 Divisions 50
6.2.7 Grid 50
6.2.8 Custom ticks 51
6.2.9 Reversed X and Y axis 52
6.2.10 Logarithmic axes 52
Exponential presentation for 10 base 52
Natural logarithm 53
6.2.11 Converting between axis values and screen coordinates 53
6.2.12 MiniScale 54
6.3 X axis 54
6.3.1 Real-time monitoring scrolling 54
None 54
Stepping 55
Scrolling 55

Sweeping 57
Triggering 57
6.4 ViewXY series, general 59
6.5 Point line series 59
6.5.1 Line style 60
6.5.2 Points style 60
6.5.3 Coloring points individually 60
6.5.4 Adding points 61
6.5.5 Adding points, alternative way 61
6.6 Sample data series 61
6.6.1 Y precision 62
6.6.2 Adding points 63
6.7 Freeform point line series 63
6.8 Advanced line coloring of line series 65
6.8.1 Y-value based coloring of line and fill with value-range palette 65
6.8.2 Custom shaping and coloring with CustomLinePointColoringAndShaping event 66
6.9 High-low series 66
6.9.1 Fill, line and point styles 67
6.9.2 Limits 68
6.9.3 Coloring by value-range palette 69
6.9.4 Adding data 69
6.10 Area series 70
6.10.1 Adding data 71
6.11 Bars 72
6.12 Stock series 74
6.12.1 Setting data to StockSeries 75
6.12.2 Setting X axis to date display 75
6.12.3 Custom formatting of appearance 76
6.13 PolygonSeries 77
6.13.1 Setting data to a Polygon 77
6.14 LineCollections 78
6.14.1 Setting data to a LineCollection 79
6.15 IntensityGridSeries 80
6.15.1 Setting intensity grid data 82

6.15.2 Creating intensity grid data from bitmap file 83
6.15.3 Fill styles 84
6.15.4 Rendering as pixel map 84
6.15.5 ValueRangePalette 85
6.15.6 Wireframe 86
6.15.7 Contour lines 87
6.15.8 Contour line labels 88
6.16 IntensityMeshSeries 89
6.16.1 Setting intensity mesh data, when geometry changes 90
6.16.2 Setting intensity mesh data, when geometry does not change 91
Creating the series and geometry 91
Updating the values periodically 91
6.17 Bands 92
6.18 Constant lines 92
6.19 Annotations 93
6.19.1 Controlling target and location 94
6.19.2 Using mouse to move, rotate and resize 95
6.19.3 Adjusting appearance 95
6.19.4 Size settings 96
6.19.5 Keeping text area visible 96
6.19.6 Clipping inside graph 96
6.19.7 Controlling the Z order 96
6.19.8 LayerGrouping performance optimization 97
6.19.9 Converting between axis values and screen coordinates 97
6.20 Legend box 98
6.21 Zooming and panning 100
6.21.1 Zooming with touch screen 101
6.21.2 Panning with touch screen 101
6.21.3 Left mouse button action 101
6.21.4 Right mouse button action 101
6.21.5 RightToLeftZoomAction 101
6.21.6 Zooming with mouse button 102
Zoom in/out by clicking 102
Zoom in with rectangle 103

Configuring zoom out rectangle 103
6.21.7 Zooming with mouse wheel 103
6.21.8 Zooming and panning with mouse wheel, over axis 103
6.21.9 Panning with mouse button 104
6.21.10 Enabling/disabling Ctrl, Shift and Alt 104
6.21.11 Zoom in/out with code 104
6.21.12 Zooming an axis by code 104
6.21.13 Rectangle zooming about a configurable origin 104
6.21.14 Linking Y axes zoom with same units 105
6.21.15 Automatic Y fit 106
6.21.16 Aspect ratio 106
6.21.17 Excluding specific X or Y axes from zooming and panning operations 107
6.22 DataBreaking by NaN or other value 107
6.23 ClipAreas 109
6.24 Maps 111
6.25 Vector maps 112
6.25.1 Selecting active map 112
6.25.2 Aspect ratio 113
6.25.3 Layers and their appearance settings 114
Setting individual fill and border style for each layer item 115
6.25.4 Mouse interactivity 116
6.25.5 Background photos 117
6.25.6 Combining other series with maps 118
6.25.7 Importing maps from ESRI shape file data 120
6.25.7.1 Programming interface for importing shp data 120
6.25.7.2 Dialogs 121
6.25.7.2.1 Shapefile Selection Dialog 121
6.25.7.2.2 Select Record Encoding and Invalid Name Fields 122
6.25.7.2.3 Layer data selection dialog 123
6.25.7.2.4 Item filter 125
6.25.8 Importing and replacing map layers 125
6.26 Tile maps 127
6.26.1 HERE 128
6.27 Line series cursors 129

6.27.1 Solving the data values in the position of LineSeriesCursor 130
Accurate method, solving Y value by X value, by using the data points array 131
Coarse method, solving Y screen coordinate by X coordinate, by using the data points array 131
6.28 Event markers 132
6.28.1 Chart event markers 133
6.28.2 Line series event markers 133
6.29 Persistent series rendering layers 134
6.29.1 Creating the layer 135
6.29.2 Clearing the layer 135
6.29.3 Adjusting layer alpha 136
6.29.4 Rendering data into the layer 136
6.29.5 Disposing the layer 137
6.29.6 Anti-aliasing data in the layer 137
6.29.7 Getting list of layers 137
6.29.8 Some layer limitations you should be aware of 137
6.30 Persistent series rendering intensity layers 138
6.30.1 Creating the layer 139
6.30.2 Clearing the layer 139
6.30.3 Changing palette colors 139
6.30.4 Adjusting the intensity effect of new trace and decay of old traces 139
6.30.5 Rendering data into the layer 140
6.30.6 Disposing the layer 140
6.30.7 Anti-aliasing data in the layer 140
6.30.8 Getting list of layers 140
7. View3D 141
7.1 3D model and dimensions 142
7.1.1 World coordinates 142
7.2 Walls 143
7.3 Camera 144
7.3.1 Predefined cameras 145
7.4 Lights 145
7.4.1 Directional light 146
7.4.2 Point of light 146
7.4.3 Lights and materials 146

7.4.4 Predefined lighting schemes 147
7.5 Axes 147
7.5.1 Location 148
7.5.2 Orientation 149
7.5.3 CornerAlignment 149
7.6 3D series, general 150
7.7 PointLineSeries3D 150
7.7.1 Point styles 151
7.7.2 Line styles 152
7.7.3 Adding points 152
Points 153
PointsCompact 153
7.7.4 Coloring points individually 154
7.7.5 Setting points sizes individually 155
7.7.6 Multi-coloring line 155
7.7.7 Displaying millions of scatter points 156
7.8 SurfaceGridSeries3D 157
7.8.1 Setting surface grid data 158
7.8.2 Creating surface from bitmap file 159
7.8.3 Fill styles 160
7.8.4 Contour palette 161
7.8.5 Wireframe mesh 163
Some notes when using wireframe simultaneously with fill 165
7.8.6 Contour lines 166
7.8.7 Scrolling surface data 168
7.9 SurfaceMeshSeries3D 170
7.9.1 Setting surface mesh data 171
7.9.2 Visualizing point clouds in 3D 172
7.10 WaterfallSeries3D 173
7.11 BarSeries3D 174
7.11.1 Bars grouping 174
7.11.2 Bar styles 177
7.11.3 Setting bar series data 179
7.11.4 Showing bars horizontally 179

7.12 MeshModels 181
7.13 Rectangle3D objects 182
7.14 Polygon3D objects 184
7.15 Zooming, panning and rotating 186
7.15.1 Mouse wheel zooming 186
7.15.2 Box zooming 186
7.15.3 Rotating and panning 187
7.15.4 Zooming with touch screen 188
7.15.5 Panning with touch screen 188
7.15.6 Using mouse wheel over an axis 188
7.15.7 Zooming, rotating and panning by code 188
7.16 Clipping objects within axis ranges 188
7.17 Annotations 189
8. ViewPie3D 190
8.1 Properties 191
8.2 Pie slices 191
8.3 Setting data by code 192
8.4 Viewing pie chart in 2D 193
9. ViewPolar 194
9.1 Axes 195
9.2 PointLineSeries 197
9.2.1 Setting data 197
9.2.2 Palette coloring 198
9.3 AreaSeries 199
9.3.1 Setting data 199
9.4 Sectors 200
9.5 Annotations 200
9.6 Markers 201
9.7 Zooming and panning 202
10. ViewSmith 203
10.1 Axis 203
10.2 PointLineSeries 208
10.3 Setting data 209
10.4 Annotations 210
10.5 Markers 210
11. Setting color theme 212
12. Scrollbars 212

13. Export and printing 214
13.1.1 Bitmap image export 214
13.1.2 Vector image export 214
13.1.3 Copy to clipboard 214
13.1.4 Printing 214
14. LightningChart performance 216
14.1 Selecting the correct API edition 216
14.2 Set the rendering options correctly 216
14.3 Updating chart data or properties 216
14.4 Line series tips 217
14.5 Intensity series tips 217
14.6 3D surface series tips 218
14.7 Maps tips 218
14.8 Hardware 218
15. LightningChart error and exception handling 219
16. ChartManager component 220
16.1 Chart interoperation, drag-drop 220
16.2 Memory management enhancement 220
17. SignalGenerator component 221
17.1 Sampling frequency, Output interval and Factor 221
17.2 Sine waveforms 222
223
17.3 Square waveforms 223
17.4 Triangle waveforms 224
17.5 Noise waveforms 225
17.6 Frequency sweeps 225
17.7 Amplitude sweeps 226
17.8 Starting and stopping 226
17.9 Multi-channel generator with master-slave configuration 226
17.10 Output data stream 227
18. SignalReader component 228
18.1 Key properties 228
18.2 Opening file quickly for playback 228
19. AudioInput component 230
19.1 Properties 230
19.2 Methods 230
19.3 Events 231
19.4 Usage (WinForms) 231

19.4.1 Creation 231
19.4.2 Event handling 231
19.4.3 Configuring 232
19.4.4 Starting 232
19.4.5 Stopping 233
19.5 Usage (WPF) 233
19.5.1 Creation 233
20. AudioOutput component 234
20.1 Properties 234
21. SpectrumCalculator component 235
22. Using LightningChart in C++ applications 237
23. Dispose pattern 243
23.1 Chart created in code 243
23.1.1 Chart disposing 243
23.1.2 Objects disposing 243
24. Object model notes 244
24.1 Sharing objects between other objects 244
25. Deployment 245
25.1 Referenced assemblies 245
25.2 License key 246
26. Troubleshooting 246
26.1 Web support 246
26.2 Running in Virtual Machine platforms 246
26.3 Credits 247
26.3.1 Intel Math Kernel library 247
26.3.2 Open-source projects 247

2. 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.

2.1 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	Performance
WPF (non-bindable)	No	No	Excellent
WPF (semi-bindable)	Yes	No	Very good
WPF (bindable)	Yes	Yes	Good
WinForms	No	No	Best

Table 1-1. Bindability and performance matrix.

- 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 binable 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.

2.2 Components

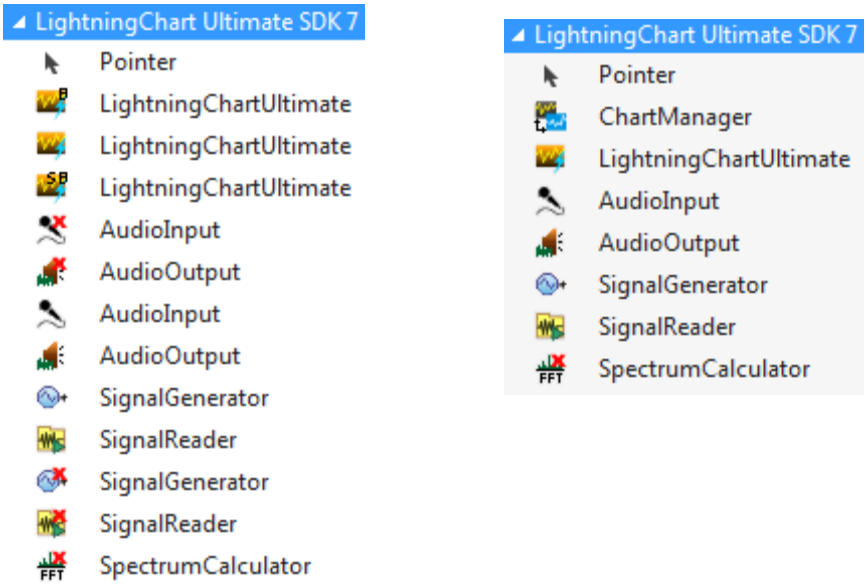


Figure 2-1. On the left, WPF toolbox components. On the right, WinForms toolbox components

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 16.

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 19.

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 20.

SignalGenerator Generates signal from multiple configurable waveform components. See chapter 17.

SignalReader Reads waveform data from a signal file, such as PCM formatted WAV. See chapter 18.

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 21.

2.3 Namespaces

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

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

3. Installation

3.1 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 2010-2015 for development, not required for deployment
- .NET framework v. 4.0 or newer installed

For LightningChart source code subscription, Visual Studio 2010-2013 is required to be able to compile Arction assemblies.

3.2 Run the setup wizard

Right-click on the **setup.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.

3.3 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.7\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 **.NET Framework 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.7\LibNet4***, and click open. The components can be now found in the toolbox.

3.4 Configuring Visual Studio 2010-2015 help manually

This chapter gives you the information how to install LightningChart Ultimate help content manually. You need this information if Visual Studio 2010-2015 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 2010-2015. You can either press F1 on LightningChart Ultimate's classes, properties etc. or use Microsoft Help Viewer to browse the help content.

3.4.1 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\LightningChartUltimate SDK v.7 \Toolbox\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.

3.4.2 Visual Studio 2012-2015

Follow these steps to manually install LightningChart Ultimate help content on Visual Studio 2012-2015:

1. Open Visual Studio 2012, 2013 or 2015.
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\LightningChartUltimate SDK v.7\Toolbox\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.

3.5 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.7\LibNet4***)

3.6 Selecting target framework

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

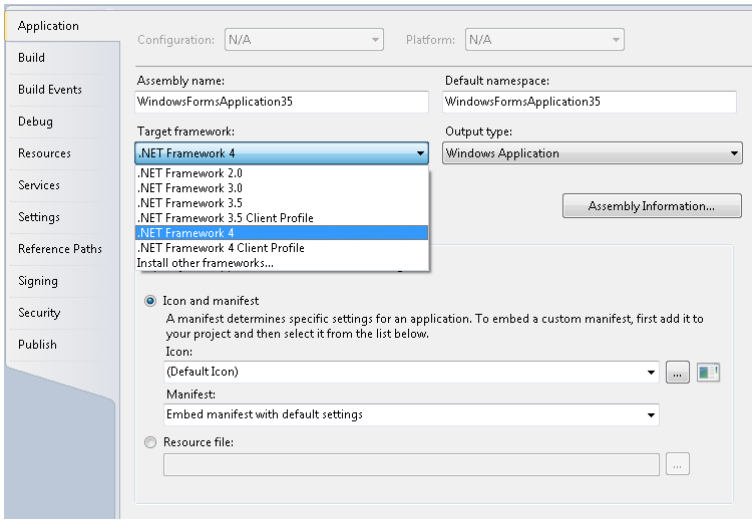


Figure 3-1. Selecting target framework in C# project.

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

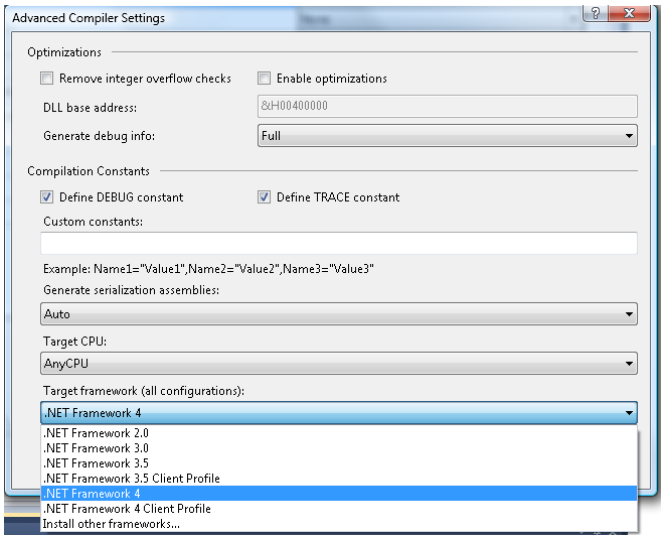


Figure 3-2. Selecting target framework in Visual Basic project.

- 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.

3.7 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
            b249NC42LjEuMjAwMSwgQ3VsdHVyZT1uZXV0cmFsLCBQdWJsaWNLZXlUb2tlbj03MmY1N
            WZiZDY5MDFm
            ... lots of encoded stuff ...
             YXlvdXQBAAAAB3ZhbHVlX18ACAIAAAAAAAAACw==
</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.

4. License management

4.1 Licenses and adding LightningChart or other control to your application

Arction components use a license key protection system. You can use the components only with a valid license. When you a 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 **.lic** file.

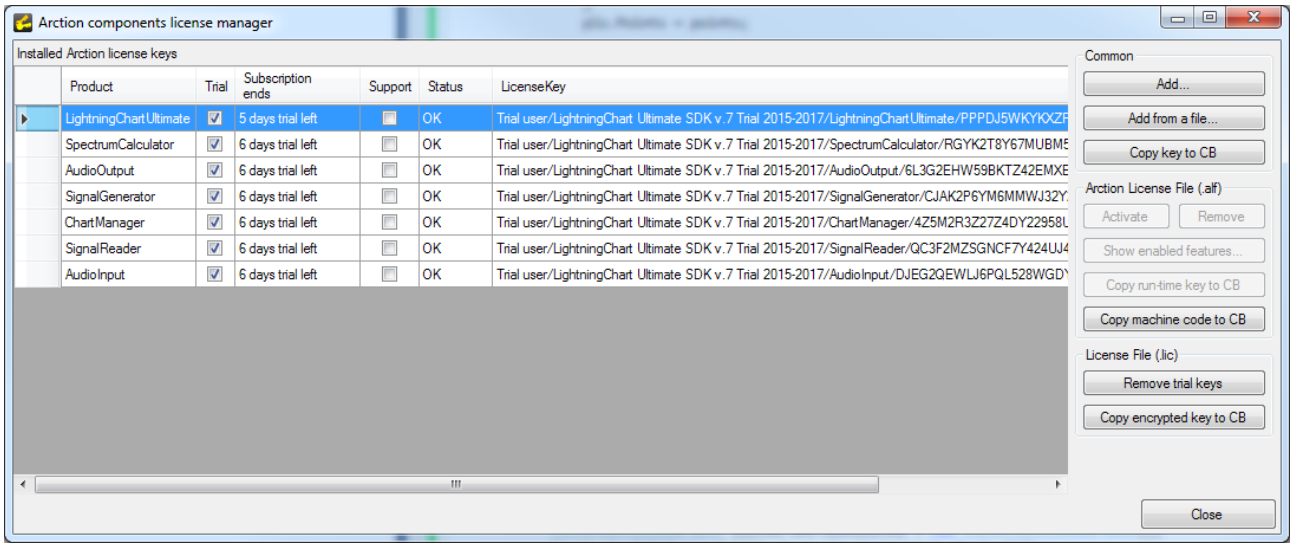


Figure 4-2. Add your license keys in LicenseManager.

Alternatively, you can add license keys beforehand by running the LicenseManager.exe from the Windows start menu Programs / Arction / LightningChart Ultimate SDK / **License Manager**.

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

4.1.1 Extracting license keys from LicenseManager

- Copying normal format license key to clipboard: Select row and click *Copy key to CB*
- Encrypted format: Company and developer name are not visible. Copy to clipboard by selecting the row and clicking *Copy encrypted key to CB*.

Arction components accept both key formats in constructor or LicenseKey property.

4.1.2 License storage when adding an Arction component from Toolbox into a form

Versions 5.1.5 and earlier used Microsoft's component model licensing scheme, which used licenses.licx to store the license info. This scheme caused a quite lot of trouble and has been replaced with ***LicenseKey*** property of the components in v.5.1.6 and newer.

When you drag a component from toolbox to a form, a license is retrieved from the Windows registry and ***LicenseKey*** property says "(From local registry)". When the application is built and run, the license is again retrieved from the registry.

NOTE! Set your actual key in LicenseKey property, otherwise license error will occur when running the application in other computer (deployment target).

4.1.3 License storage when creating a control by code

If you are creating Arction control by code, you can do that *with* or *without* license key parameter.

Without license key parameter

WinForms

```
LightningChartUltimate chart = new LightningChartUltimate();

chart.Parent = this; //where 'this' is a form or other container

chart.LicenseKey = "Trial user/LightningChart Ultimate SDK v.7 Trial 2015-2017/LightningChartUltimate/PPPDJ5WKYKXZFK32SXR6XB8JNXJT5N3YWMUU";
```

WPF

```
LightningChartUltimate chart = new LightningChartUltimate();

chart.LicenseKey = "Trial user/LightningChart Ultimate SDK v.7 Trial 2015-2017/LightningChartUltimate/PPPDJ5WKYKXZFK32SXR6XB8JNXJT5N3YWMUU";

grid.Children.Add(chart); //Grid is the container, parent of the chart.
```

If LicenseKey is left empty, the license key is then searched from the registry. **Remember to set LicenseKey property before deploying the software to end user.**

With normal license key parameter

WinForms

```
LightningChartUltimate chart = new LightningChartUltimate(
        "Trial user/LightningChart Ultimate SDK v.7 Trial 2015-
        2017/LightningChartUltimate/PPPDJ5WKYKXZFK32SXR6XB8JNXJT5N3YWMUU");

chart.Parent = this; //where 'this' is a form or other container
```

WPF

```
LightningChartUltimate chart = new LightningChartUltimate(
        "Trial user/LightningChart Ultimate SDK v.7 Trial 2015-
        2017/LightningChartUltimate/PPPDJ5WKYKXZFK32SXR6XB8JNXJT5N3YWMUU");

grid.Children.Add(chart); //Grid is the container, the parent of the chart.
```

With encrypted license key parameter

WinForms

```
LightningChartUltimate chart = new LightningChartUltimate(
        "Y3PT77S2RQRCKU6M6PR2EQ7XKJEHMSG9WHAZUQAHZLWY7RTFYU7GX6JBF8JRFQ3E4UH7
        MJ3PTT67KVLETDD5C4GTT7P77EEJ9VK7YKDQRBER5Y9NHFDW3GVAN7CDKMZA88MLWSP8S
        DF8YJV4MSUBCR9XN82BU2MTGAF92V7ZRULLZ3UCFRUK94G6RAELVZTR");

chart.Parent = this; //where 'this' is a form or other container
```

WPF

```
LightningChartUltimate chart = new LightningChartUltimate(
        "Y3PT77S2RQRCKU6M6PR2EQ7XKJEHMSG9WHAZUQAHZLWY7RTFYU7GX6JBF8JRFQ3E4UH7
        MJ3PTT67KVLETDD5C4GTT7P77EEJ9VK7YKDQRBER5Y9NHFDW3GVAN7CDKMZA88MLWSP8S
        DF8YJV4MSUBCR9XN82BU2MTGAF92V7ZRULLZ3UCFRUK94G6RAELVZTR");

grid.Children.Add(chart); //Grid is the container, the parent of the chart.
```

4.2 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.

5. LightningChartUltimate component

5.1 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***.

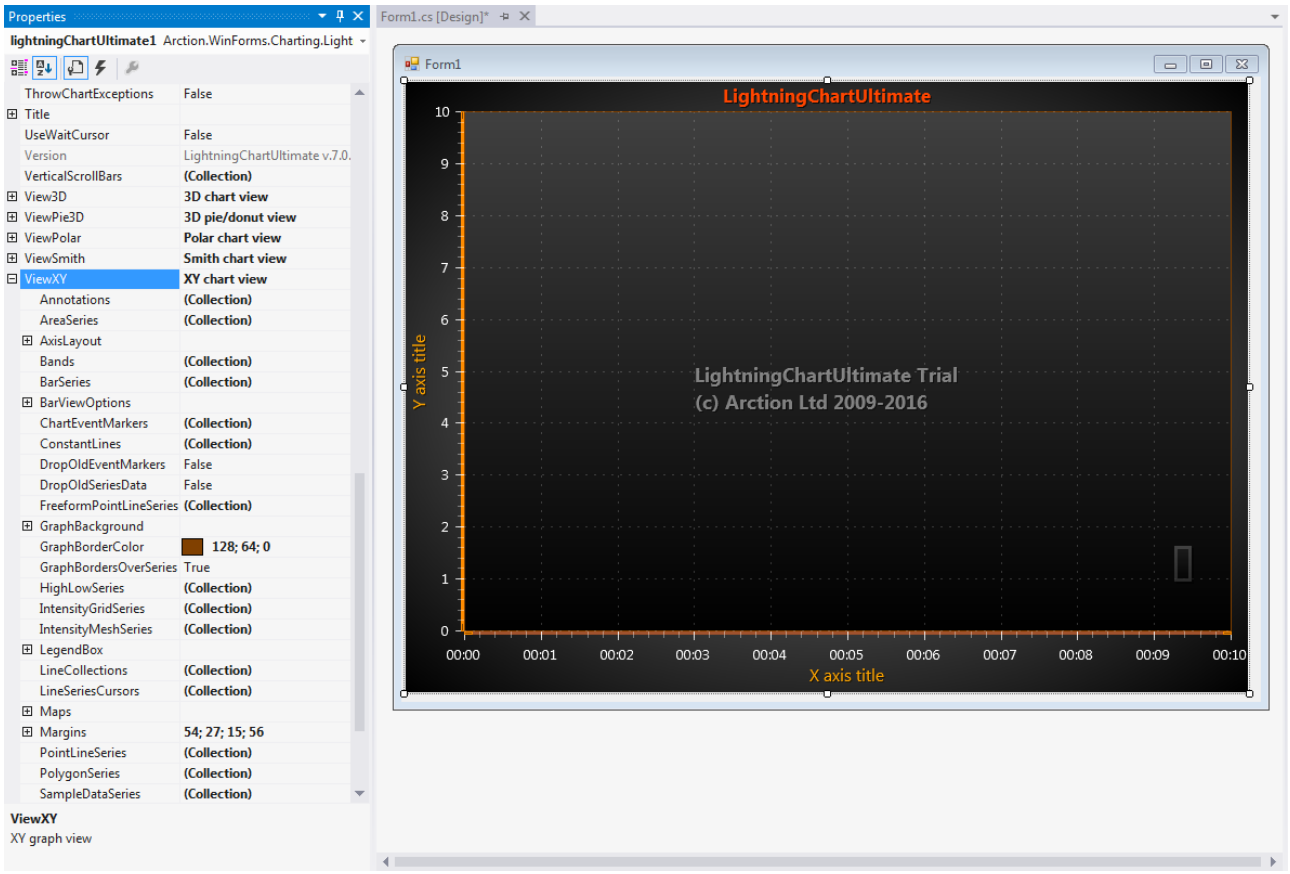


Figure 5-1. LightningChartUltimate control added into Windows Forms designer.

5.1.1 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.

5.1.2 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.

5.1.3 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.

5.2 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.

5.2.1 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

5.2.2 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.

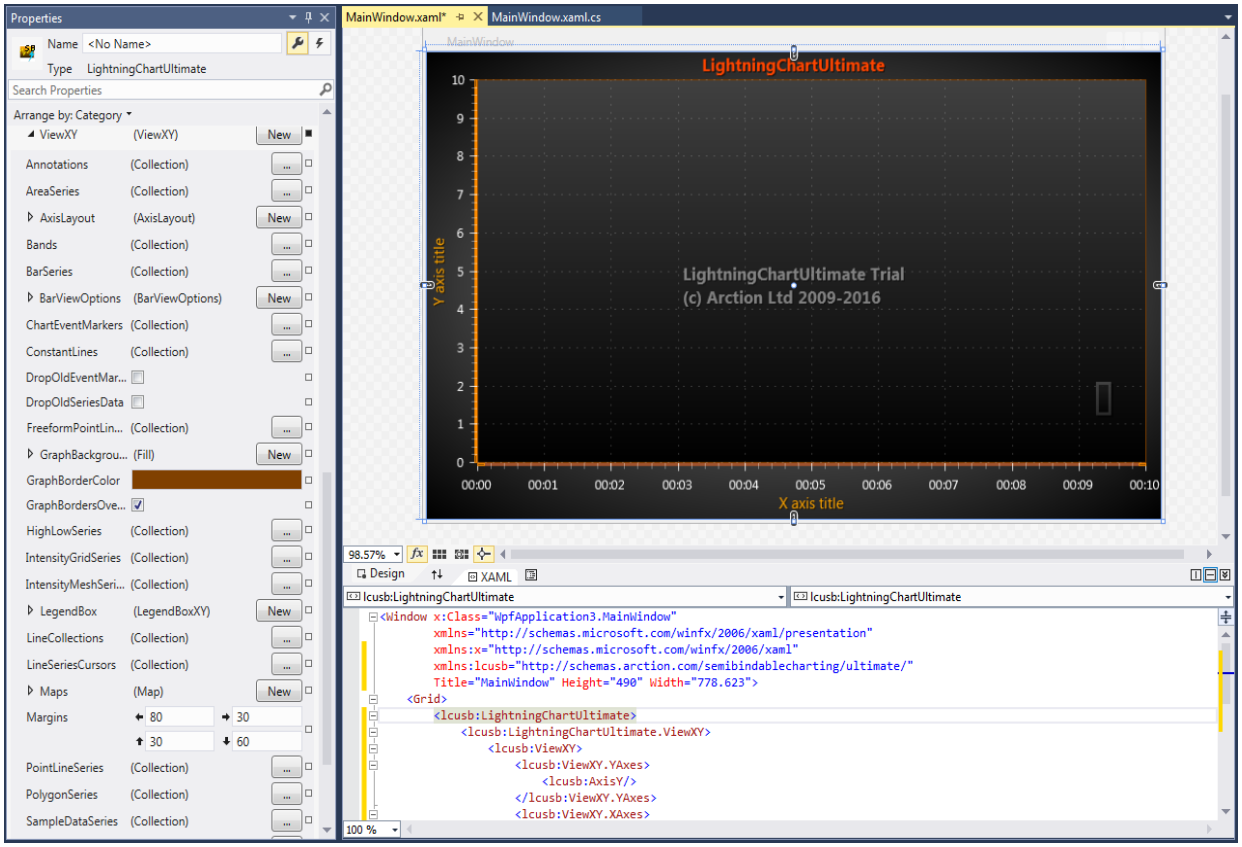


Figure 5-2. LightningChartUltimate control added into WPF designer.

5.3 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.7\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.

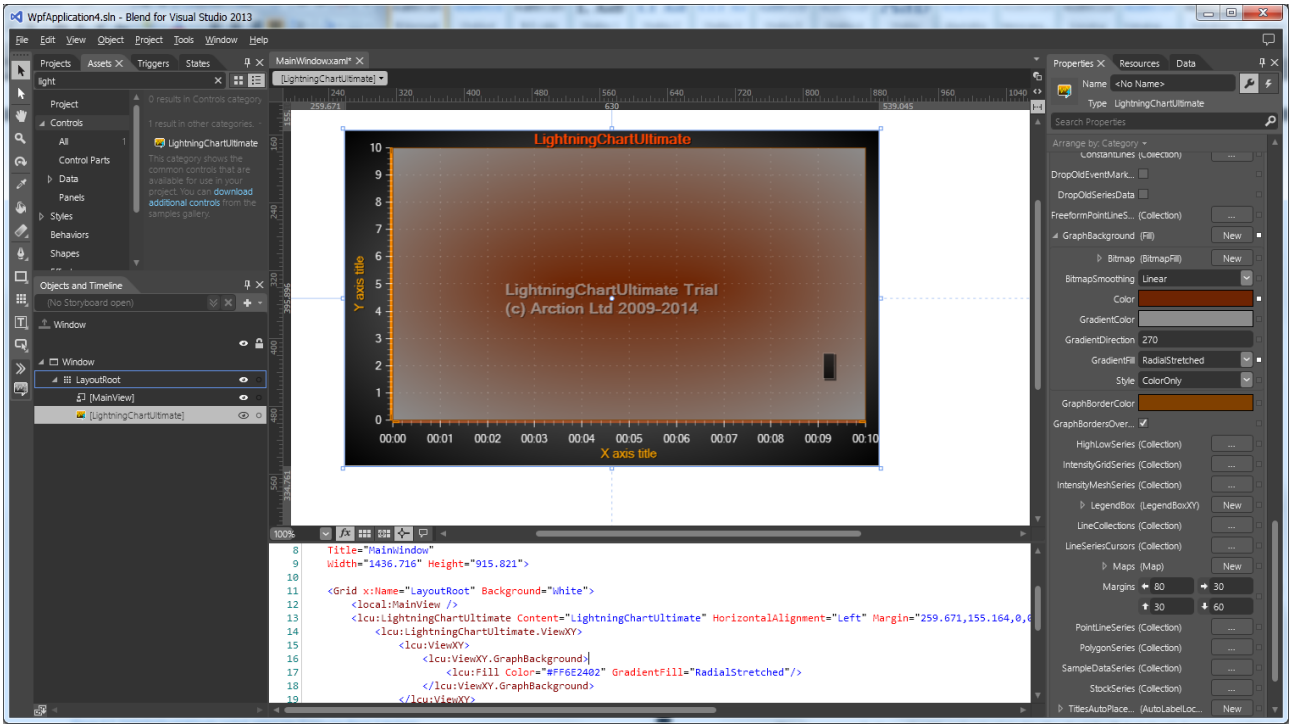


Figure 5-3. LightningChartUltimate control added into Blend For Visual Studio 2013 designer.

5.3.1 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.

5.3.2 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.

5.4 Using Windows Forms chart in WPF application

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.

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.

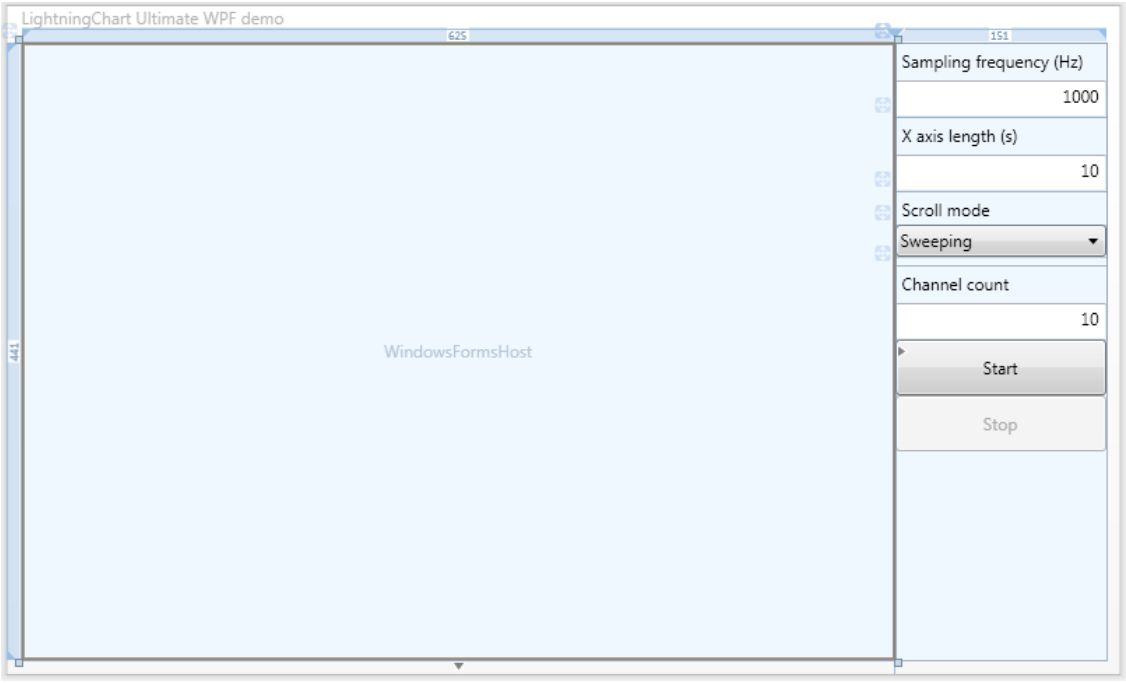


Figure 5-4 WPF example application in designer. WindowsFormsHost control keeps the LightningChartUlitmate object inside when the application is executed.

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()
{
        //Embeddable key can be produced with License manager program, from the
        installed license keys.
        m_chart = new LightningChartUltimate(" /*Create key in License manager
        program */");

        //Set the chart object as child to the WindowsFormsHost control
        windowsFormsHost1.Child = m_chart;

}
```

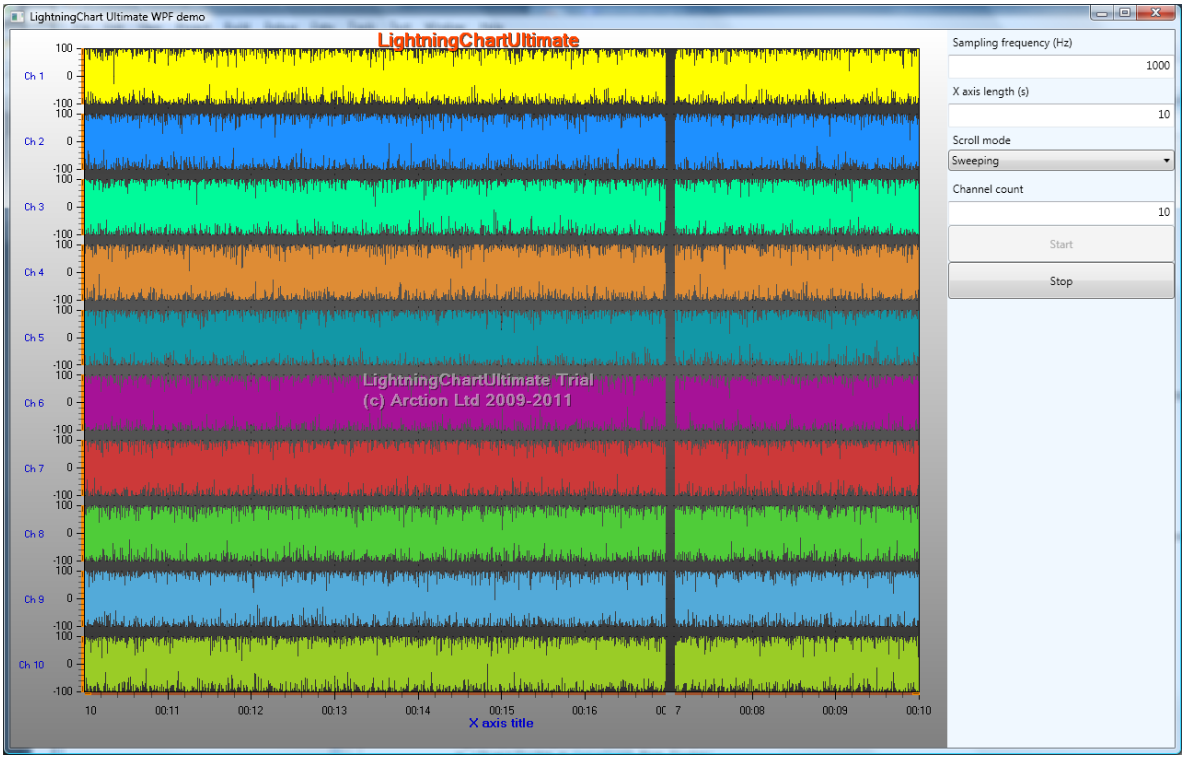


Figure 5-5 WPF demo application in executed. Refer to actual WPF demo application source code for feeding the real-time data on-the-fly.

5.5 Object model

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

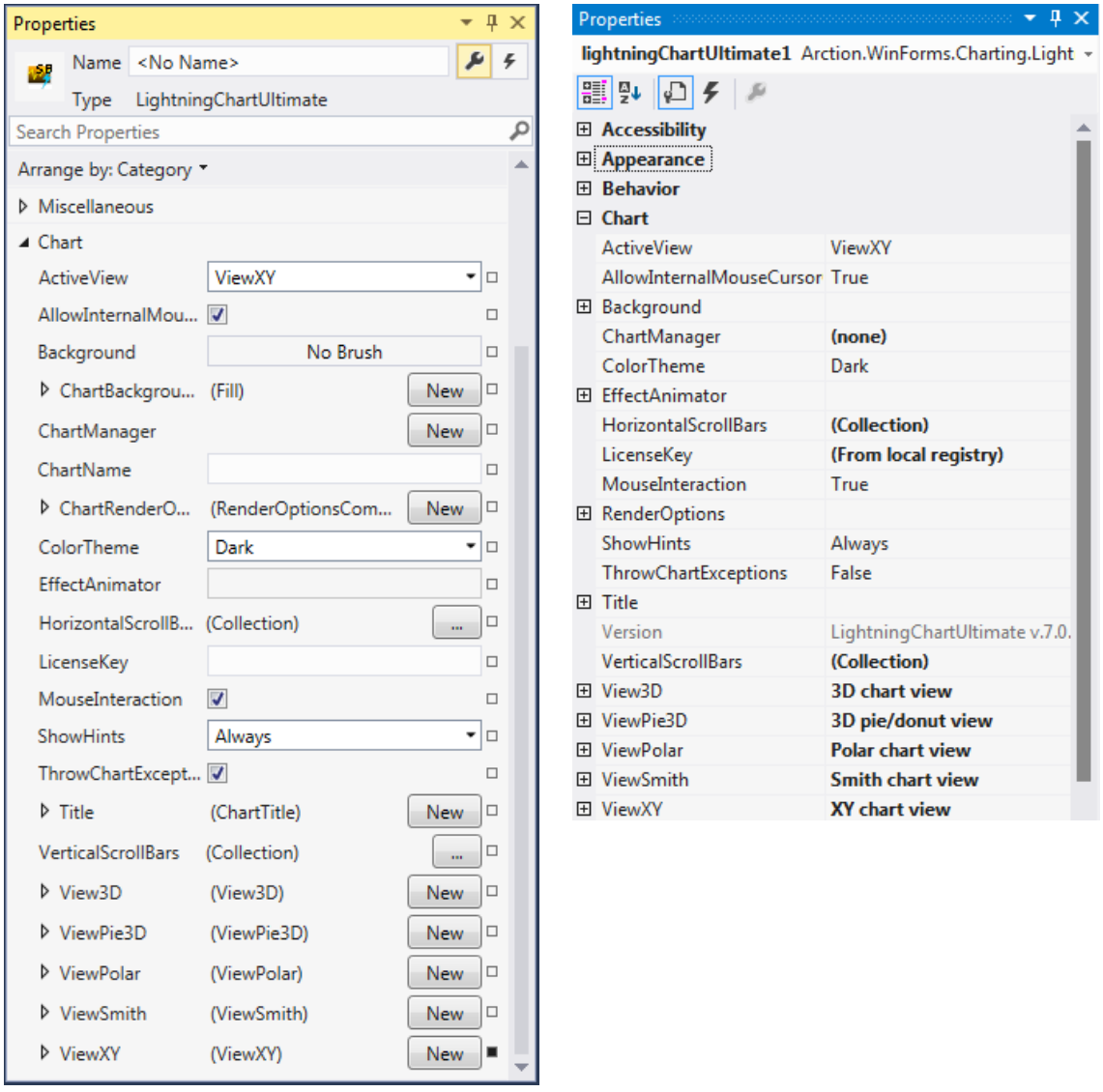


Figure 5-6. LightningChart specific properties can be found under *Chart* category in both Windows Forms and WPF Properties window. By expanding the nodes, or in WPF creating new objects, you will notice a huge set of properties exist in overall.

5.5.1 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
Rendering options property	RenderOptions	ChartRenderOptions
Background fill property	Background	ChartBackground
Fonts	System.Drawing.Font	Arction.WPF.LightningChartUltimate.WPFFont
Colors	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 6)

- *View3D* (see chapter 7)

- *ViewPie3D* (see chapter 8)

- ***ViewPolar*** (see chapter 9)

- ***ViewSmith*** (see chapter 10)

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

5.6 Configuring appearance / performance settings

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

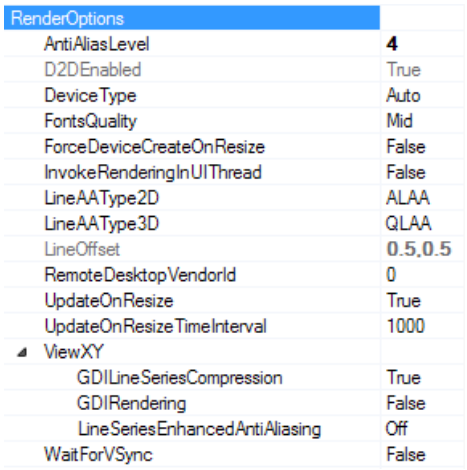


Figure 5-7 RenderOptions properties.

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)

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.

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, False. 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.

6. 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 6.24.

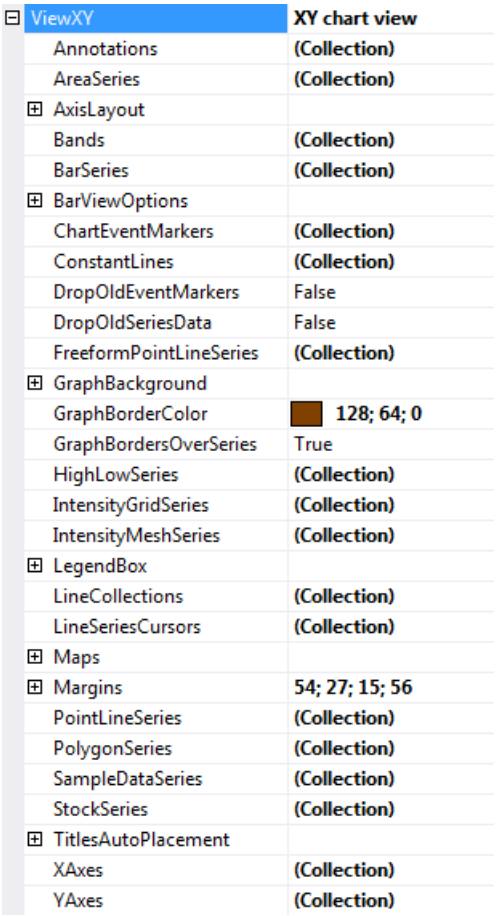


Figure 6-1. ViewXY object tree.

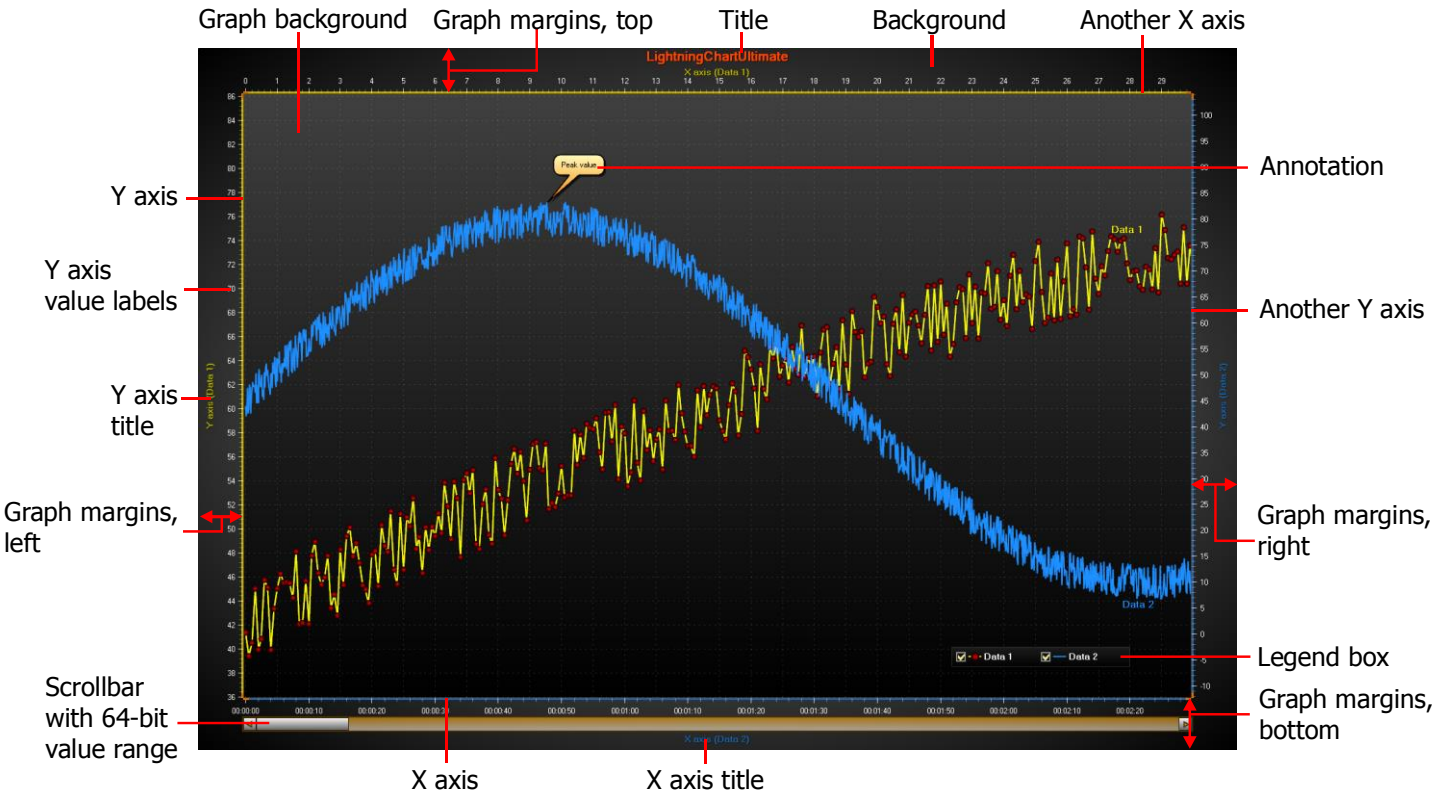


Figure 6-2. Quick overview of ViewXY parts

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*** property. There are plenty of filling options available.

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 6.2.

X axis
See section 6.3.

Annotations
See section 6.19.

Legend box
See chapter 6.20.

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 11.

6.1 Axis layout options

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

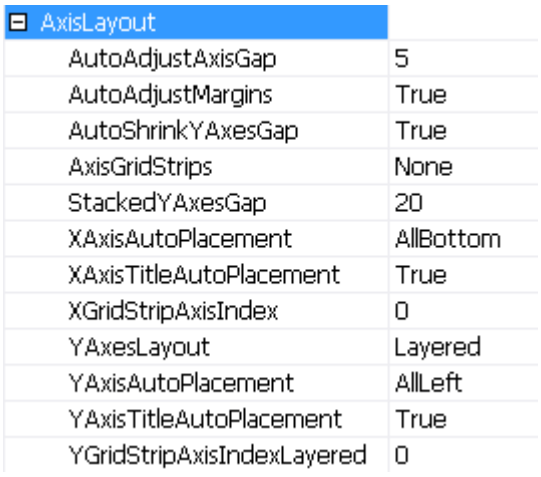


Figure 6-3. AxisLayout property tree.

6.1.1 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.

6.1.2 Setting how axes are placed

X axis automatic placement

XAxisAutoPlacement controls how the X axes are placed vertically.

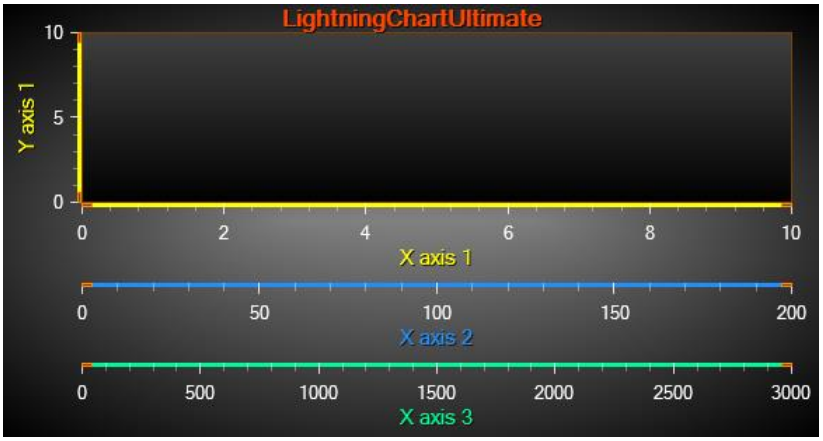


Figure 6-4. XAxisAutoPlacement = AllBottom. Three X axes added, all are positioned below the graph.

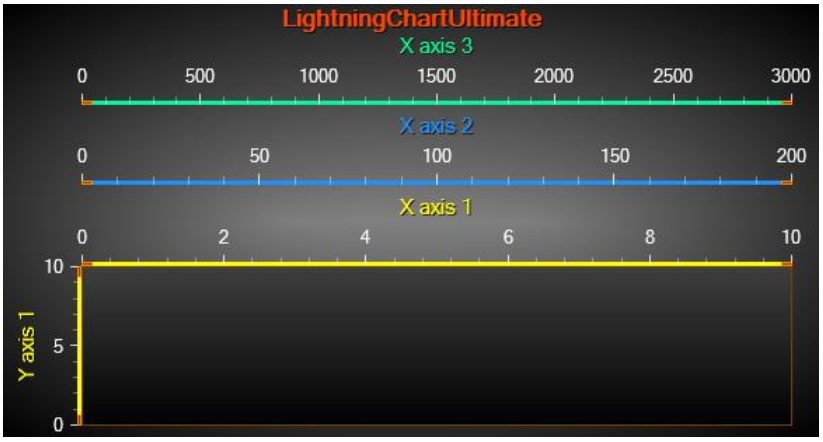


Figure 6-5. XAxisAutoPlacement = AllTop. All X axes are positioned above the graph.

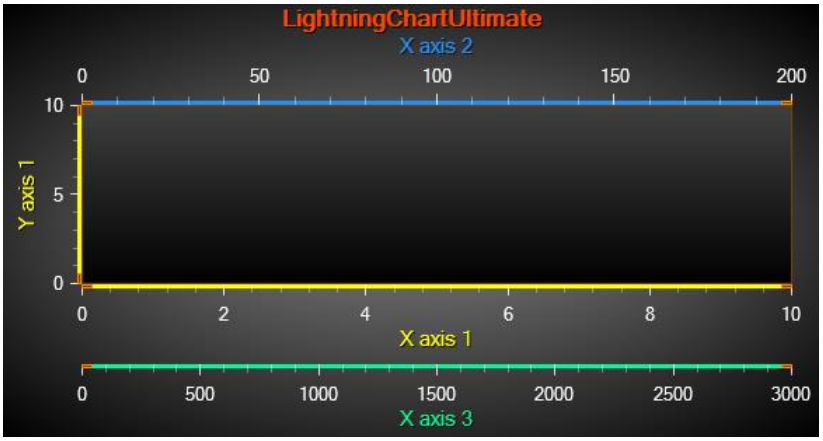


Figure 6-6. XAxisAutoPlacement = BottomThenTop. Axes are distributed below and above the graph, every other axis to the opposite side, starting from bottom.

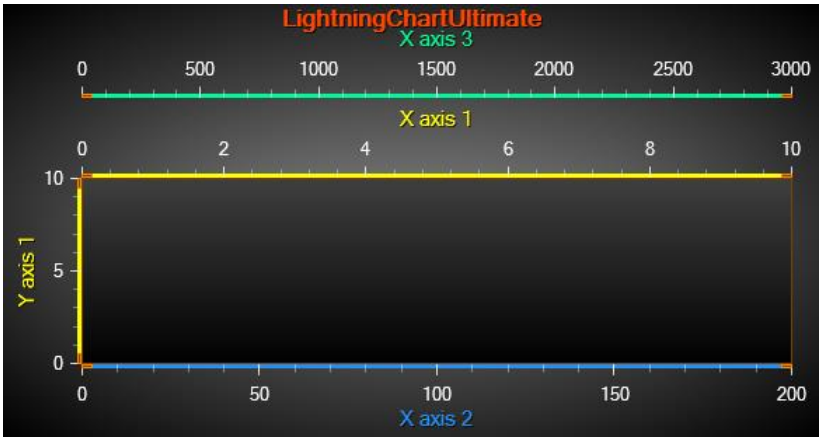


Figure 6-7. XAxisAutoPlacement = TopThenBottolm. Axes are distributed below and above the graph, every other axis to the opposite side, starting from top.

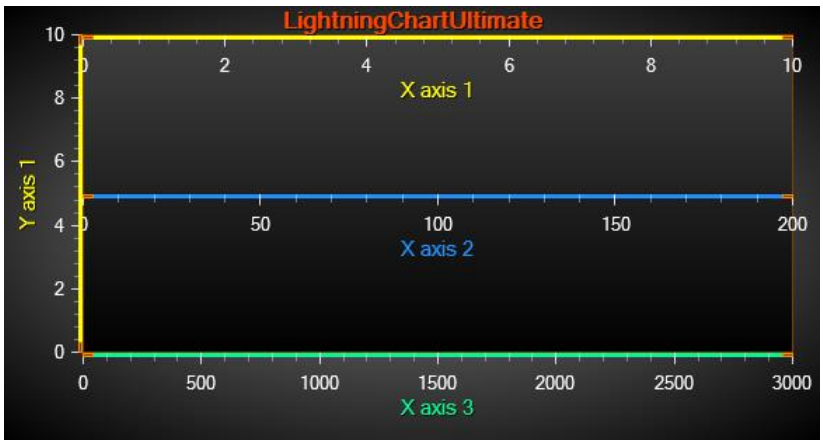


Figure 6-8. XAxisAutoPlacement = Off. Automatic axis placement is disabled, and Position and alignment properties of each axis applies separately. First axis Position = 0, Second axis Position = 50 and Third axis position = 100.

Y axis automatic placement

YAxisAutoPlacement controls how the Y axes are placed vertically.

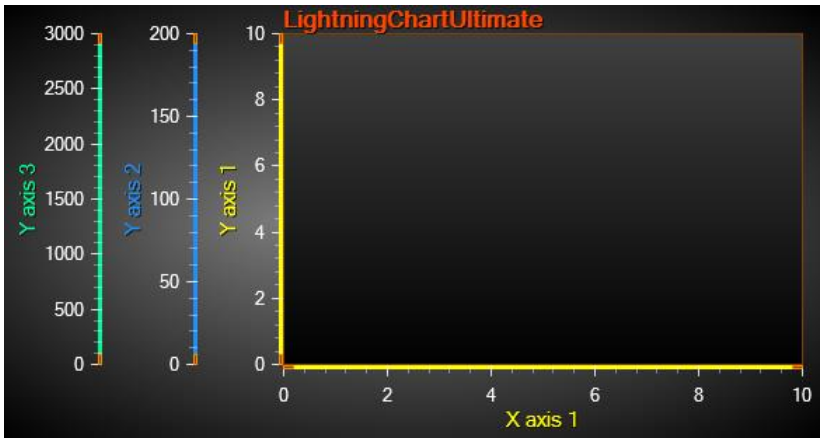


Figure 6-9. YAxisAutoPlacement = AllLeft. Three Y axes added, all are positioned to the left side of the graph.

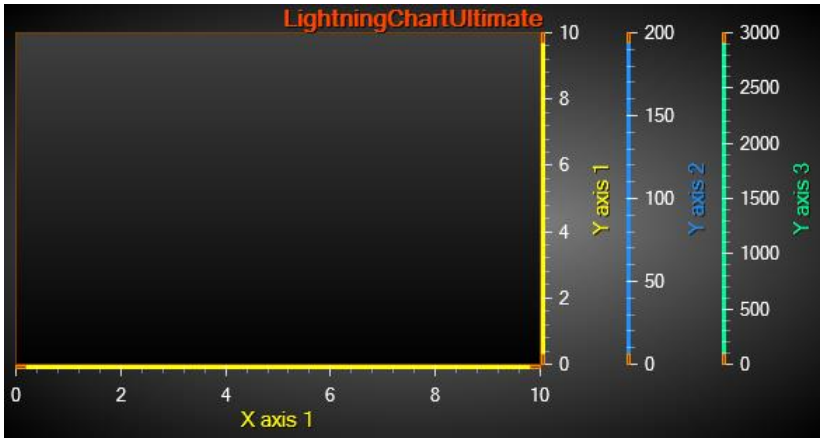


Figure 6-10. YAxisAutoPlacement = AllRight. All Y axes are positioned to the right side of the graph.

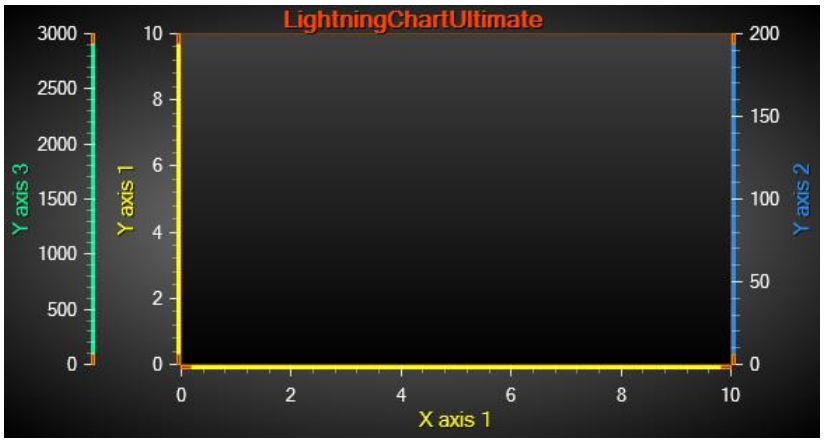


Figure 6-11. YAxisAutoPlacement = LeftThenRight. Axes are distributed to left and right side of the graph, every other axis to the opposite side, starting from left side.

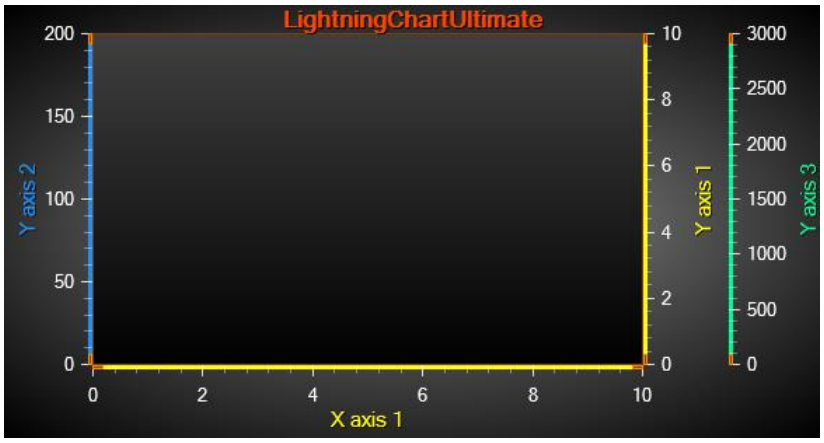


Figure 6-12. YAxisAutoPlacement = RightThenLeft. Axes are distributed to left and right side of the graph, every other axis to the opposite side, starting from right side.

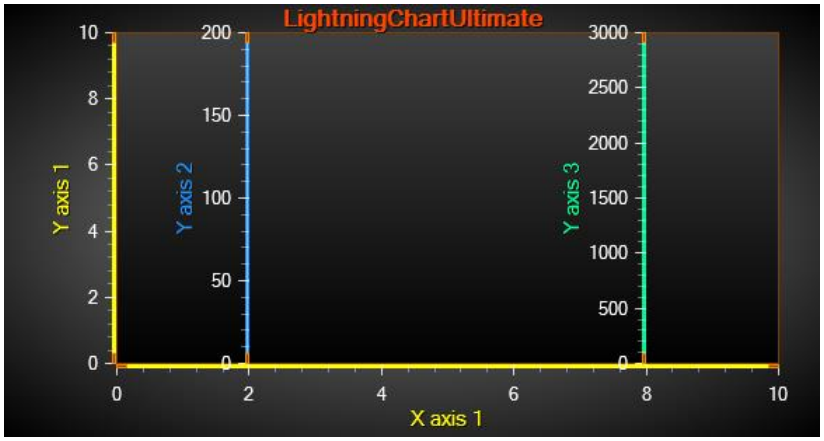


Figure 6-13. YAxisAutoPlacement = Off. Automatic axis placement is disabled, and Position and alignment properties of each axis applies separately. First axis Position = 0, Second axis Position = 20 and Third axis position = 80.

6.1.3 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 selected by ***ViewXY.AxisLayout.YAxesLayout*** property.

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.

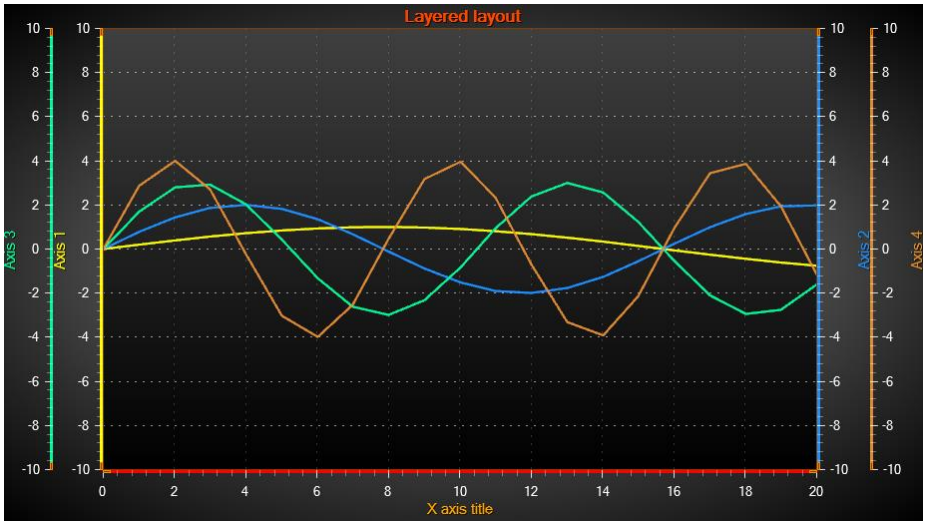


Figure 6-14. An example view of 4 Y axes in YAxesLayout = Layered.

Stacked

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

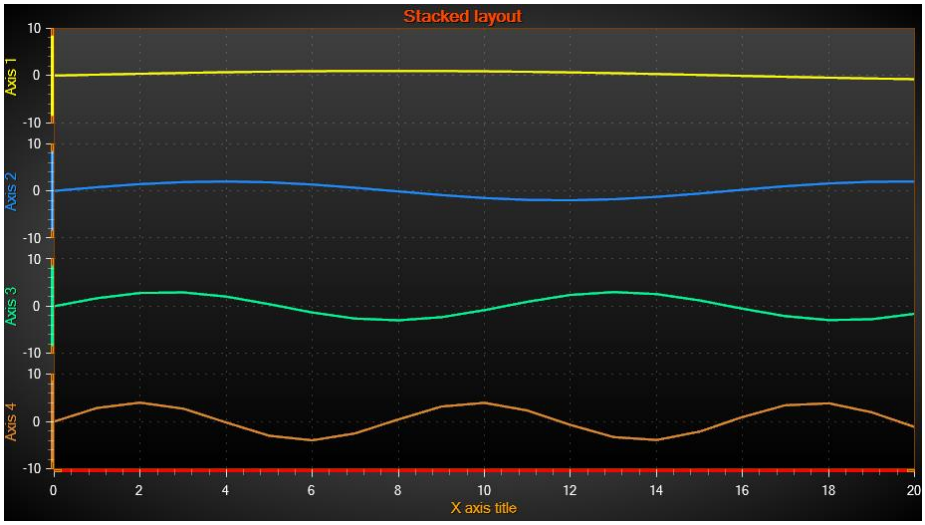


Figure 6-15. An example view of 4 Y axes in YAxesLayout = Stacked.

Segmented

In Stacked view the vertical space is divided between ***Segments***. Each segment can contain several Y axes. Each segment relational height can bet 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.

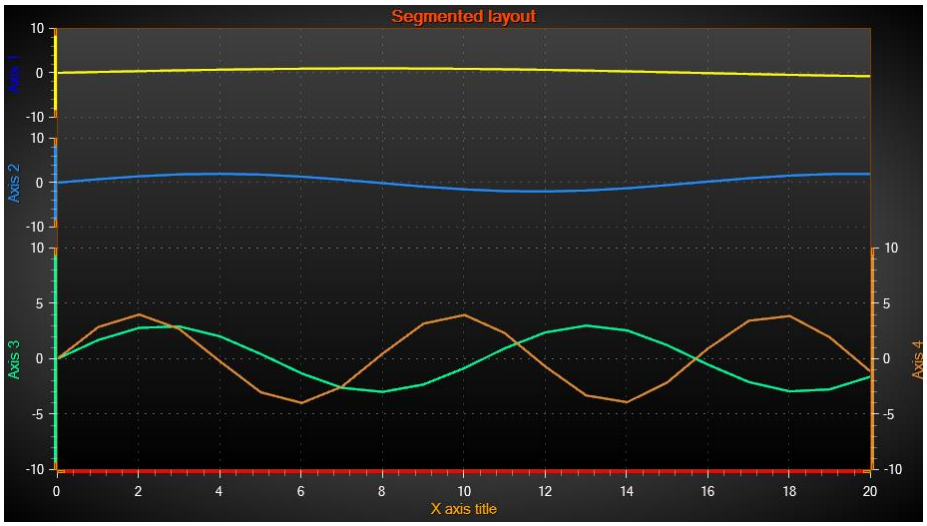


Figure 6-16. An example view of 4 Y axes in YAxesLayout = Segmented. First two segments have Height = 1, and last segment has Height of 2.5. Axis1.SegmentIndex = 0, Axis2.SegmentIndex = 1, Axis3 and Axis4.SegmentIndex = 3.

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

6.1.4 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.

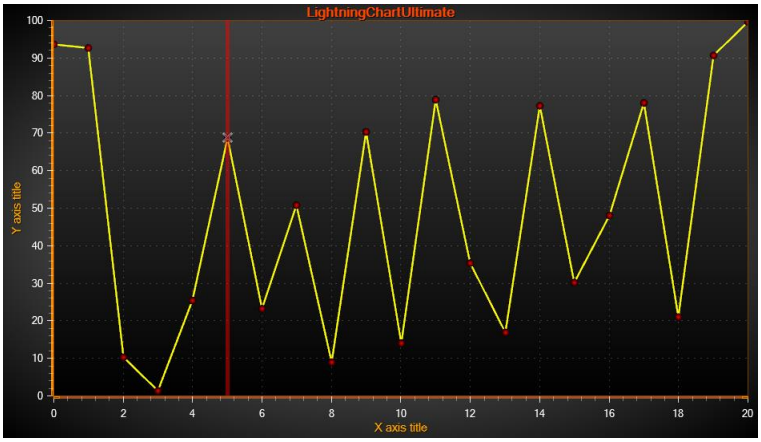


Figure 6-17. AxisGridStrips = None.

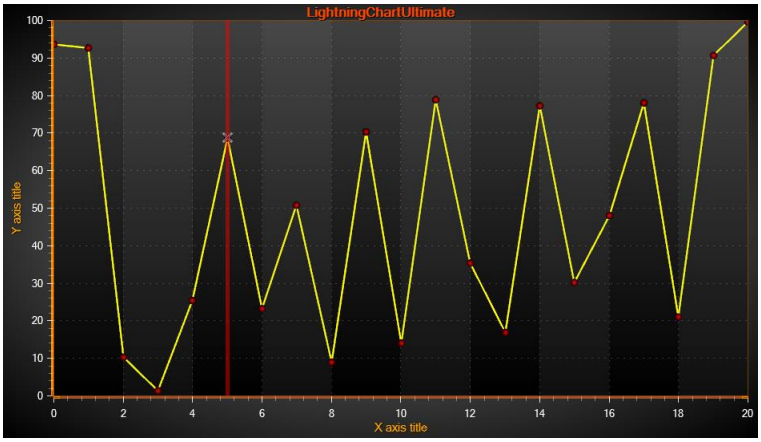


Figure 6-18. AxisGridStrips =X.

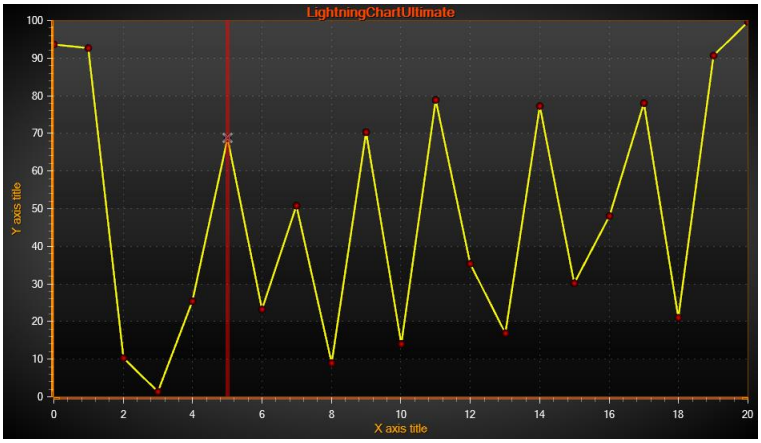


Figure 6-19. AxisGridStrips = Y.

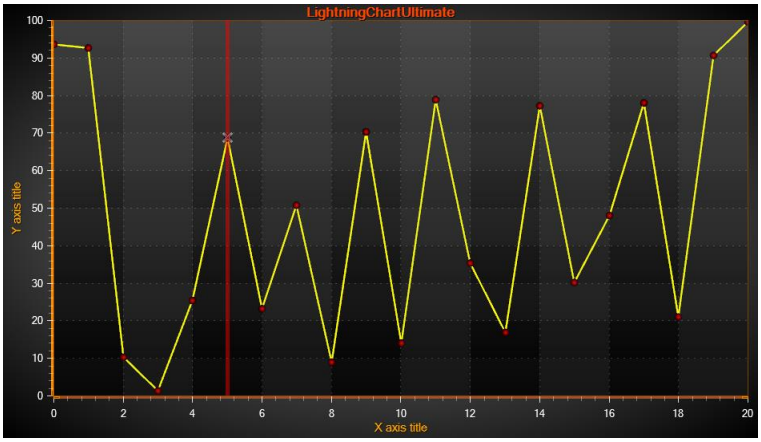


Figure 6-20. AxisGridStrips = Both.

6.1.5 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.

6.2 Y axes

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

6.2.1 AxisY class properties

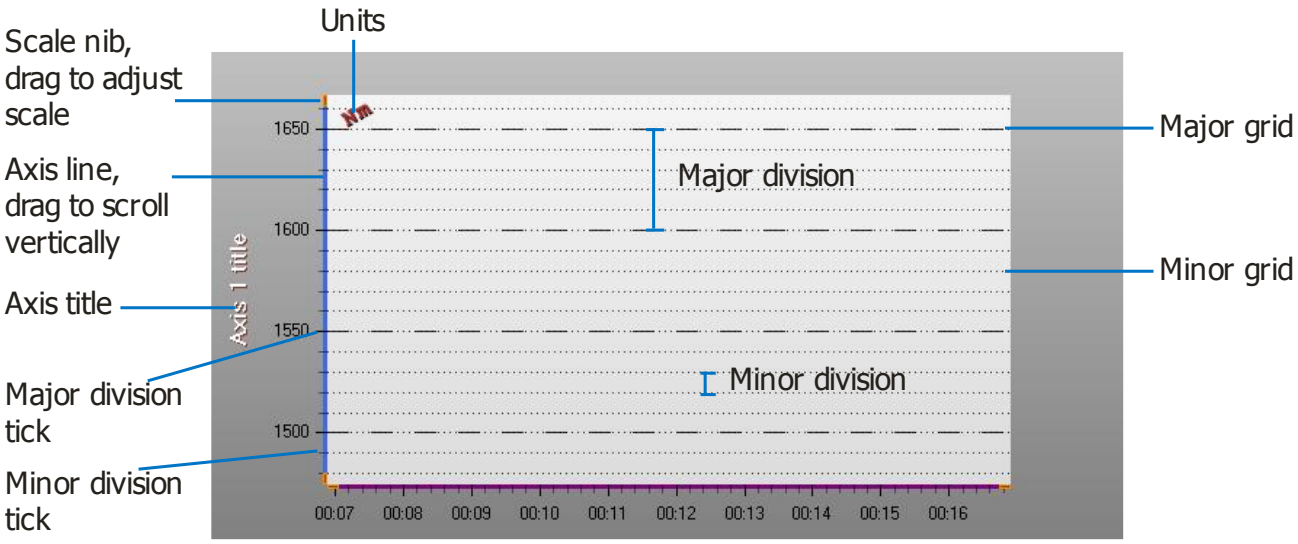


Figure 6-21. Y axis, divisions and grid.

6.2.2 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. ".ffffffff") allowing good zoomed views. ***LabelsNumberFormat*** property is generally used, when ***AutoFormatLabels*** is enabled, to format the decimal presentation of the numbers.

6.2.3 Value type

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

Number

Regular numeric format, for integer and decimal presentation. When ***AutoFormatLabels*** is disabled, ***LabeslNumbersFormat*** 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 adviced 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° -79.948862°`

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°2'13"N 9°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°02'13"N 9°58'02"W`

6.2.4 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.

6.2.5 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 6.21.5 for details.

6.2.6 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.

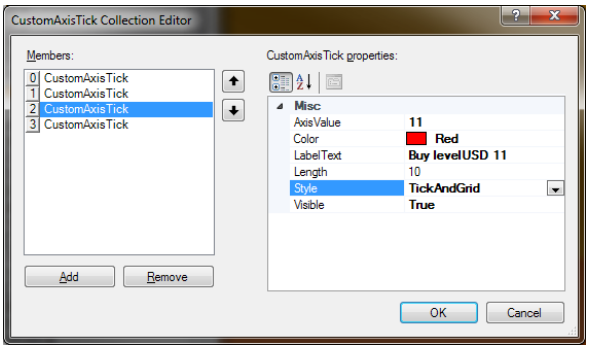
Major division tick style can be set from ***MajorDivTickStyle*** property. Edit the ticks and labels orientation by using ***MajorDivTickStyle.Alignment*** property. The value labels are drawn next to major division ticks. Edit the minor division properties with ***MinorDivTickStyle*** property, respectively.

6.2.7 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.

6.2.8 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.

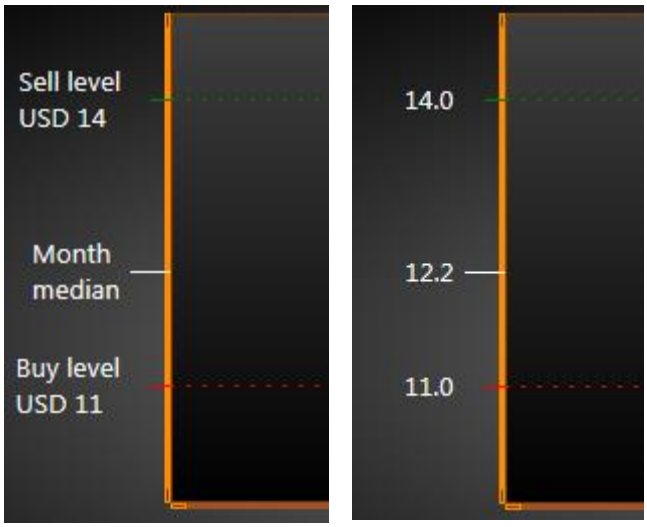


Figure 6-22. Custom ticks on Y axis. On the Left, axis.AutoFormatLabels = false. On the right, AutoFormatLabels = True.

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.

6.2.9 Reversed X and Y axis

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

6.2.10 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***.

Exponential presentation for 10 base

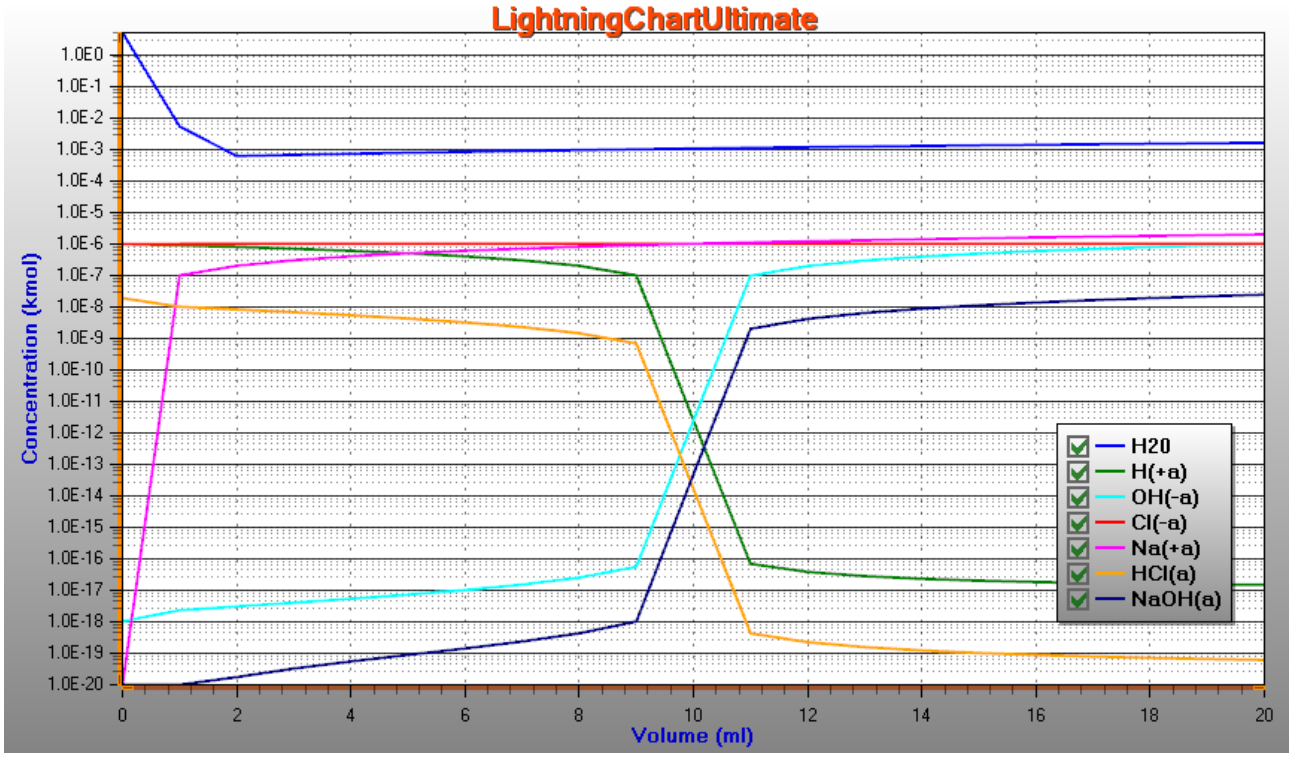


Figure 6-23. Logarithmic Y axis with values near zero. LogZeroClamp is set to 1.0E-20. LogBase is set to 10, LogLabelsType is set to Log10Exponential, to show the values in 1.0E presentation.

Natural logarithm

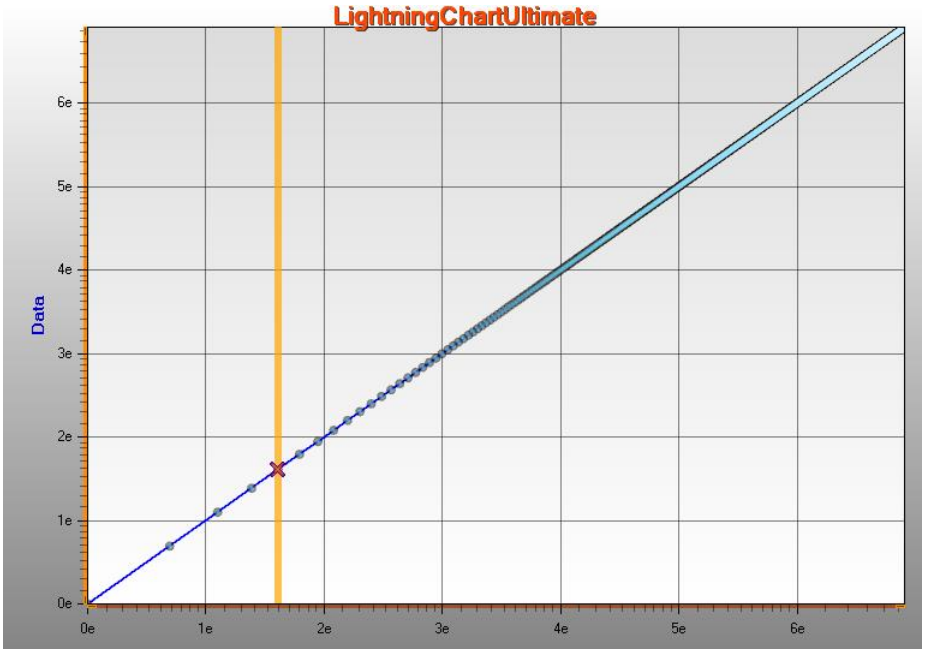


Figure 6-24. Natural logarithm view. LogBase is set to Math.E LogLabelsType is set to LogE_MultiplesOfNeper.

6.2.11 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).

6.2.12 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 6-25. MiniScale in the bottom-right corner of the graph.

6.3 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.

6.3.1 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.

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.

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.

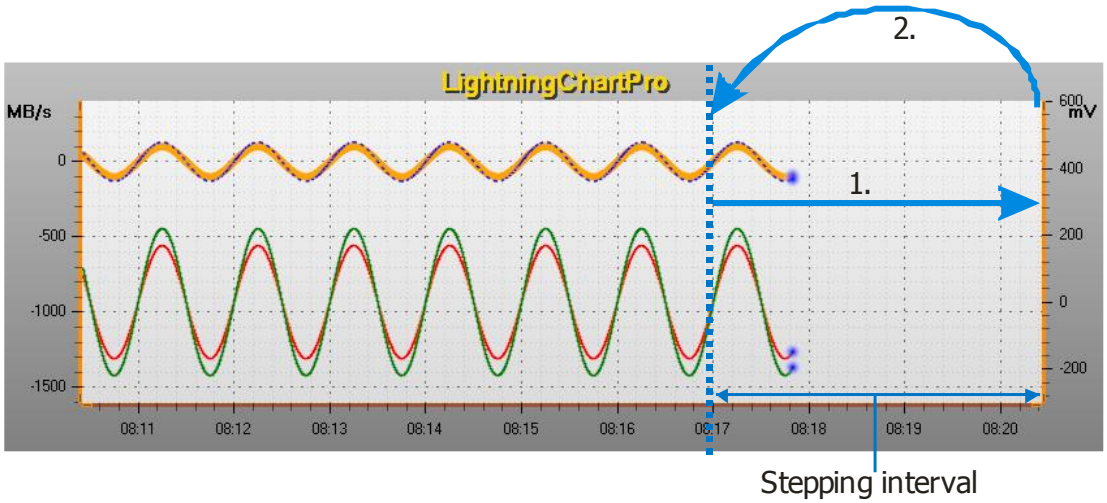


Figure 6-26. X axis scroll mode: stepping

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.

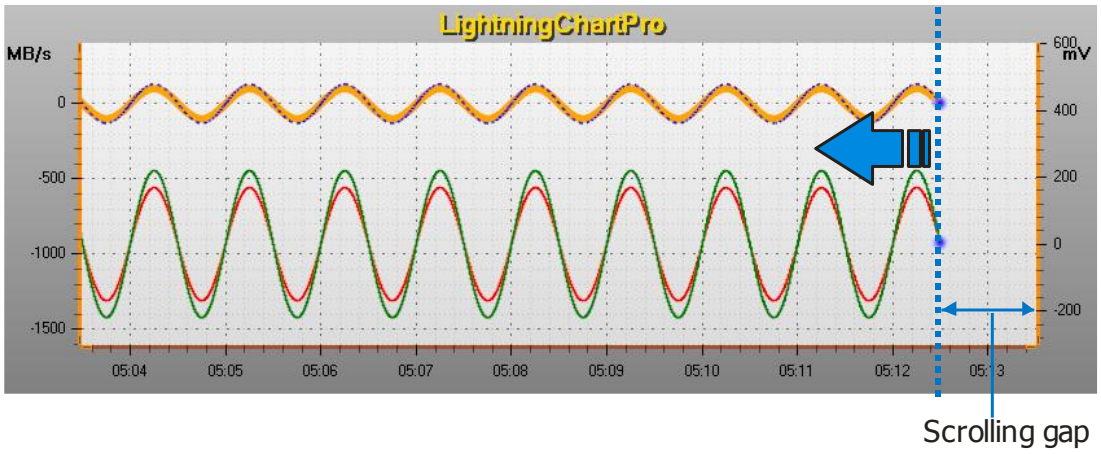


Figure 6-27. X axis scroll mode: scrolling

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
series.AddPoints(), ScrollStabilizing *disabled*	Perfect	Impaired	Good
series.AddPoints(), ScrollStabilizing *enabled*	Perfect	Best	Slightly impaired
series.AddPoints(), InvalidateData()	Impaired	Impaired	Perfect

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.

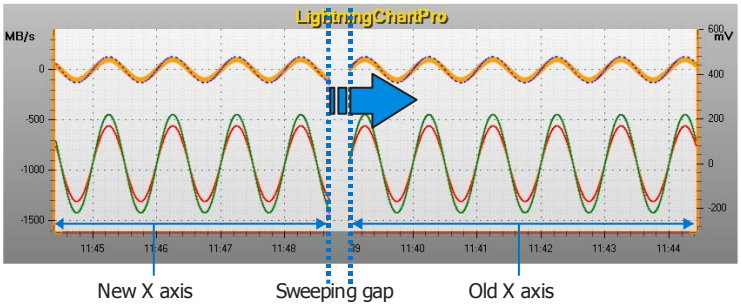


Figure 6-28. X axis scroll mode: sweeping

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.

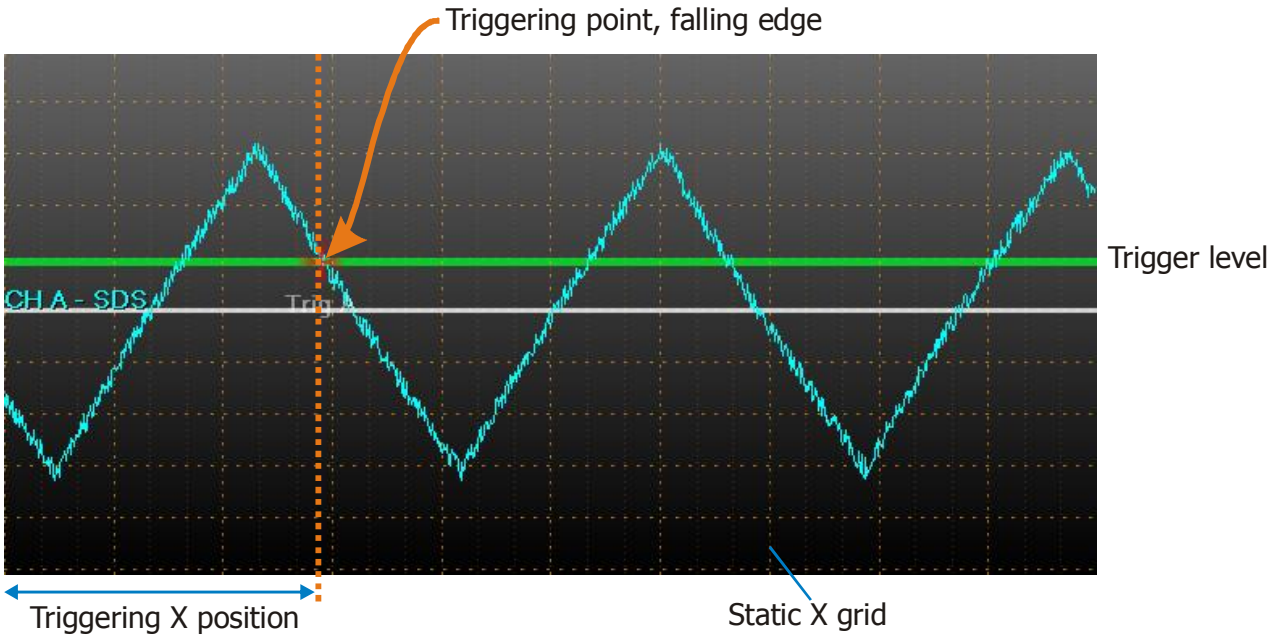


Figure 6-29. X axis scroll mode: Triggered with static X grid.

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. Instead, you should use static X grid. Hide the regular X axis objects by setting ***LabelsVisible*** = false, ***MajorGrid.Visible*** = false and ***MinorGrid.Visible*** = false. Then, show the static X grid by setting ***MajorStaticXGridOptions*** and ***MinorStaticXGridOptions***.

For scale indication, you can use Y axis **MiniScale** or define an **Annotation** (see section 6.19) object to show range like “200 ms/div”.

6.4 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.

6.5 Point line series

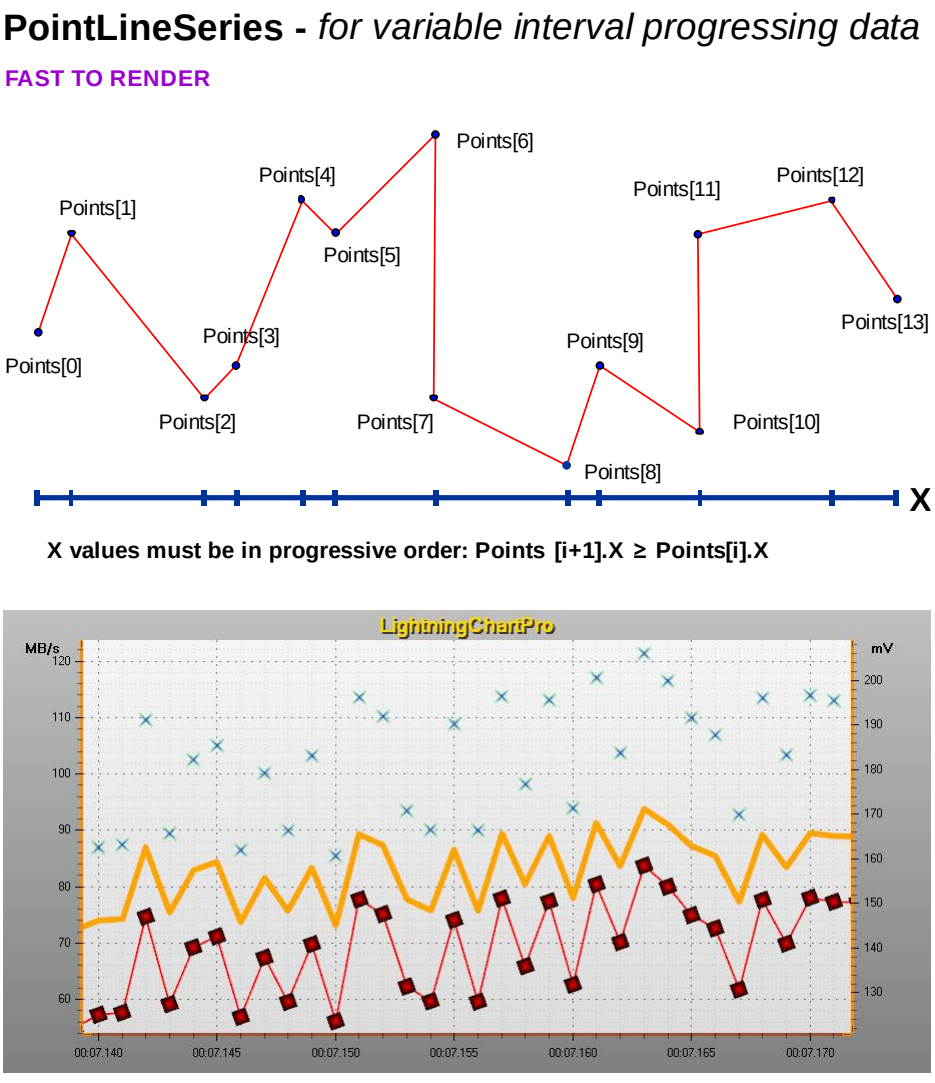


Figure 6-30 Some point line series.

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.

6.5.1 Line style

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

6.5.2 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.

6.5.3 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 corresponds that color in ***PointStyle*** property.

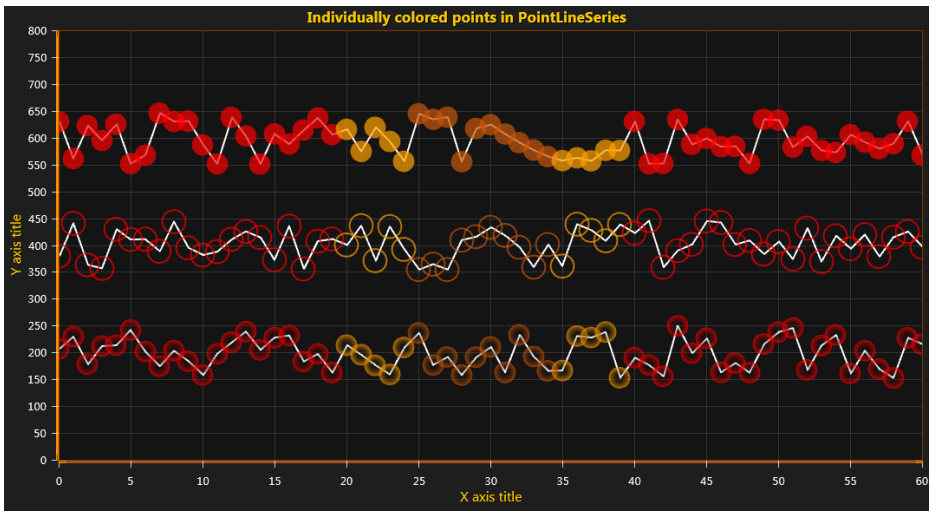


Figure 6-31 In top, IndividualPointColoring = Color1 (solid colored point). In the middle, IndividualPointColoring = BorderColor. In the bottom, IndividualPointColoring = Color2 (having gradient coloring with Color1 = transparent).

6.5.4 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***.

6.5.5 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

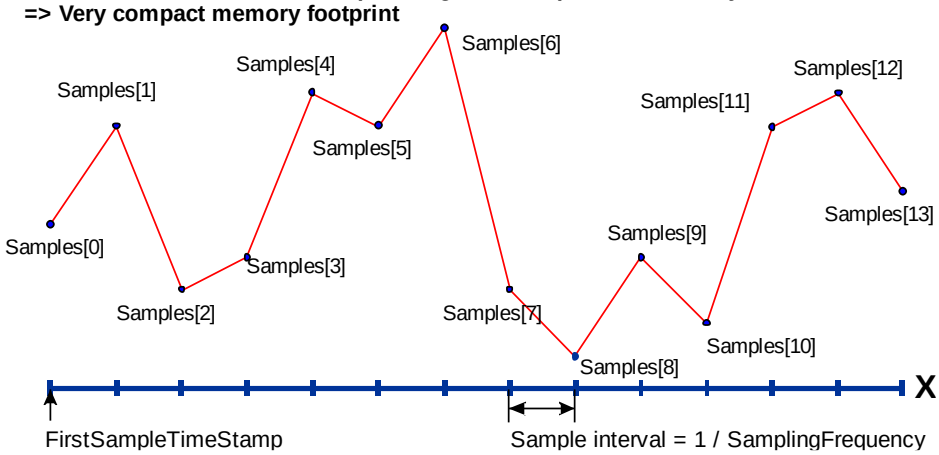
```
chart.ViewXY.PointLineSeries[0].SetValues(xValuesArray, yValuesArray);
```

6.6 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.

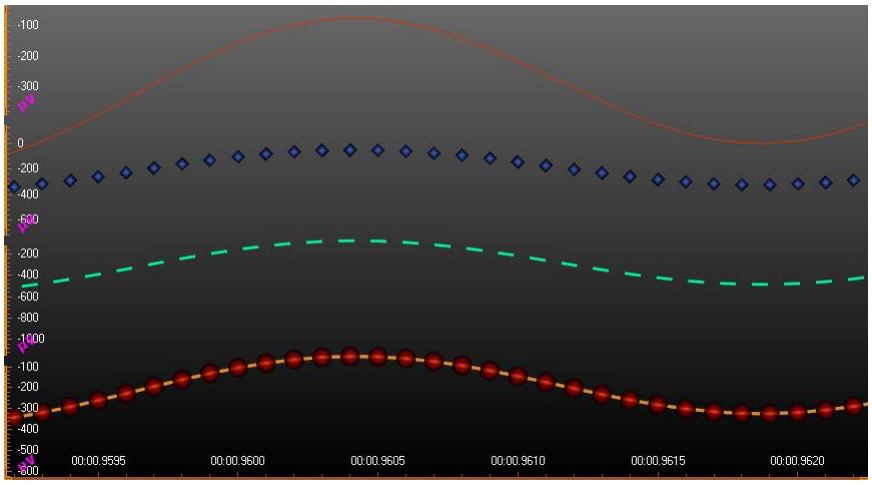


Figure 6-32. Some sample data series.

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.

6.6.1 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.

6.6.2 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;
```

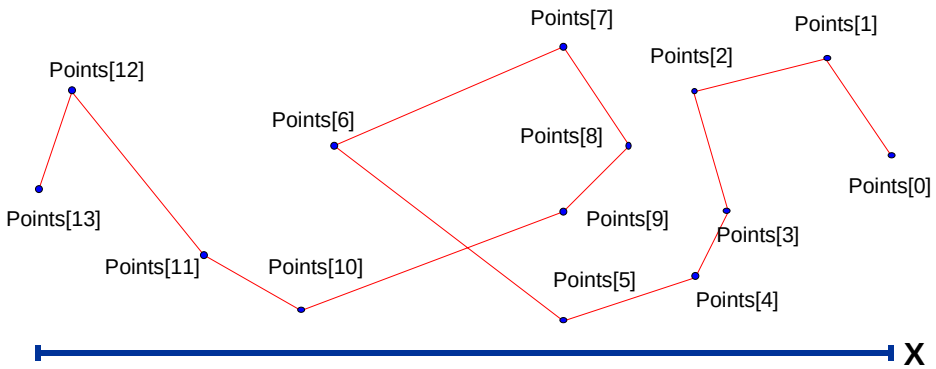
Or if ***SampleFormat*** Is ***DoubleFloat***

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

6.7 Freeform point line series

FreeformPointLineSeries - *for arbitrary data*

HEAVY TO RENDER WHEN POINT COUNT IS VERY HIGH



X values can be in any order

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.

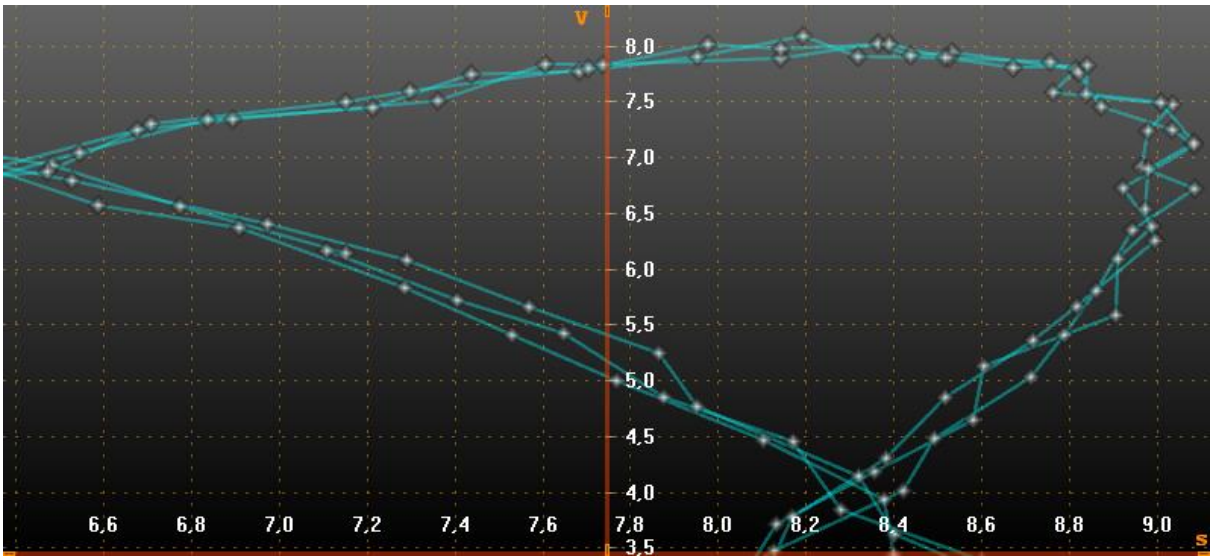


Figure 6-33. A freeform point line series

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.

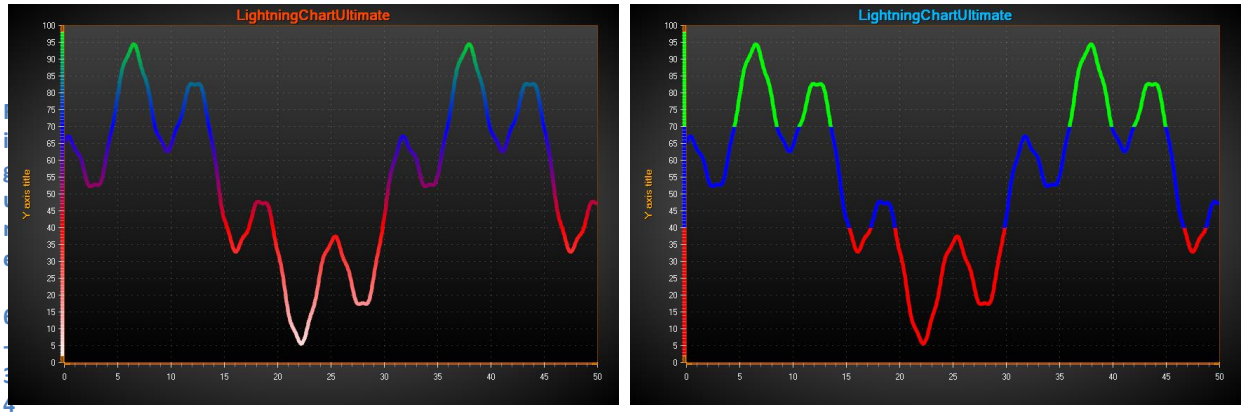
6.8 Advanced line coloring of line series

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

6.8.1 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.



. 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 6-35. Gradient palette coloring for bipolar signal data. UsePalette for Y axis is disabled.

6.8.2 Custom shaping and coloring with CustomLinePointColoringAndShaping event

Custom coloring logic 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***.

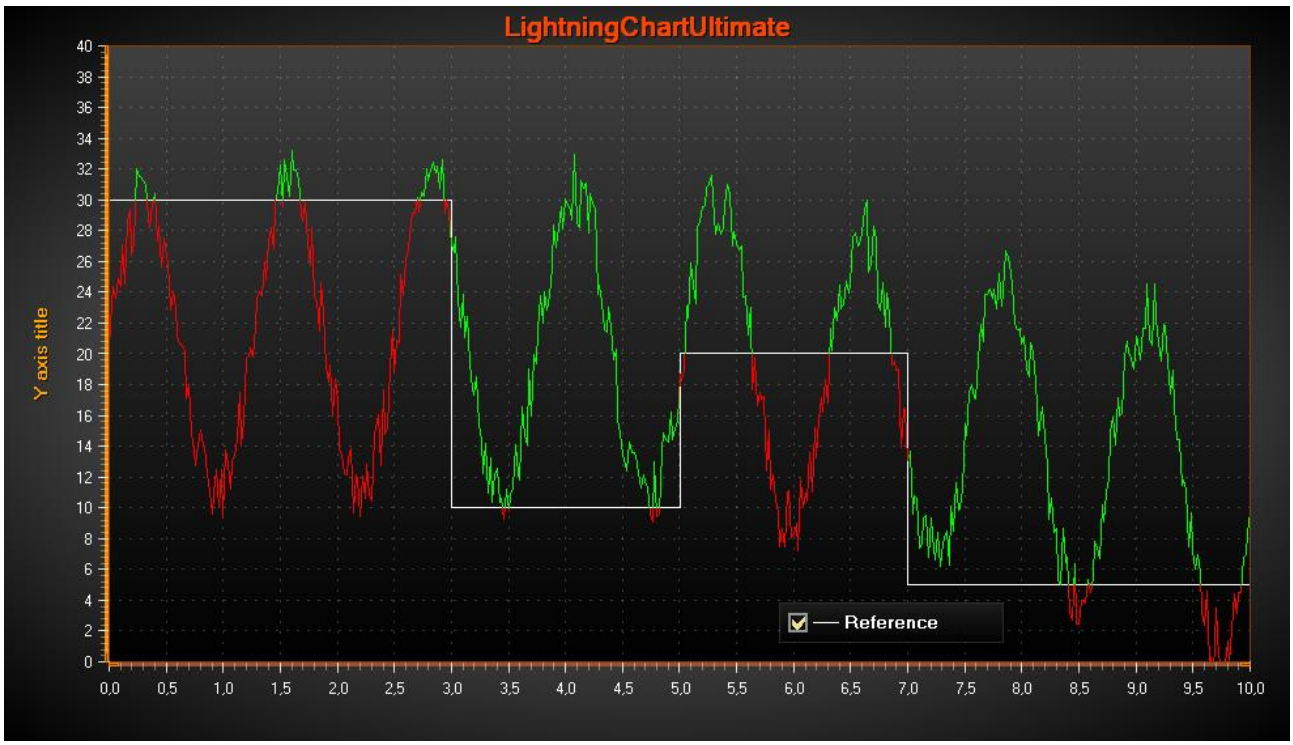


Figure 6-36. CustomLinePointColorAndShaping event handler used to transform color by the specific changing reference level.

6.9 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.

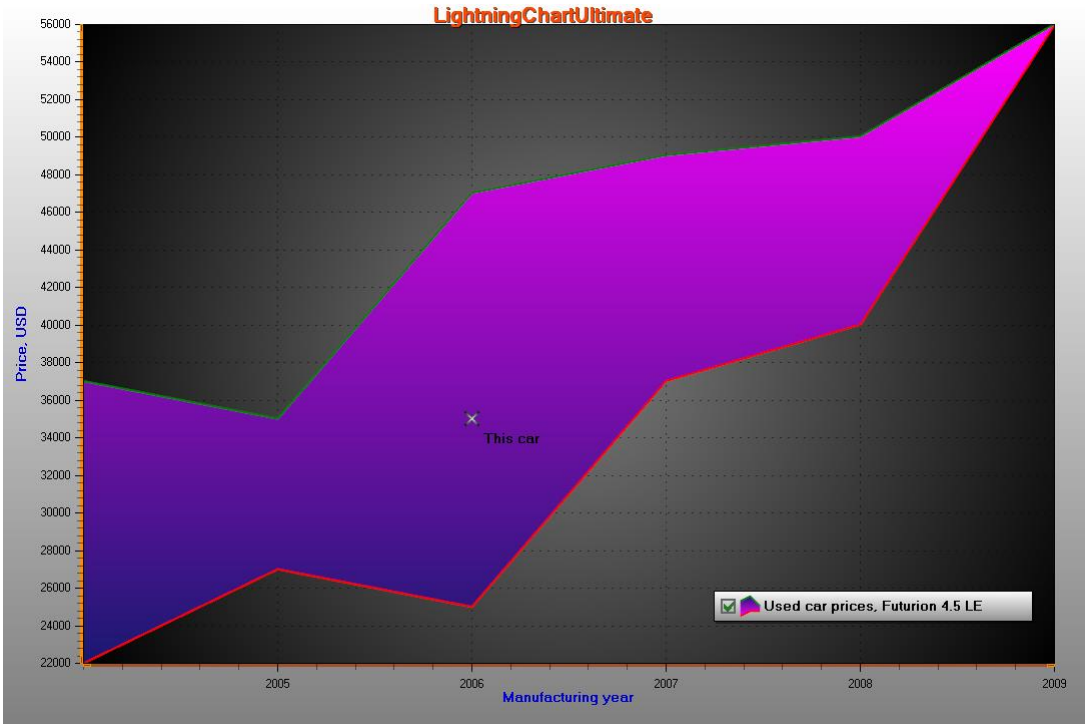


Figure 6-37. A high-low series with a marker over it.

6.9.1 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 6.5.1 and 6.5.2 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.

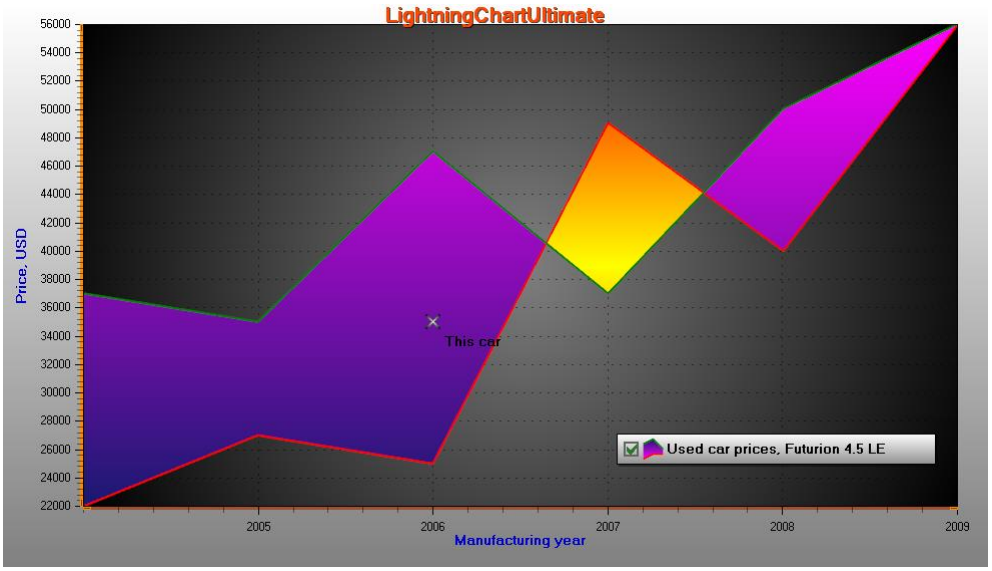


Figure 6-38. Fourth data item is given reversed: high value is < low value.

6.9.2 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.

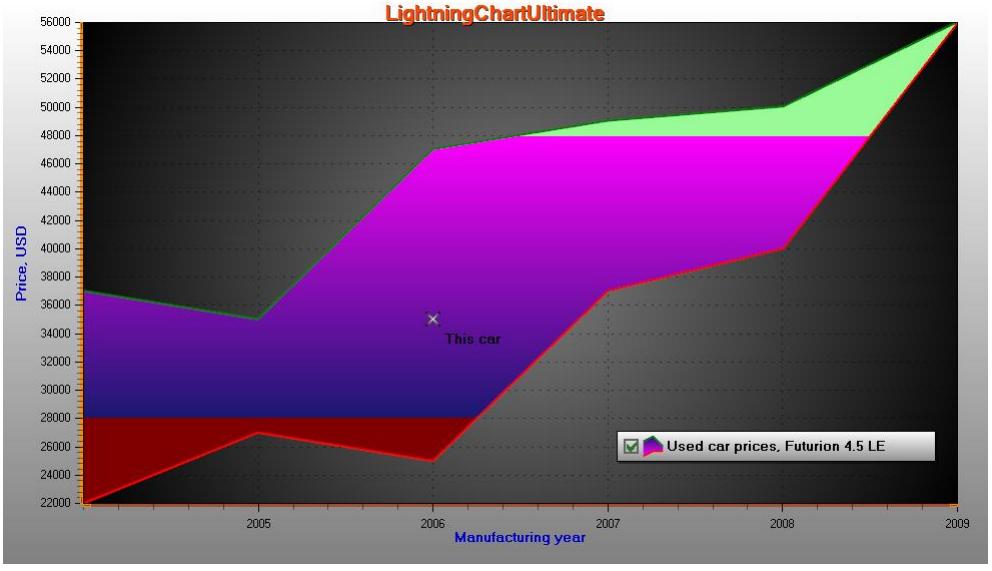


Figure 6-39. UseLimits = true, ExceedLimit = 48000 and DeceedLimit = 28000.

6.9.3 Coloring by value-range palette

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

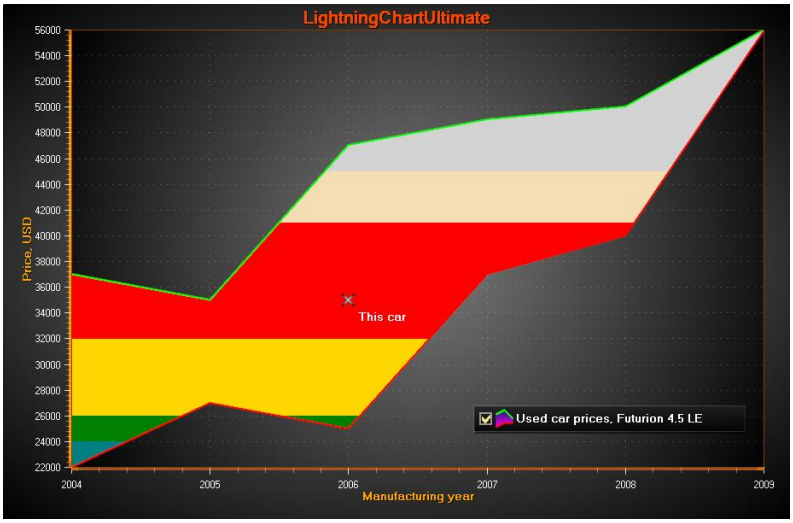


Figure 6-40. UsePalette = True, several steps defined in ValueRangePalette. Uniform coloring.

6.9.4 Adding data

You must add the data values in code. The data must be given in ascending order by X values.

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;
```

6.10 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 6.9, but simpler.

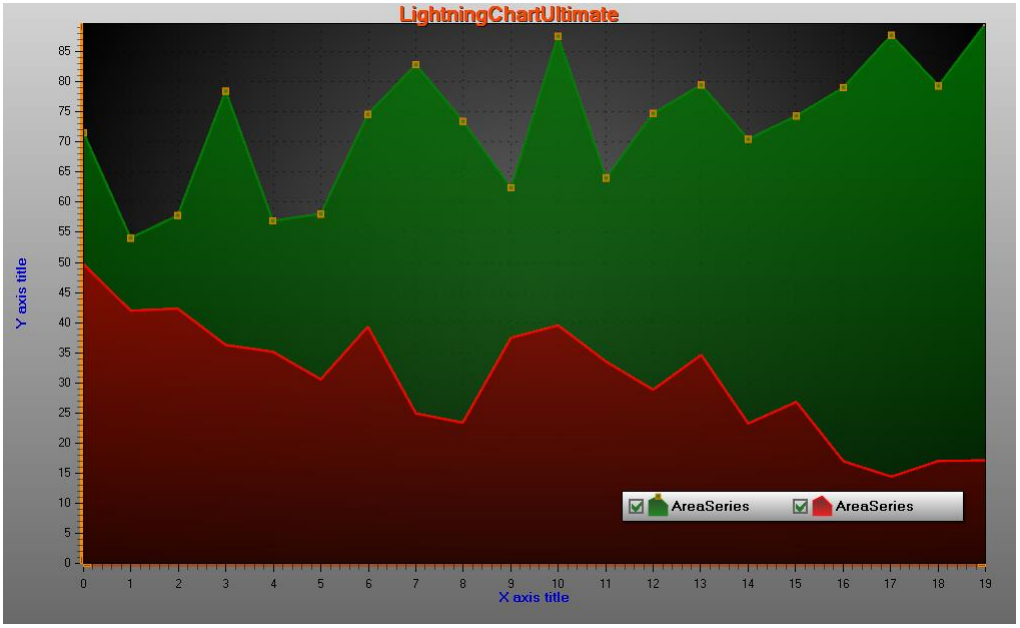


Figure 6-41. Two area series. BaseValue = 0 in both. Points are visible here only in first series.

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***.

6.10.1 Adding data

You must add the data values in code. The data must be given in ascending order by X values.

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;
```

6.11 Bars

Bar series allows displaying data in horizontal or vertical bars.

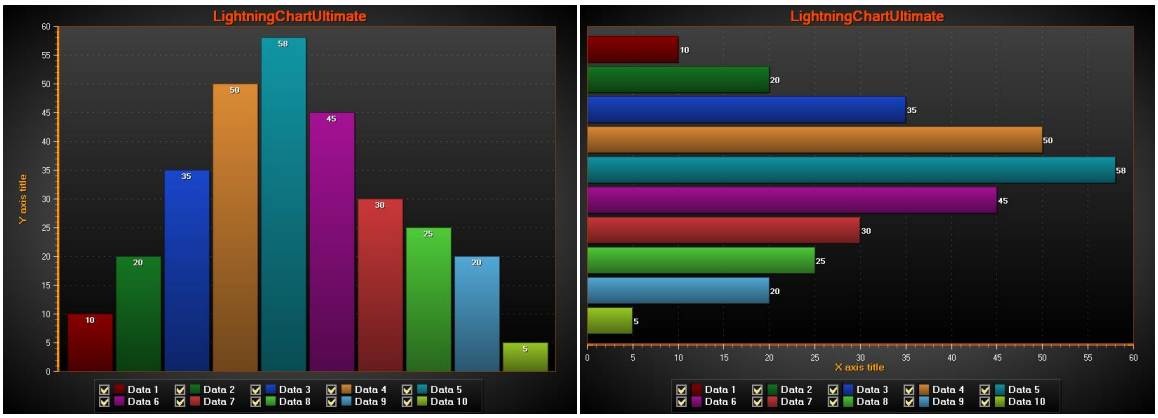


Figure 6-42. Bars series, vertical and horizontal.

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 parameterVarious 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 selects 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 %.

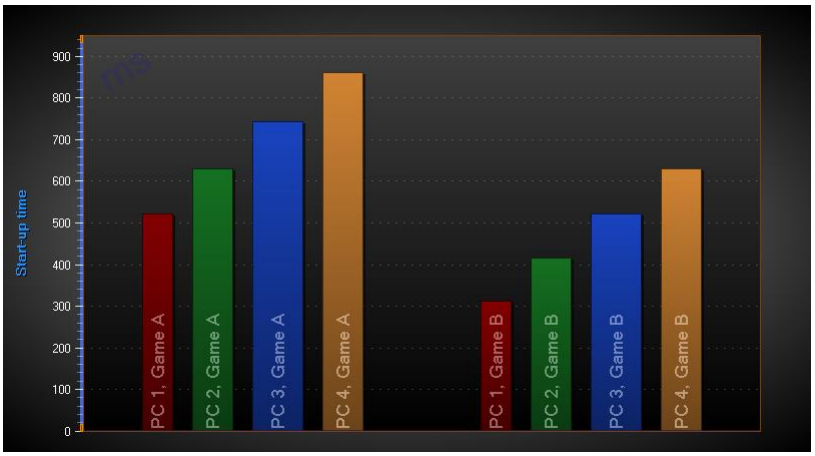


Figure 6-43. Bars series Grouping = ByIndex, Stacking = None.

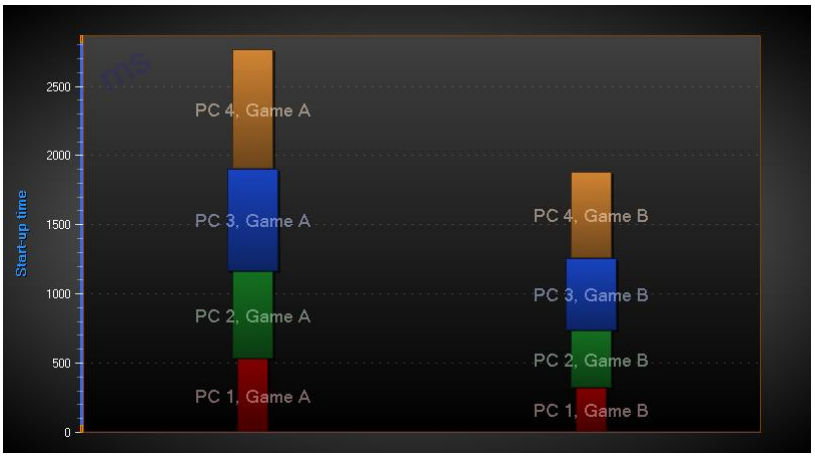


Figure 6-44. Bars series Grouping = ByIndex, Stacking = Stack.

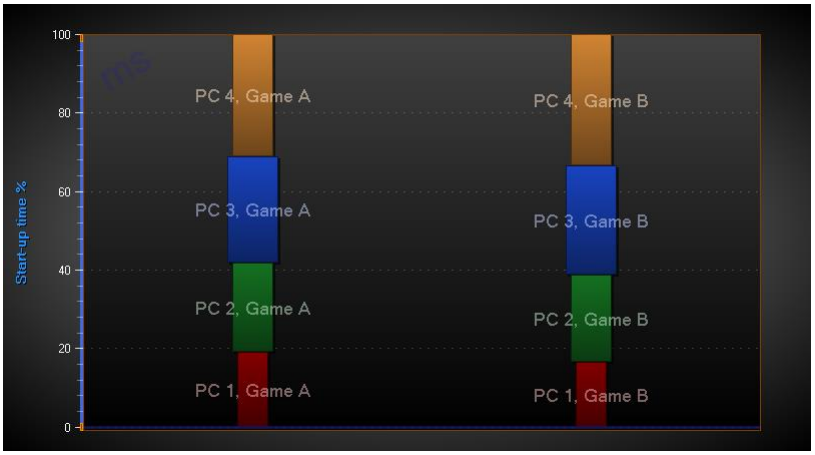


Figure 6-45. Bars series Grouping = ByIndex, Stacking = StackStretchToSum. StackSum = 100.

6.12 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 6-46. One StockSeries with Style = CandleStick. Light blue line is a PointLineSeries in the background, going through all Close values.

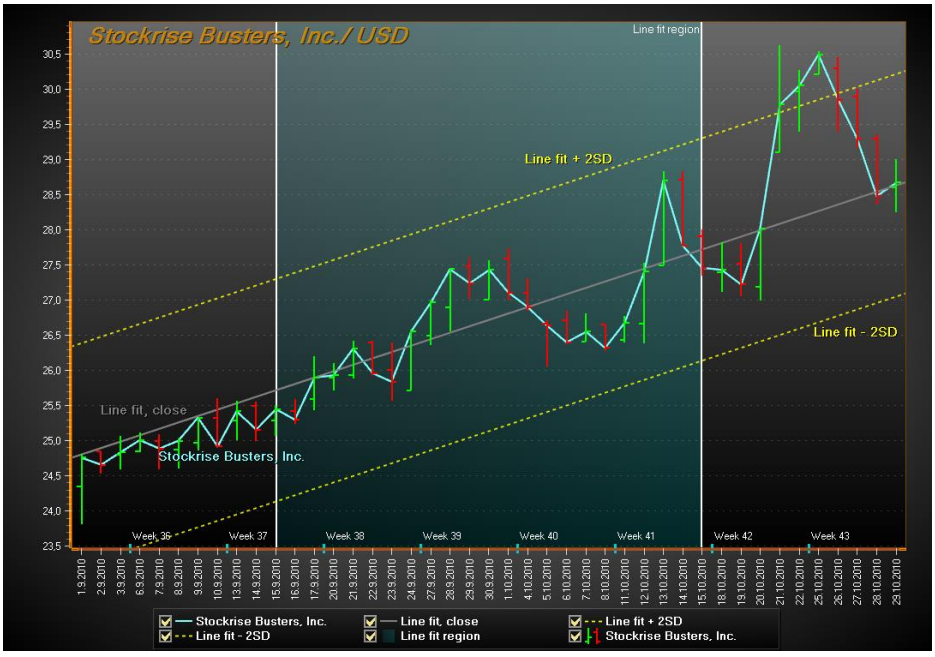


Figure 6-47. One StockSeries with Style = Bars. Line series are used for showing linear regression fit, and offset of that line (2 * standard deviation). A band is used for selecting a date range for line fit.

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. ***OptimizedCandleStick*** has only limited set of fill effects available.

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

6.12.1 Setting data to StockSeries

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

Date	DateTime value (year, month, day)
Open	The opening value of the day
Close	Close value of the day
Low	Lowest value during day
High	Highest value during day
Transaction	The total trading sum
Volume	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;
```

6.12.2 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);
```

6.12.3 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 6-48. CustomStockDataAppearance used to highlight specific data items with greater width and brighter, gradient color.

6.13 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.

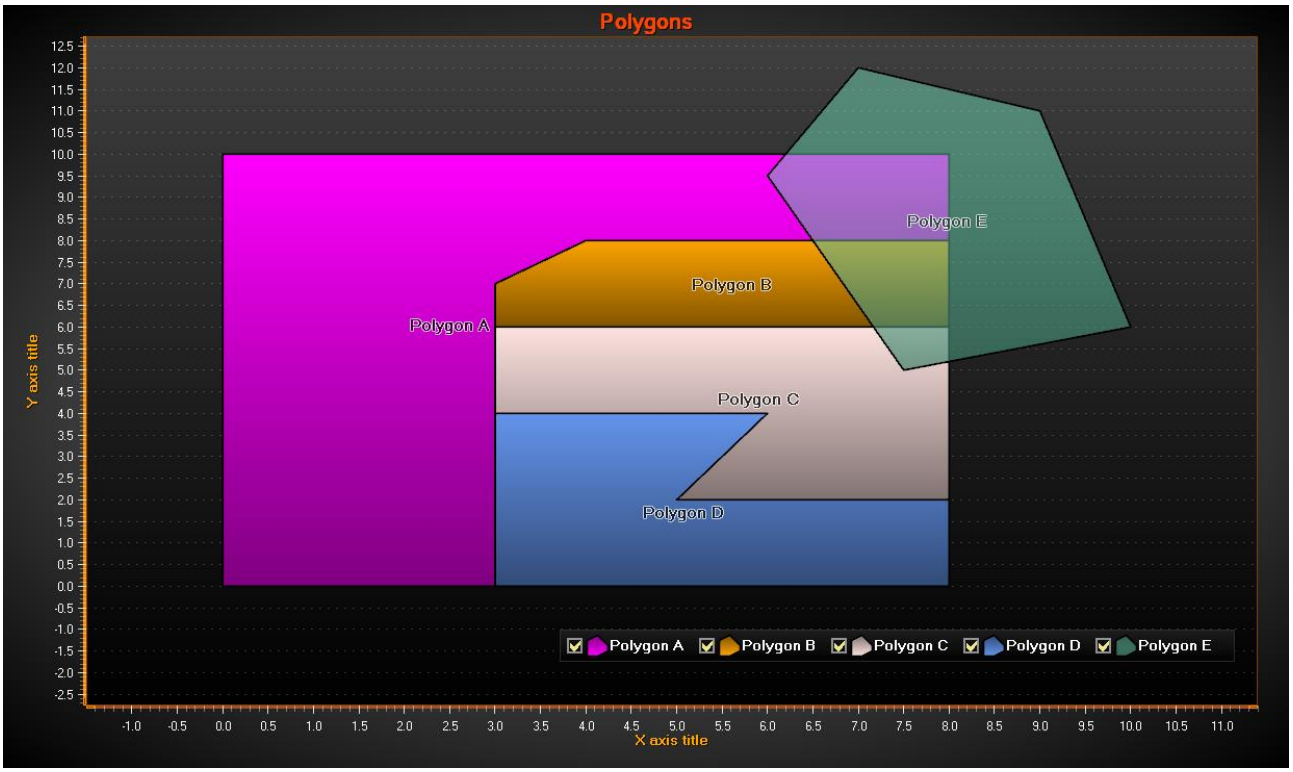


Figure 6-49. Several polygons.

6.13.1 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)};
```

6.14 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.

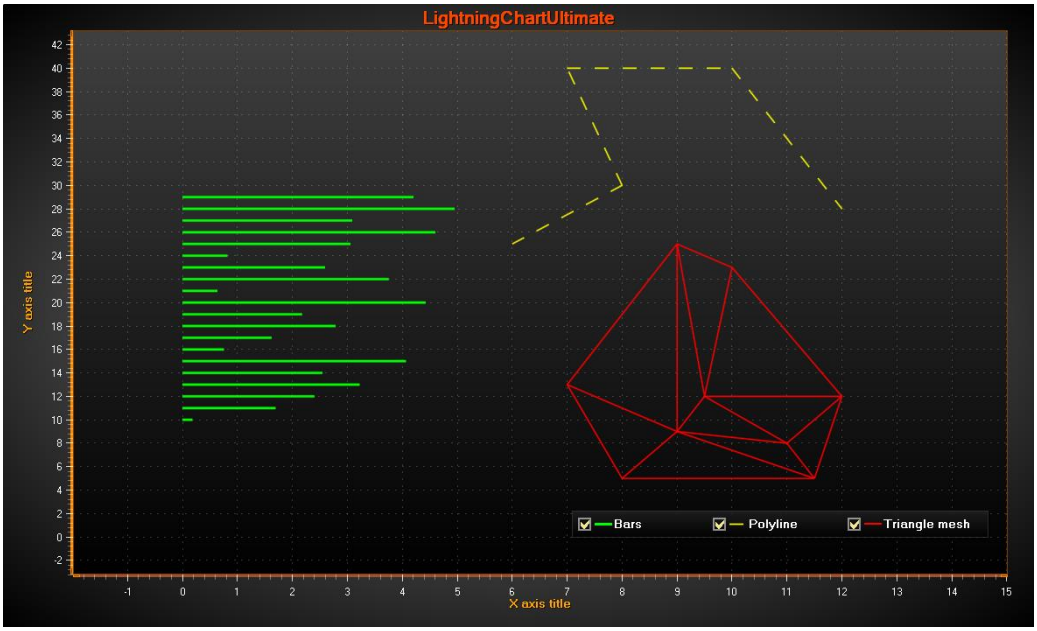


Figure 6-50. Three LineCollections in use. Green acts as very rapidly rendering bars, yellow as a polyline, and red as an arbitrary triangle wireframe mesh.

6.14.1 Setting data to a LineCollection

SegmentLine structure consists of four fields:

AX Start point, X
AY Start point, Y
BX End point, X
BY 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) };
```

6.15 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	**ColorLine**
Data	**IntensityPoint[,] Array**
DisableDragToAnotherAxis	True
FastContourZoneRange	1
Fill	Paletted
FullInterpolation	False
IncludeInAutoFit	True
InitialValue	0
LegendBoxUnits	Units
LegendBoxValuesFormat	0.0
LegendBoxValueType	Number
LimitYToStackSegment	**False**
MouseHighlight	**None**
MouseInteraction	**False**
Optimization	DynamicData
PixelRendering	False
RangeMaxX	**90**
RangeMaxY	**90**
RangeMinX	**10**
RangeMinY	**10**
ShowInLegendBox	True
ShowNodes	False
SizeX	**200**
SizeY	**200**
Stencil	
Title	**SeriesTitle: Data surface**
ToneColor	Black
TraceMouseCell	**True**
ValueRangePalette	
Visible	True
WireframeLineStyle	
WireframeType	None

Figure 6-51. IntensityGridSeries properties.

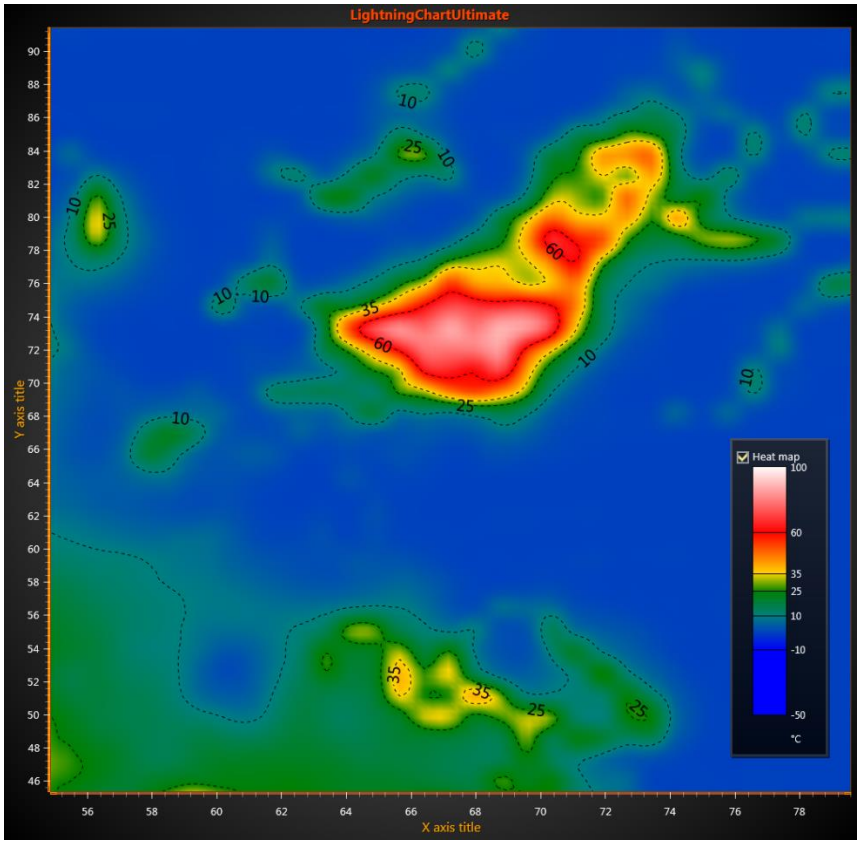


Figure 6-52. IntensityGrid series showing a heat map presentation. Legend box shows the value-range color palette.

The data is stored in Data property, as two-dimensional array. Each array is 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***.

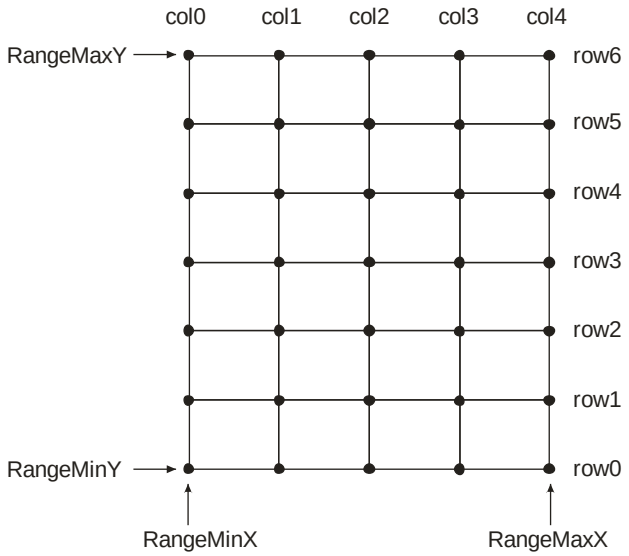


Figure 6-53. IntensityGridSeries nodes. SizeX = 5, SizeY = 7.

Node distances are automatically calculated as

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

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

6.15.1 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 ís 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

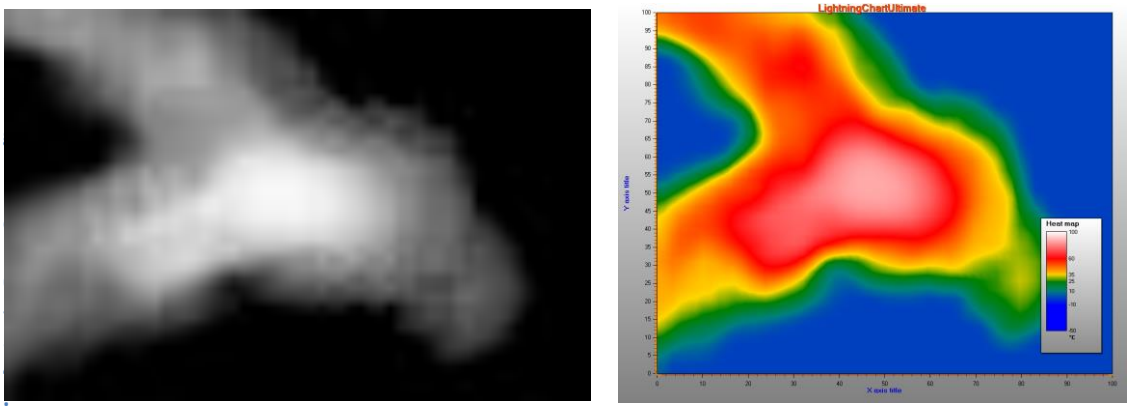
When the geometry of IntensityMesh doesn't change or SizeX or SizeY for IntensityGrid series doesn't change, and data is changing rapidly, it is most advantageous to use ***SetValuesData*** method. It accepts Double[][] format values. With this kind of data array, scrolling or re-ordering rows or columns is quick. Especially this combined to ***PixelRendering*** property (see 6.15.4) 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 6.15.4) 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.

6.15.2 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.



Source bitmap and calculated intensity values data. Dark values stay low and bright values get higher values.

6.15.3 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 6.15.4.

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.

6.15.4 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.

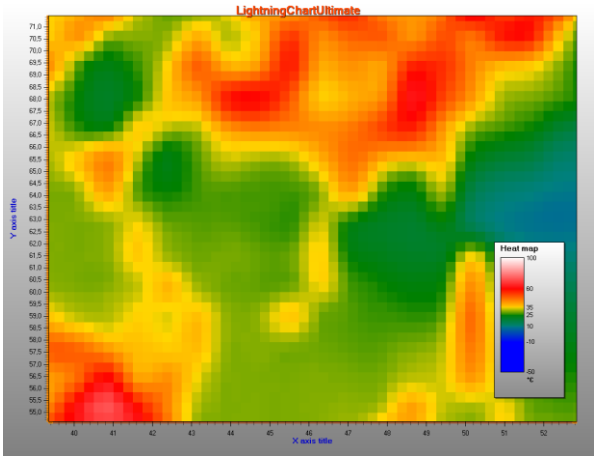


Figure 6-5. ContourLineType = ColorLine.

6.15.5 ValueRangePalette

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

- ***Fill*** (see section 6.15.3)
- ***Wireframe*** (see section 6.15.6)
- ***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.

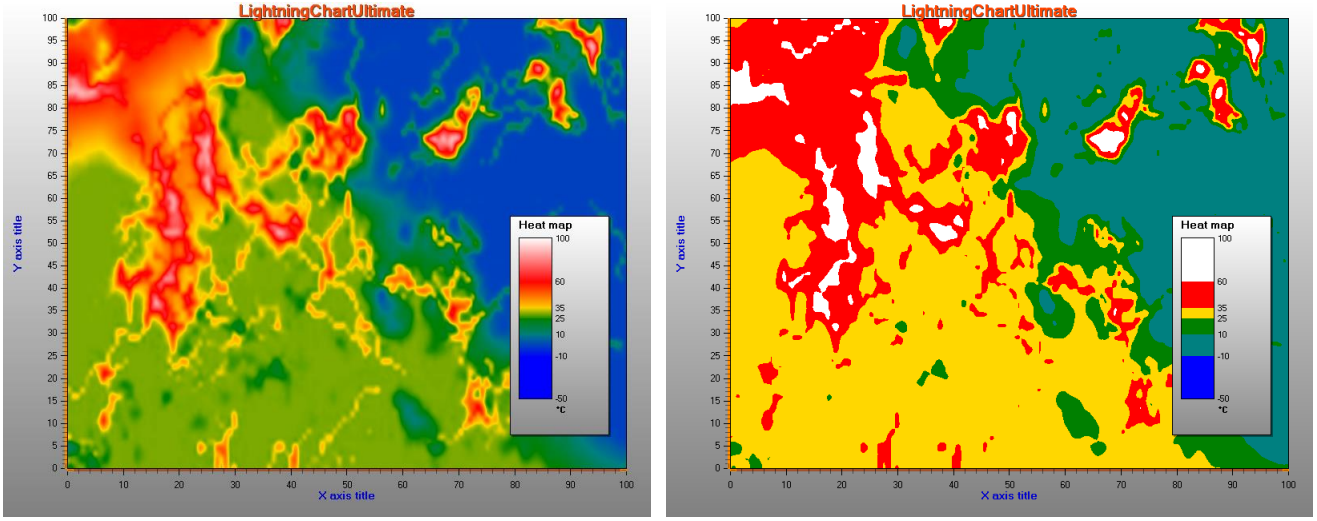


Figure 6-55. On the left, IntensityGridSeries Fill is set to Paletted and palette Type is set to Gradient. On the right, palette Type is set to Uniform.

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.

6.15.6 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 6.15.4)
- ***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***.

6.15.7 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 6.15.4)
- ***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.

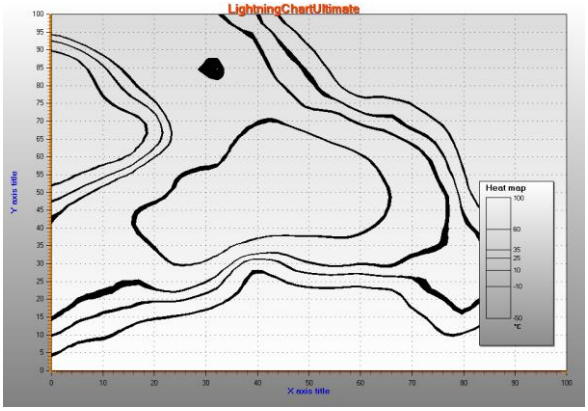


Figure 6-5. ContourLineType = FastColorZones.

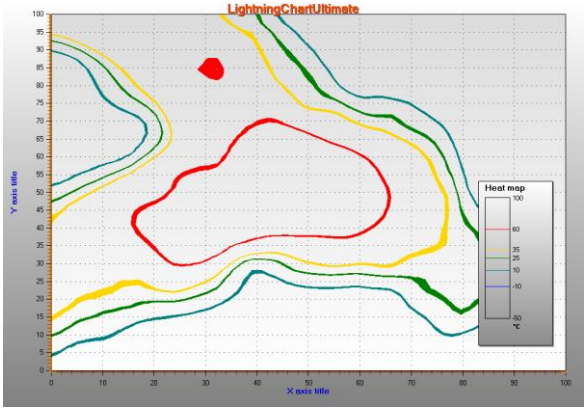


Figure 6-5. ContourLineType = FastPalettedZones.

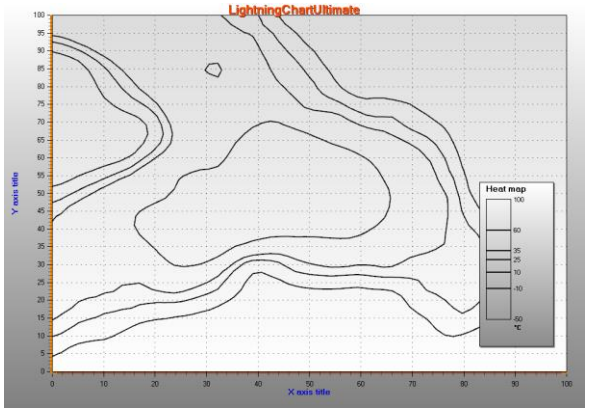


Figure 6-5. ContourLineType = ColorLine.

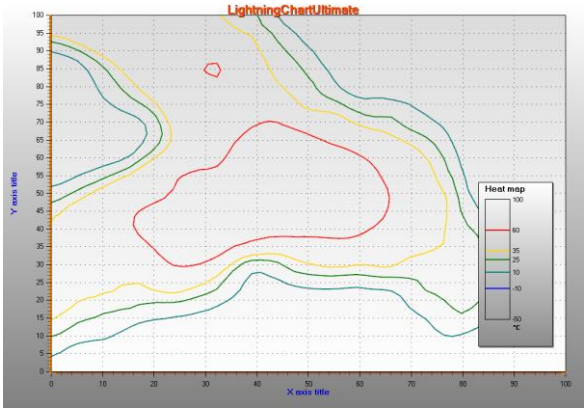


Figure 6-5. ContourLineType = PalettedLine.

6.15.8 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

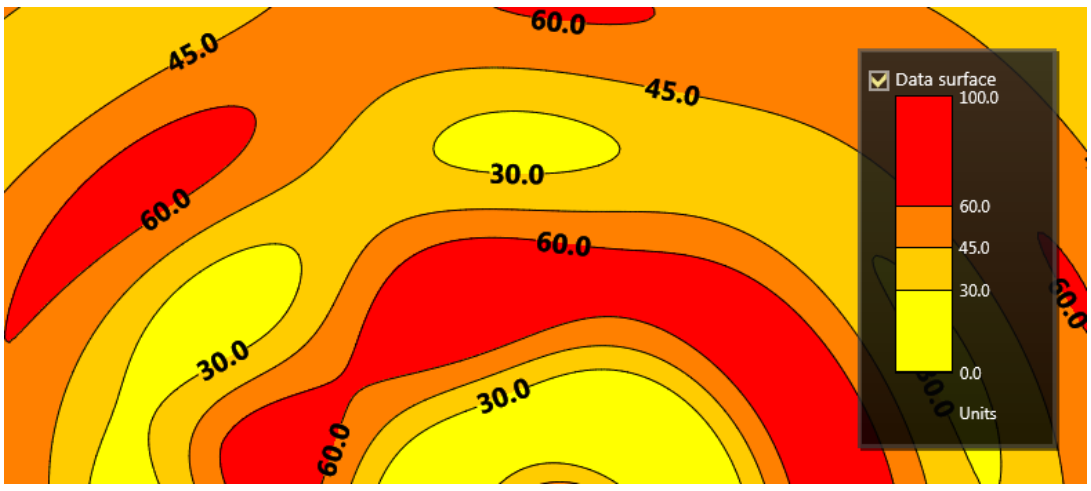


Figure 6-5. ContourLineLabels.Visible = true.

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

6.16 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.

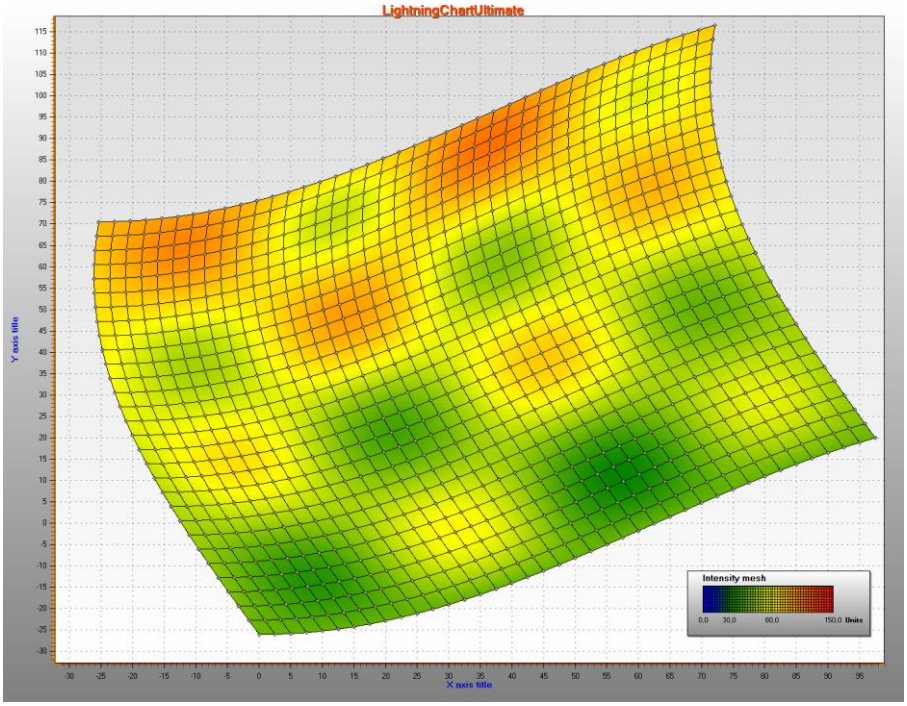


Figure 6-56. IntensityMeshSeries with freely positioned X and Y values for each node. WireframeType = Wireframe and ShowNodes = true.

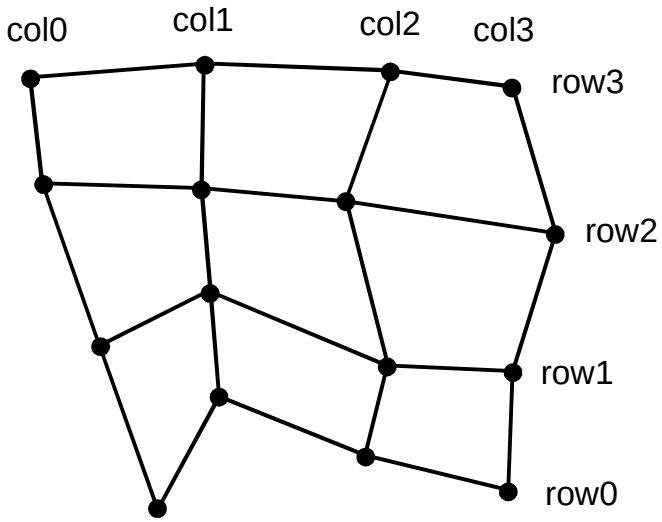


Figure 6-57. Intensity mesh nodes. SizeX = 4, SizeY =4.

6.16.1 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
```

6.16.2 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.

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
```

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
```

6.17 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).

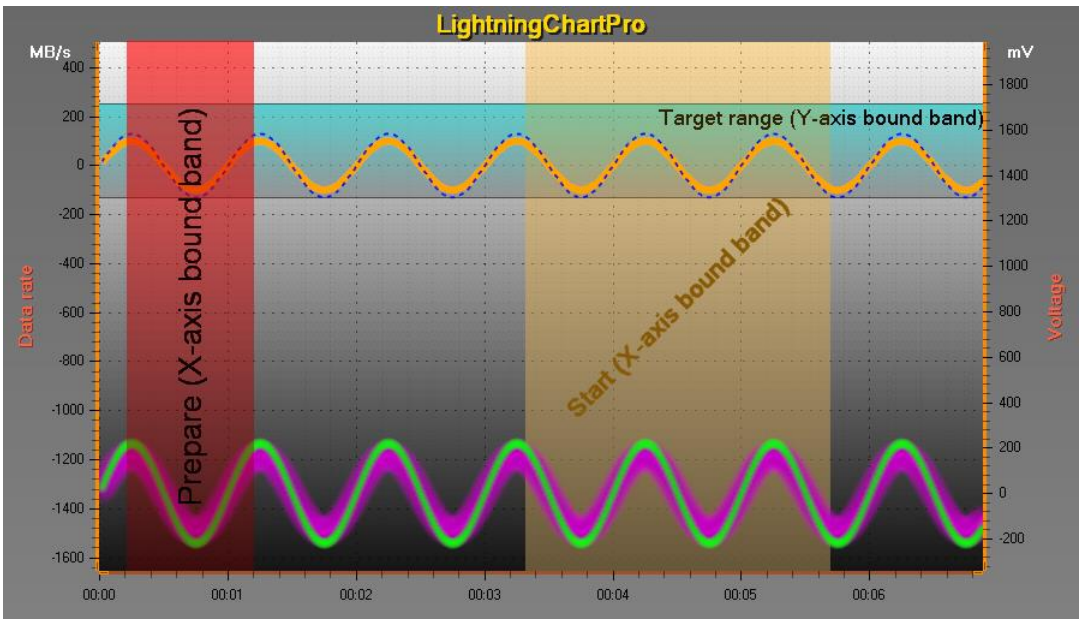


Figure 6-58. A couple of bands with line series

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***.

6.18 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.

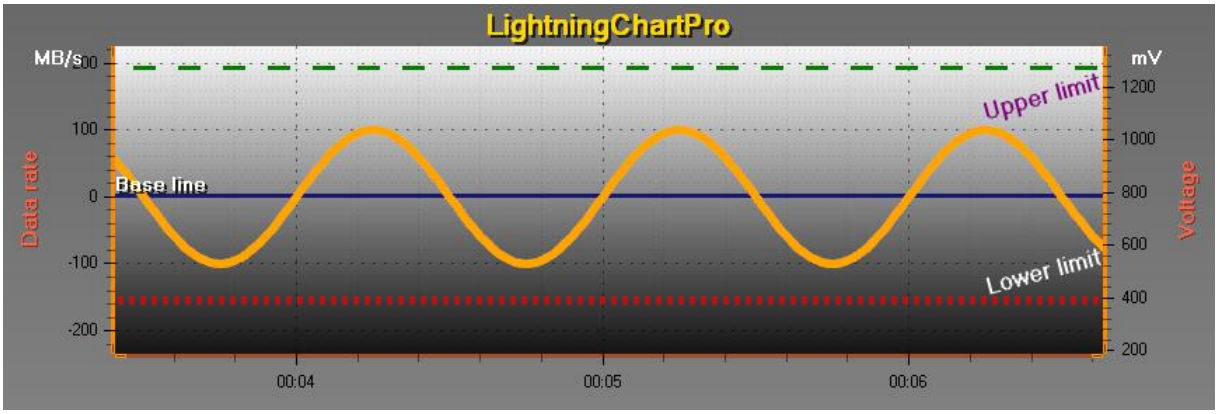


Figure 6-59 Some constant lines around a sine line series.

6.19 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.

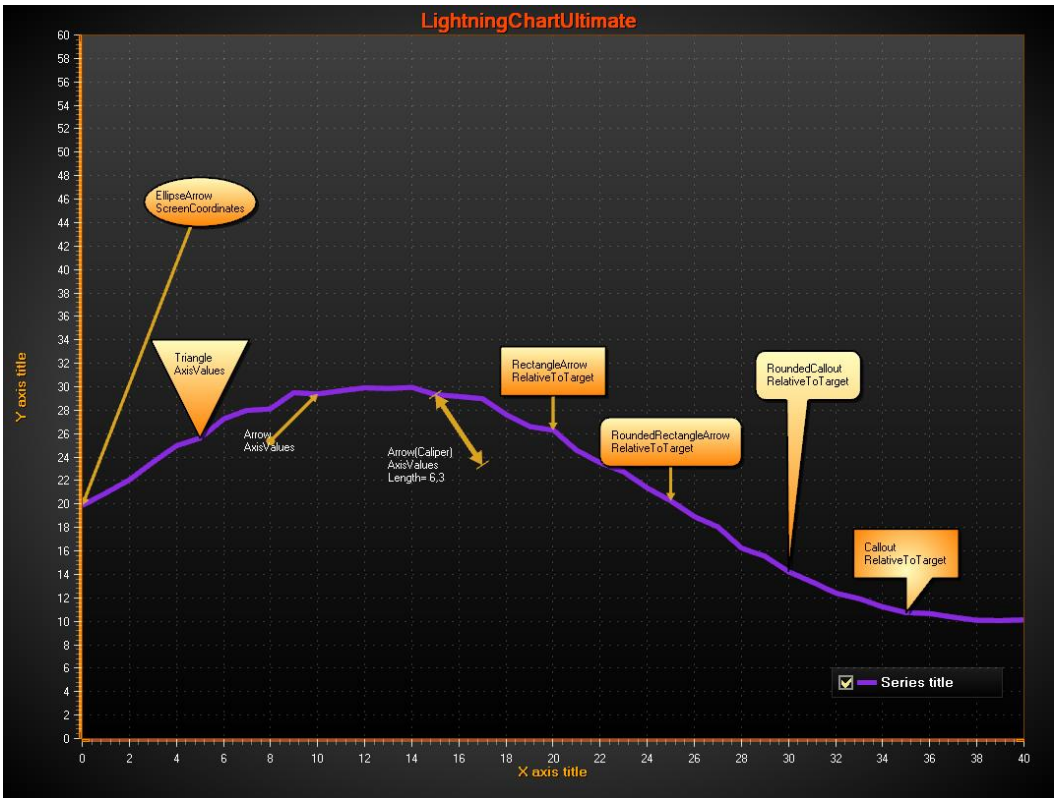


Figure 6-60 AnnotationXY objects with various styles, placed around a line series. Use *Style* property to select the shape.

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.

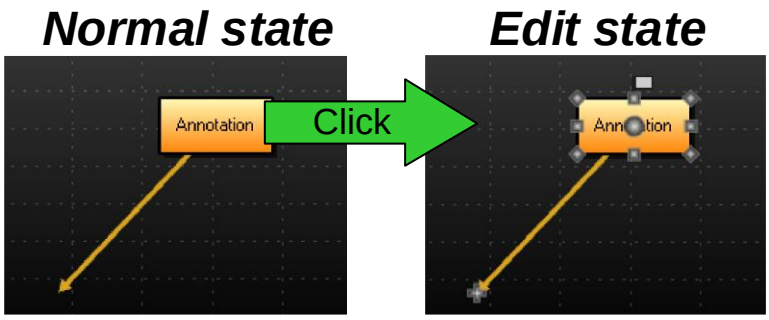


Figure 6-61 Click the annotation to enter the editing state. Click another annotation or graph background to leave the edit state.

6.19.1 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:

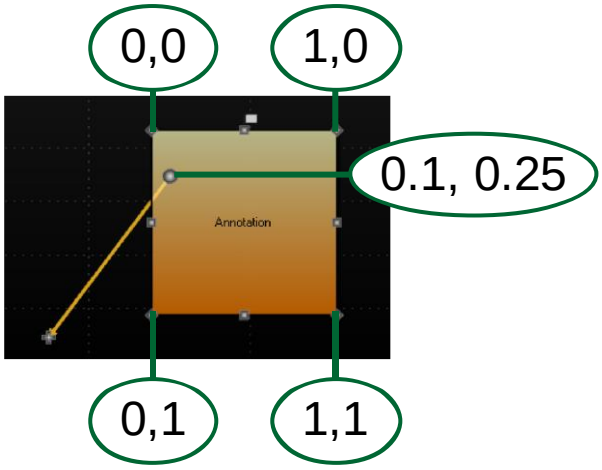


Figure 6-62 Anchor values explained. Current Anchor.X = 0.1 and Anchor.Y = 0.25. When the anchor values are between 0…1, the arrow start point is inside the text area.

6.19.2 Using mouse to move, rotate and resize

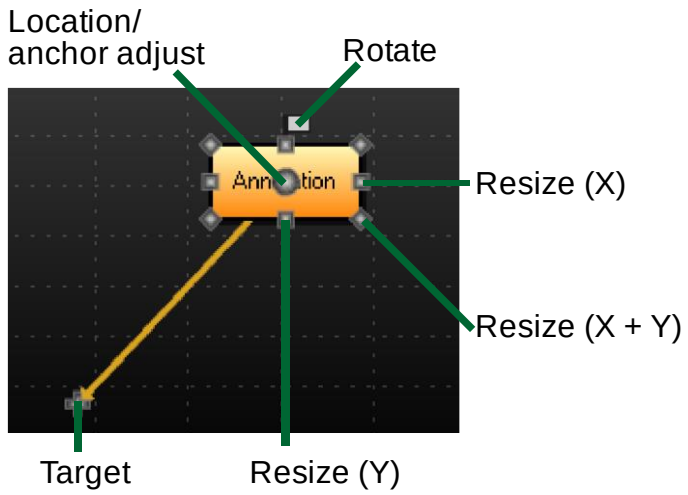


Figure 6-63 Annotation mouse-interactive nodes.

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.

6.19.3 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.

6.19.4 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***.

6.19.5 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.

6.19.6 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.

6.19.7 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

6.19.8 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.

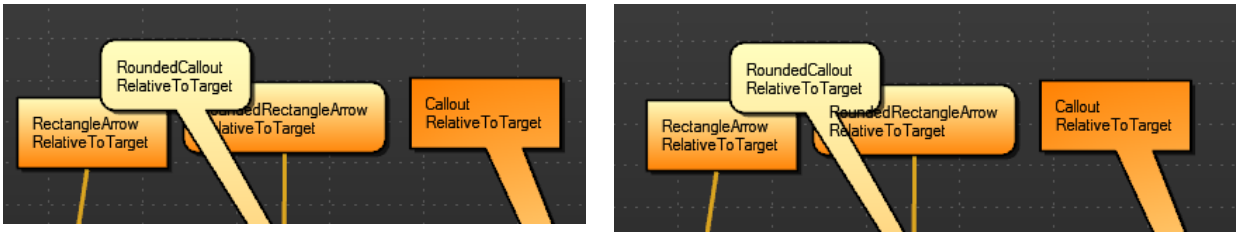


Figure 6-64 On the left LayerGrouping = False. On the right, LayerGrouping = True, Z order of texts is lost.

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

6.19.9 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.

6.20 Legend box

LegendBox	**LegendBoxXY**
AllowMouseResize	True
AutoSize	**False**
BorderColor	**40, 255, 255, 255**
BorderWidth	**3**
Categorization	**YAxes**
CategoryColor	White
CategoryFont	**Segoe UI, 10pt, style=Bo**
CheckBoxColor	**140, 255, 255, 255**
CheckMarkColor	Khaki
Fill	
Height	**108**
HighlightSeriesOnTitle	True
HighlightSeriesTitleColor	Yellow
IntensityScales	
Layout	**Vertical**
MouseHighlight	Simple
MouseInteraction	True
MoveByMouse	True
MoveFromSeriesTitle	True
Offset	
Position	**Manual**
ScrollBarVisibility	Both
SeriesTitleColor	White
SeriesTitleFont	**Segoe UI, 10pt**
Shadow	
ShowCheckboxes	True
ShowIcons	True
UnitsColor	White
UnitsFont	**Segoe UI, 9pt**
UseSeriesTitlesColors	False
ValueLabelColor	White
ValueLabelFont	**Segoe UI, 9pt**
Visible	True
Width	**226**

Figure 6-65 Extensive LegendBox property tree.

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

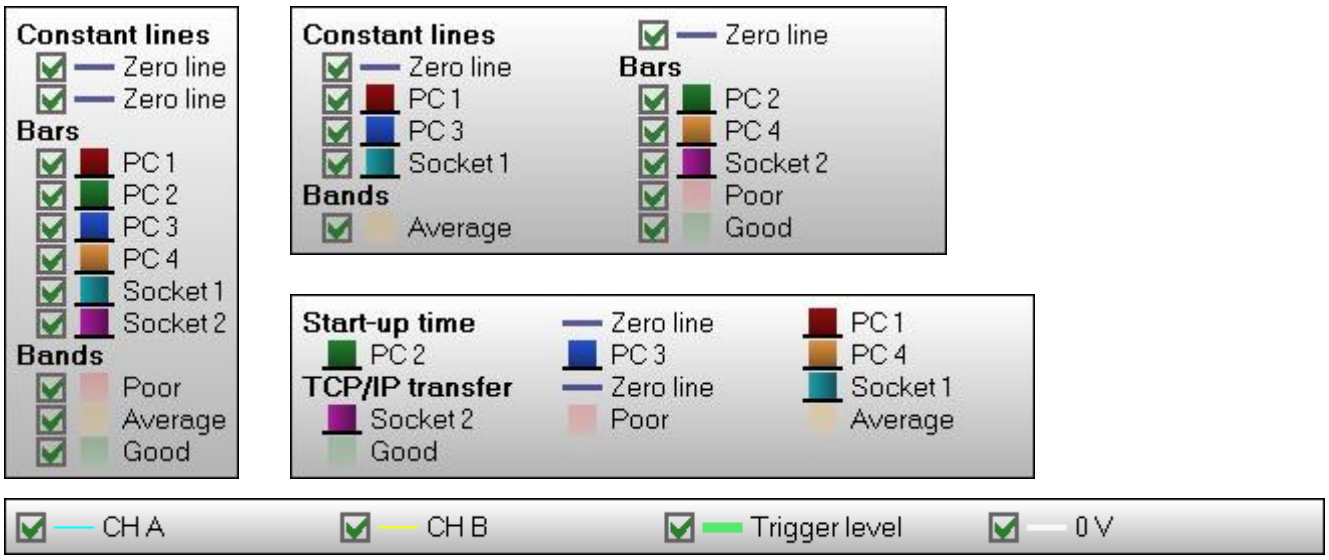
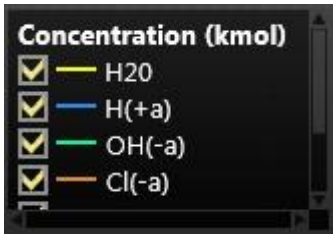


Figure 6-66. Some legend box layout variations:
- Categorization by series type, vertical layout
- Categorization by series type, horizontal row span layout
- Categorization by Y axes, horizontal row span, no checkboxes
- No categorization, horizontal layout

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



6.21 Zooming and panning

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

ZoomPanOptions	
AltEnabled	True
AspectRatioOptions	
AspectRatio	Off
ManualAspectRatioWH	**2**
XAxisIndex	0
YAxisIndex	0
AutoYFit	
Enabled	False
MarginPercents	5
TargetAllYAxes	**False**
Thorough	True
UpdateInterval	100
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
RectangleZoomingThreshold	
X	**4**
Y	**4**
RectangleZoomLimitInsideGraph	False
RectangleZoomUnitsLinkYAxes	False
RightMouseButtonAction	Pan
RightToLeftZoomAction	FitView
ShiftEnabled	True
ViewFitYMarginPixels	0
ZoomFactor	2
ZoomOutRectFill	
ZoomOutRectLine	
ZoomRectFill	
ZoomRectLine	

Figure 6-67 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.

6.21.1 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.

6.21.2 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.

6.21.3 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***.

6.21.4 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***.

6.21.5 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:

FitView: 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 6.2.5.

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

6.21.6 Zooming with mouse button

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.

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. If a ***YAxesLayout*** is set to **stacked** setting, the view is zoomed only in X dimension. The dimensions are selected by ***RectangleZoomDirection*** property.

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

Configuring zoom out rectangle

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

6.21.7 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*****.**

6.21.8 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 is mouse 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 is mouse over a Y axis. **Note**, applies to other axes only when ***YAxisLayout*** = ***Layered***.

6.21.9 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.

6.21.10 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***.

6.21.11 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. ***FitView()*** method fits does "Zoom to fit" operation.

6.21.12 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.

6.21.13 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.

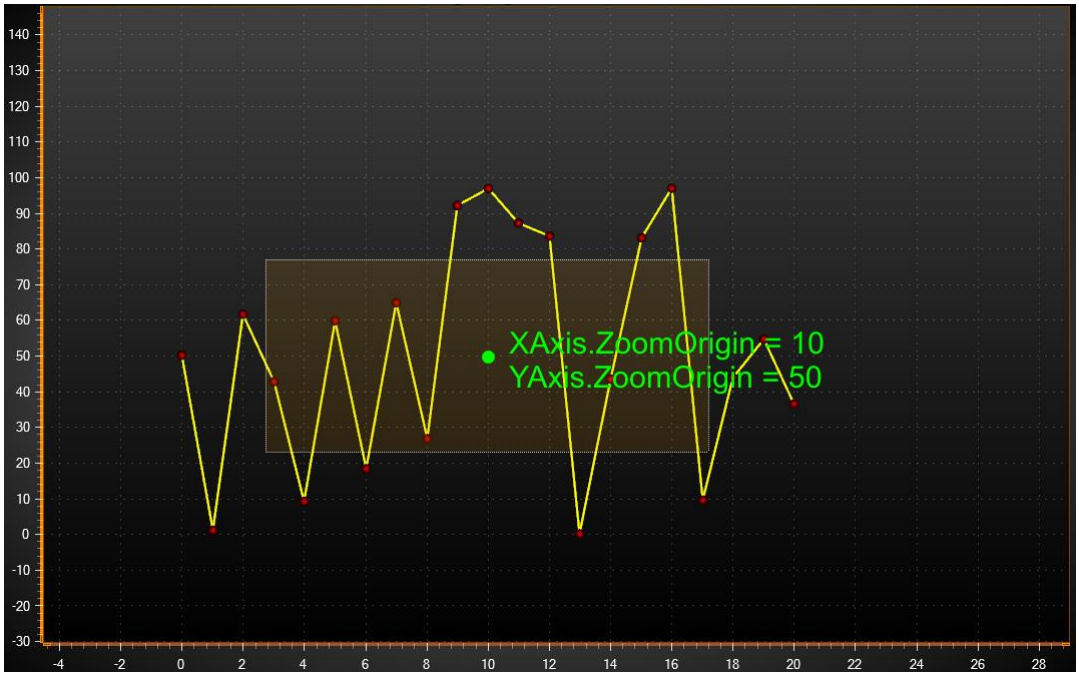


Figure 6-68 ZoomPanOptions.RectangleZoomAboutOrigin enabled. ViewXY.XAxes[0].ZoomOrigin = 50 and ViewXY.YAxes[0].ZoomOrigin = 50.

6.21.14 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.

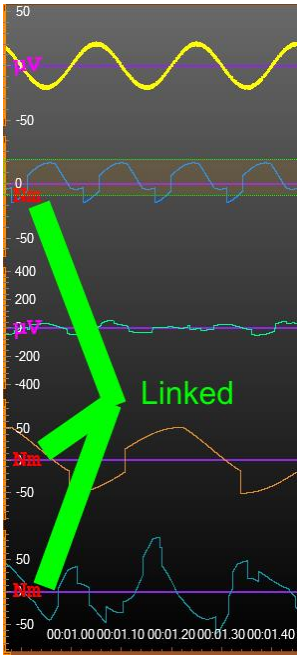


Figure 6-69 Stacked view with 5 Y axes. When rectangle zoom is applied over a graph segment, the Y axes of it get zoomed, and the new Y axis range is copied to all Y axes having same Units.Text.

6.21.15 Automatic Y fit

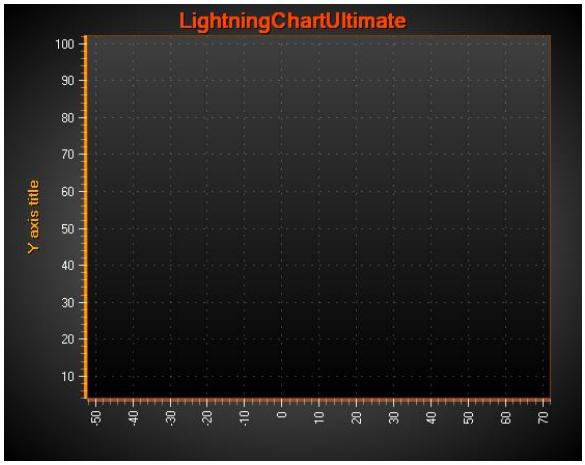
Use ***AutoYFit*** property to control the automatic Y axis adjustment. Automatic Y fit can be used to adjust the Y axis ranges to show all the data in the chart, in visible X axis range. Automatic Y fit is especially intended for real-time monitoring purposes. The fit is applied in time intervals, use ***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.

6.21.16 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 6.24), ***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.

6.21.17 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

6.22 DataBreaking by NaN or other value

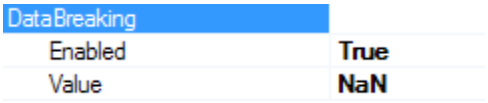


Figure 6-70 DataBreaking options in series that support it.

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.

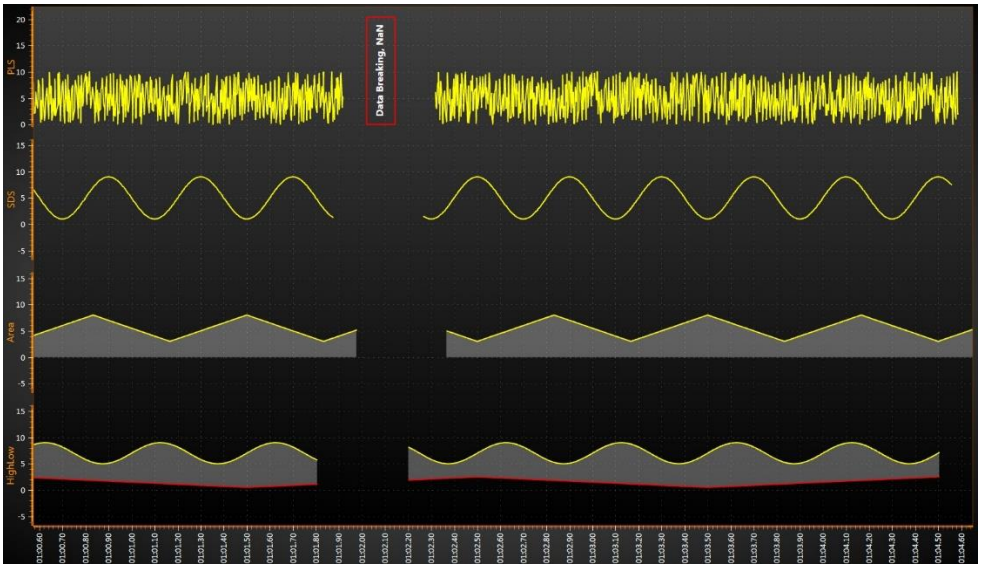
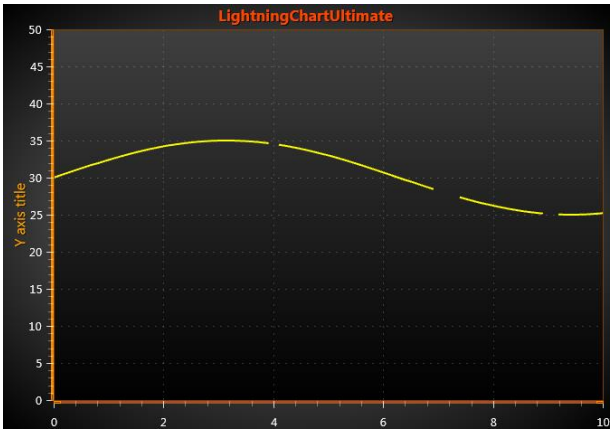


Figure 6-71 DataBreaking in use for PointLineSeries, SampleDataSeries, AreaSeries and HighLowSeries.

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 6.23.

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);
```

6.23 ClipAreas

Like ***DataBreaking*** (see 6.22), ***ClipAreas*** can be used to prevent part of the series data from rendering. They can be used to filter out bad data ranges, out-of-range data by Y, etc.

ViewXY's series have ***SetClipAreas*** method for setting or updating the clipping areas. It accepts an array of **ClipArea** structures. The ClipAreas array can be changed frequently, and performance stays good up to thousands of ClipAreas.

The ***ClipArea*** applies for the series that it has been assigned to. **Note**, this is a rendering-stage clipping and mouse operations will respond to series when placed over the ClipArea if there's actual data under it.

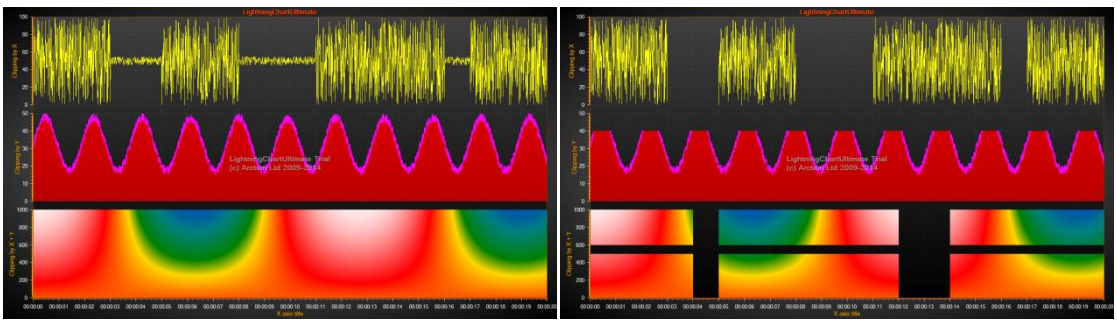


Figure 6-72 ClipAreas defined for 3 series. For PointLineSeries, AreaSeries, and IntensityGridSeries. On the left, the ClipAreas are not used. On the right, ClipAreas are enabled. For yellow PointLineSeries, X dimensional clipping areas have been defined to mask off low-amplitude data. For red AreaSeries, Y-dimensional ClipArea cuts too high-amplitude data from the top. For IntensityGridSeries, X- and Y-dimensional ClipAreas are used to prevent the series from rendering in specific areas.

Using ClipAreas is the performance-wise preferred way to break a line to several data segments instead of spawning hundreds of separate series during real time monitoring, or using DataBreaking feature.

6.24 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 *equirectangular* projection.

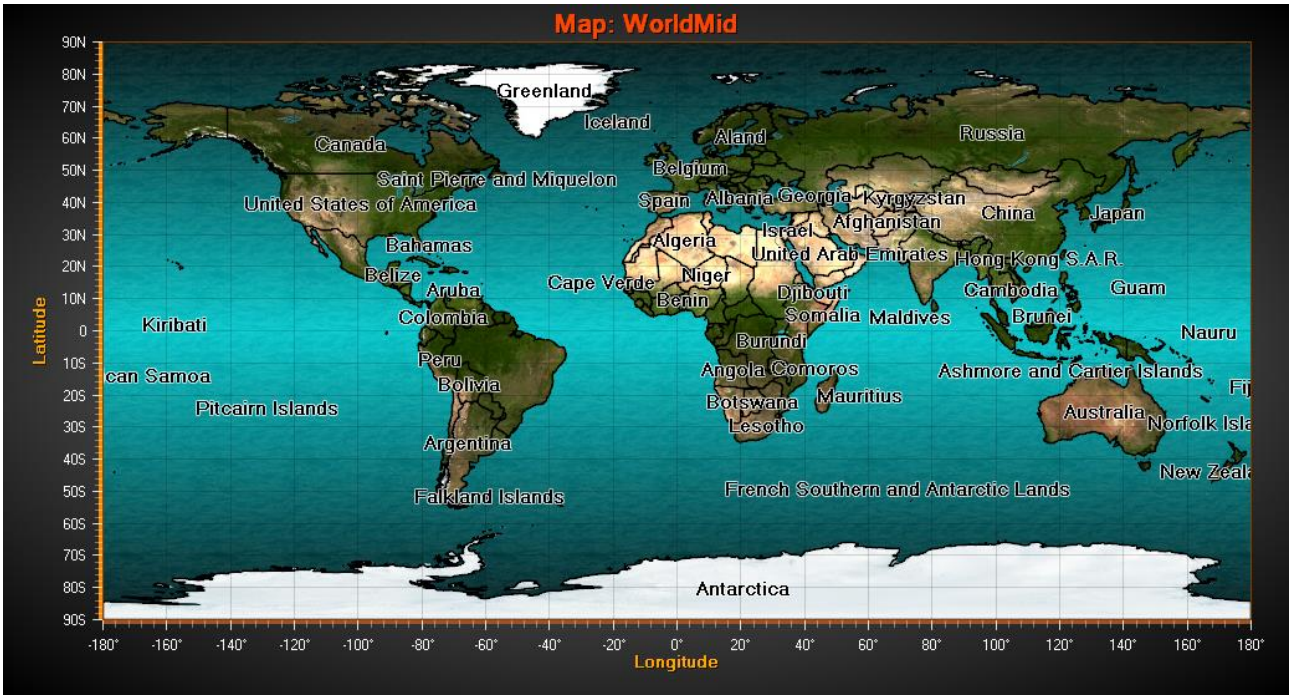


Figure 6-73 Equirectangular projection of the world. X range is from -180 to 180 degrees (180W to 180E), Y range from -90 to 90 degrees (90S to 90N). Polar areas get greatly stretched in this projection.

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.

6.25 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 6.2.3 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	**(Collection)**
⊞ CityOptions	
Description	Map of world in mid resolution.
FileName	**WorldMid**
⊞ LakeOptions	
⊞ LandOptions	
⊞ Layers	MapLayer[] Array
MouseHighlight	Simple
MouseInteraction	True
MouseOverMapItemLayer	1
Names	(Collection)
Optimization	**None**
⊞ OtherOptions	
OverlapLabels	False
Path	**..\..\..\Maps**
RenderIntensitySeriesBeforeLayerIndex	-1
⊞ RiverOptions	
⊞ RoadOptions	
SimpleHighlightColor	**100, 0, 255, 0**
TileCacheFolder	**c:\temp\map_cache**
TileLayers	**(Collection)**
Type	**WorldMid**
XAxisIndex	0
YAxisIndex	0

Figure 6-74 Maps properties and sub-properties. The whole tree is for vector maps, except TileLayers collection and TileCacheFolder, which is for tile maps.

6.25.1 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***.

Figure 6-75 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.

6.25.2 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.

6.25.3 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	**Cities**
Priority	**0**
Type	City
Visible	**True**
[1]	Arction.LightningChartUltimate.Maps.Reg
[2]	Arction.LightningChartUltimate.Maps.Reg
BorderDrawStyle	**Options**
Items	Region[] Array
Name	**Lakes**
Priority	**2**
RegionDrawStyle	**Options**
Type	Lake
Visible	**True**
[3]	Arction.LightningChartUltimate.Maps.Line
AutoAdjustLineWidth	**True**
Items	Line[] Array
LineDrawStyle	**Options**
LineWidthCoeff	**1**
Name	**Rivers2**
Priority	**3**
Type	River
Visible	**True**

Figure 6-76 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.M
AntialiasFill	**False**
Fill	Arction.LightningChartUltimate.F
Bitmap	Arction.LightningChartUltimate.B
Color	**OldLace**
GradientColor	**Moccasin**
GradientDirection	**270**
GradientFill	**Linear**
Style	**ColorOnly**
FillVisible	**True**
LabelStyle	Arction.LightningChartUltimate.M
Angle	**0**
Color	**Black**
Font	**Microsoft Sans Serif; 12pt; style**
Shadow	Arction.LightningChartUltimate.T
Visible	**True**
LineStyle	Arction.LightningChartUltimate.L
AntiAliasing	**Normal**
Color	**DimGray**
Pattern	**Solid**
PatternScale	**1**
Width	**1**
LineVisible	**True**

Figure 6-77 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

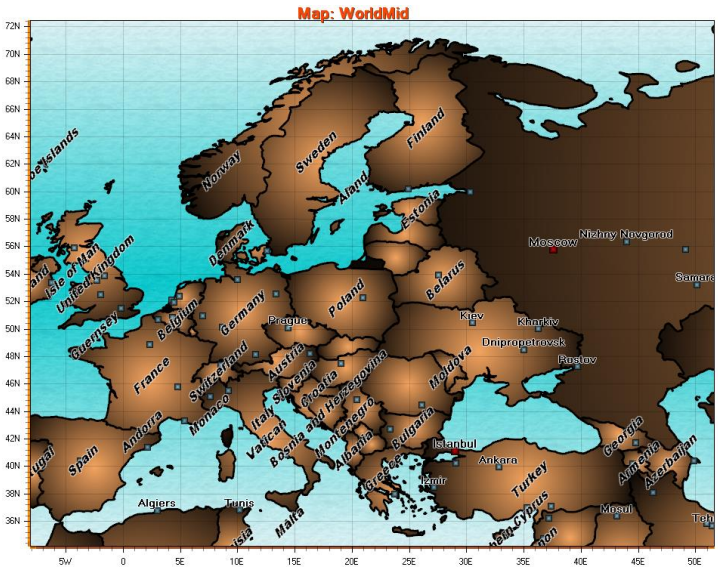


Figure 6-78 Modified LandOptions.

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 6-79 Setting layer border line and region fill styles to Invidual and editing region in Items collection.

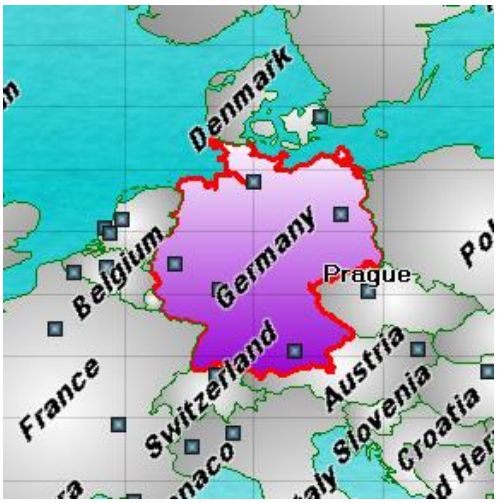


Figure 6-80 Germany region drawn with individual fill and border.

6.25.4 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);
              }
          }
      }
  }
```

6.25.5 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.

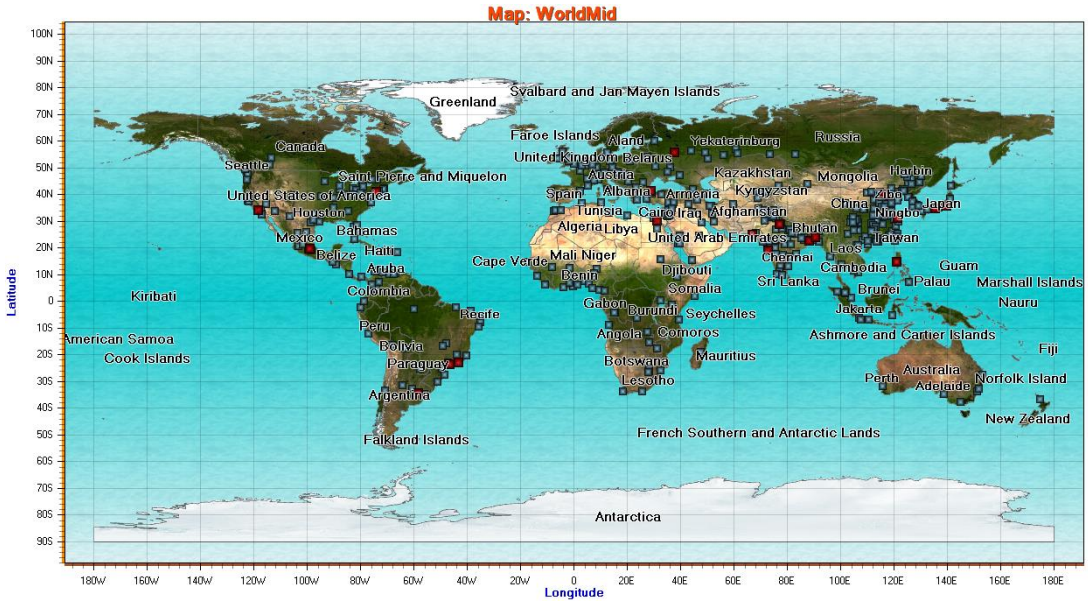


Figure 6-81 Map of the world. LandOptions.FillVisible is set to false, and one background image is set to latitude range of -90…90 and longitude range of -180…180. The map region borders and cities are shown.

6.25.6 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 6-82 Map of Europe, with a couple of FreeformPointLine series as routes. Flag markers are added to them as mouse-interactive waypoints.

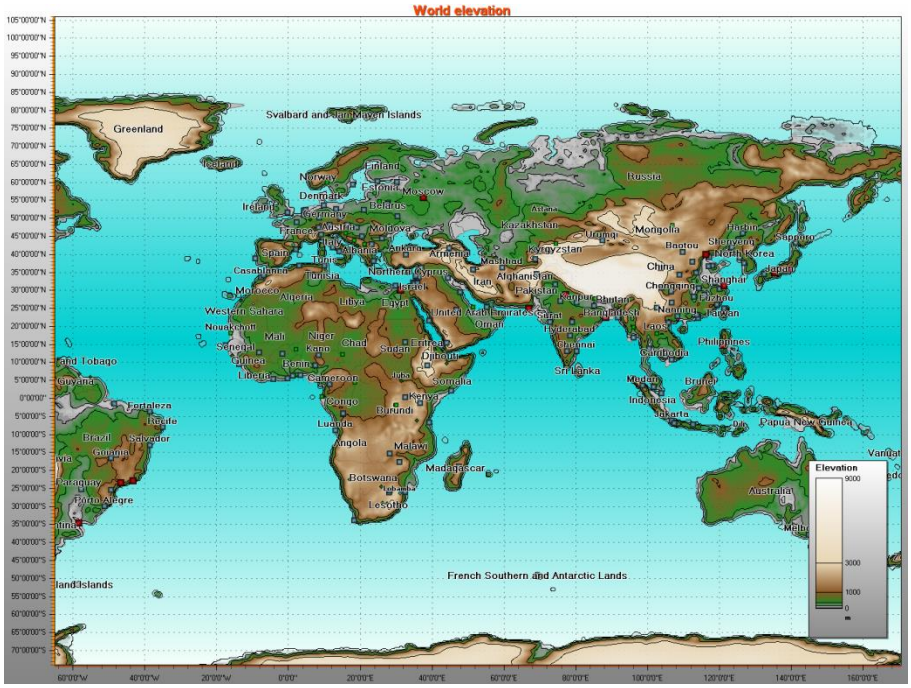


Figure 6-83 Map of the world, with IntensityGrid series presenting the elevation.

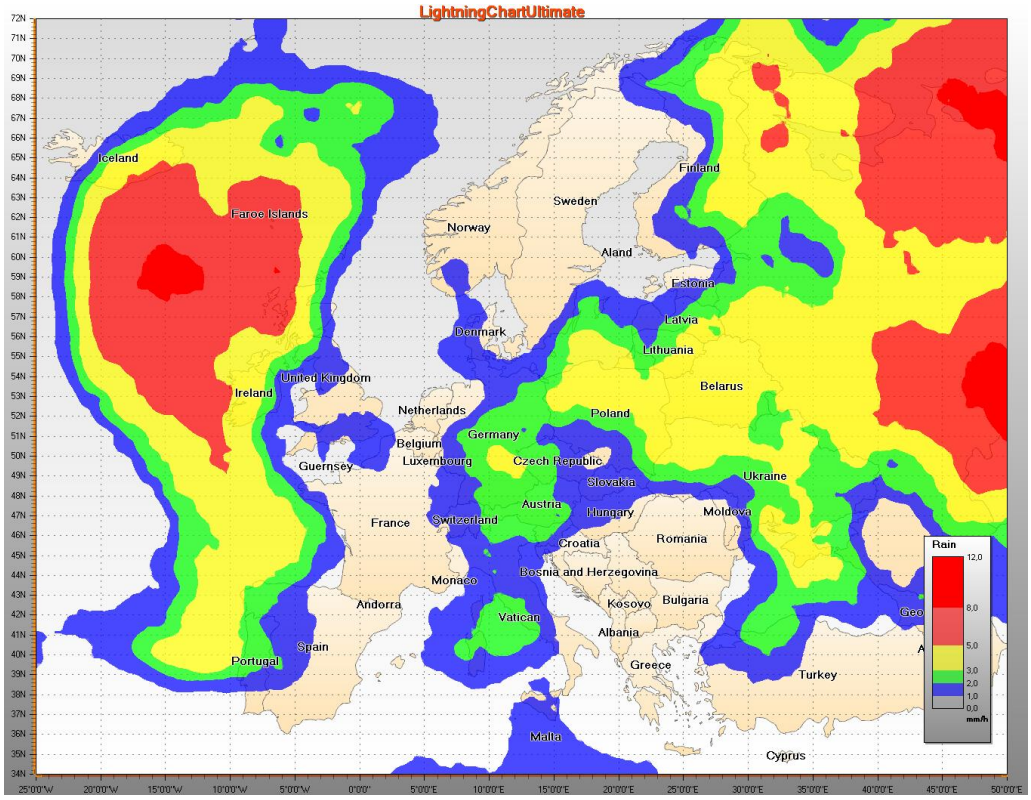


Figure 6-84 Weather radar data visualization with IntensityGrid series over the map of Europe.

6.25.7 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 Dialo.
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 prefiltering data based on the layer.

6.25.7.1 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_ConversionStateCha
nged);
```

6.25.7.2 Dialogs

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

6.25.7.2.1 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.

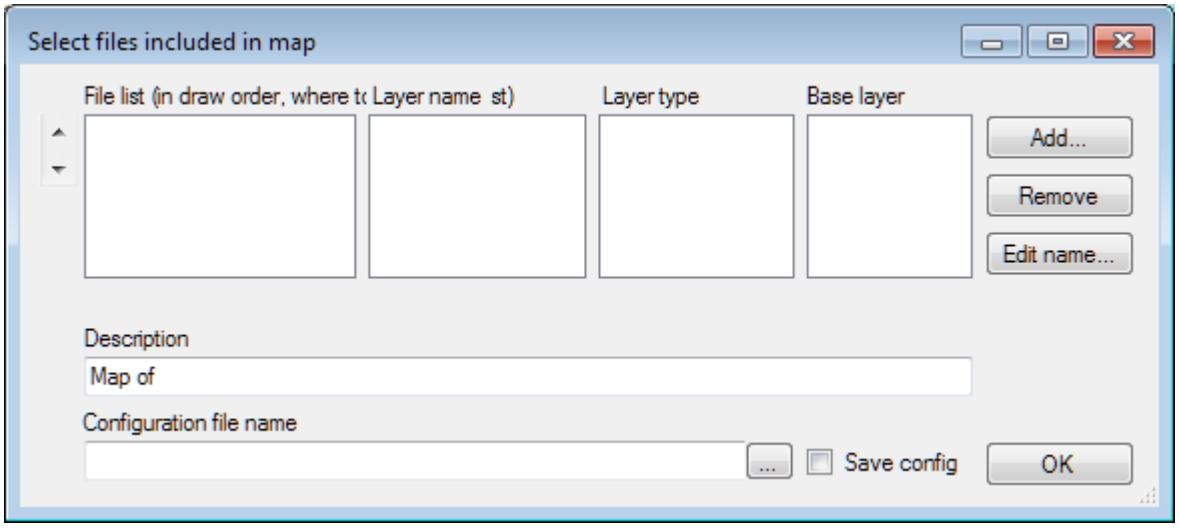


Figure 6-85. Source shape file selection dialog.

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 chaged 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).

6.25.7.2.2 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.

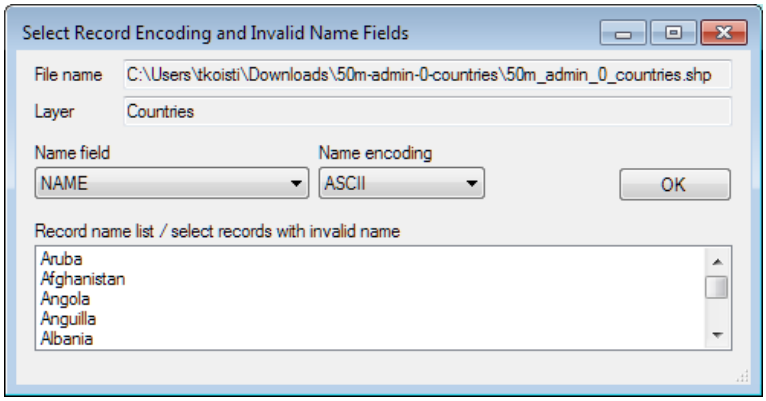


Figure 6-86. 'Record encoding' and 'Invalid name' fields selection dialog.

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).

6.25.7.2.3 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.

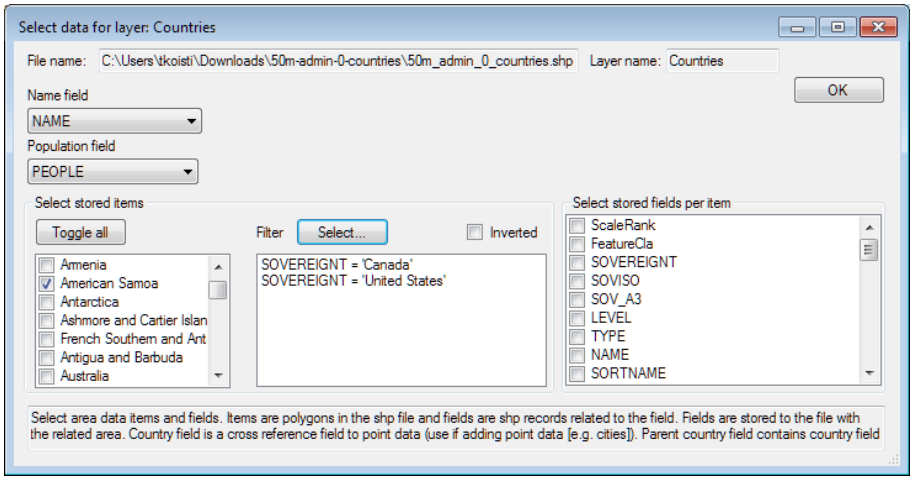


Figure 6-87. Layer data selection dialog.

The user interface items avilable 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.

6.25.7.2.4 Item filter

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

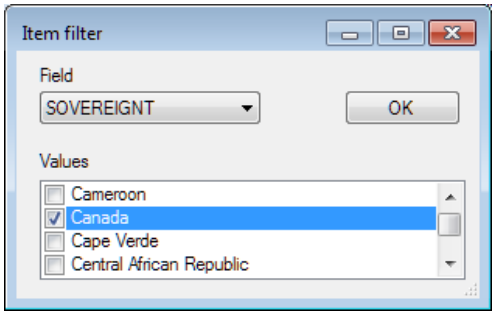


Figure 6-88. Item filter dialog.

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.

6.25.8 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.

6.26 Tile maps

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

- Here: Street maps, Satellite imagery

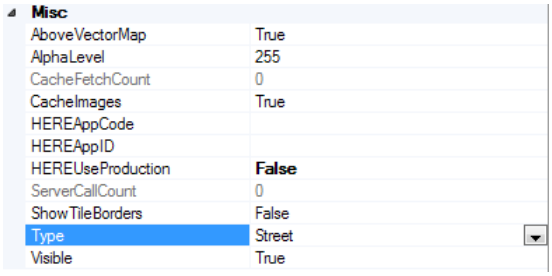


Figure 6-89. Properties of a TileLayer.

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 appeareance 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 6.25). 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.

6.26.1 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 6-90. TileLayer.Type = Satellite.

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

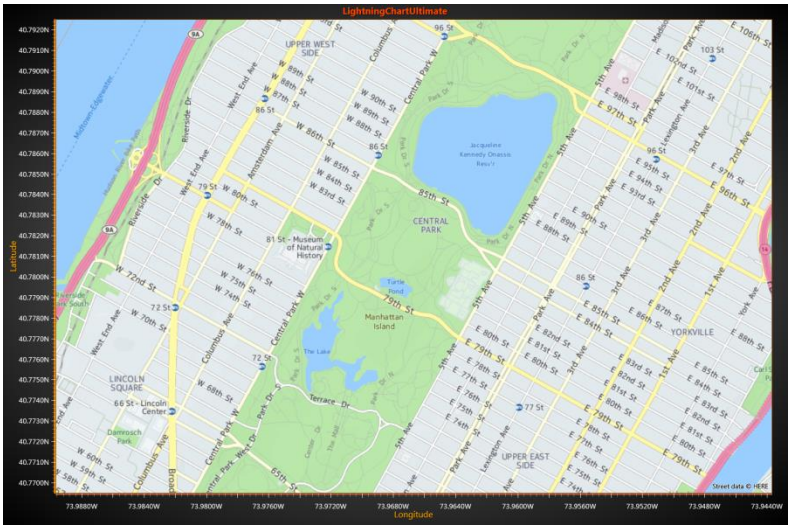


Figure 6-91. TileLayer.Type = Street.

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

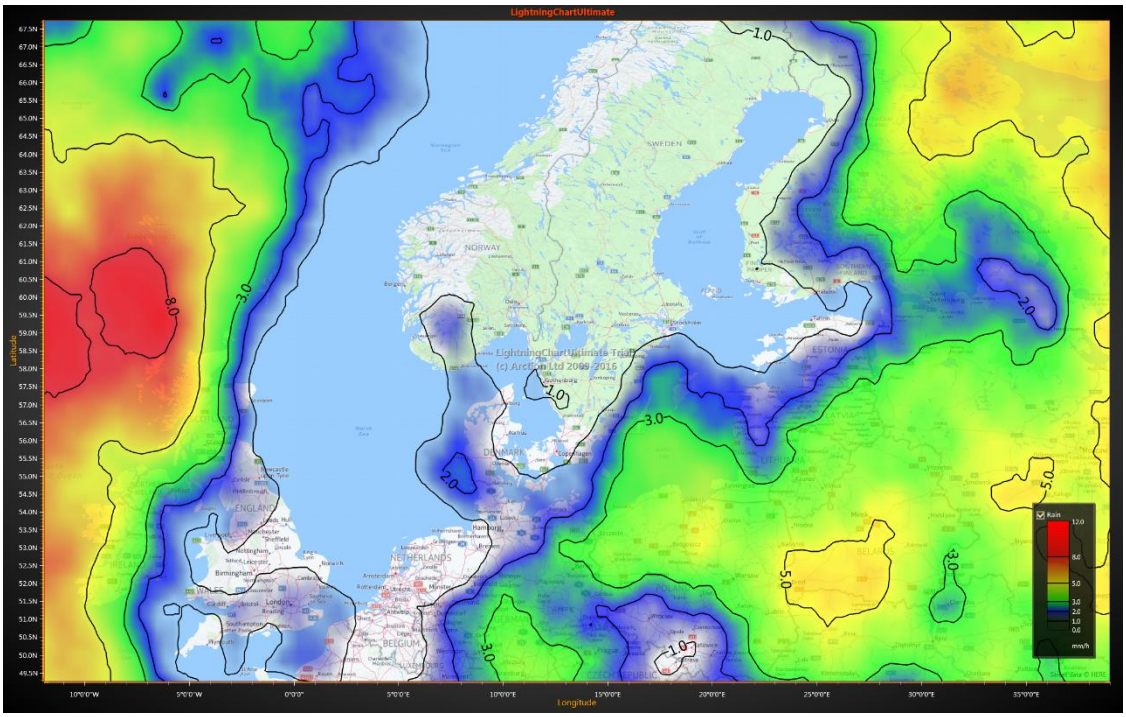


Figure 6-92. Street maps with IntensityGridSeries presenting weather data.

6.27 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.

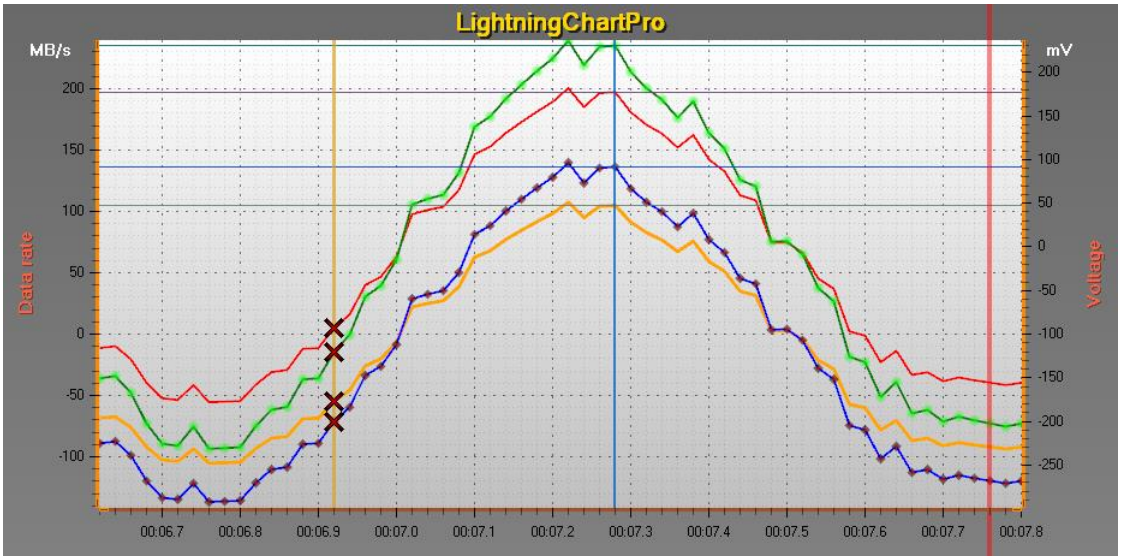


Figure 6-93. Line series cursors: Vertical point tracking cursor, hair-cross tracking cursor and vertical full-length cursor without tracking

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

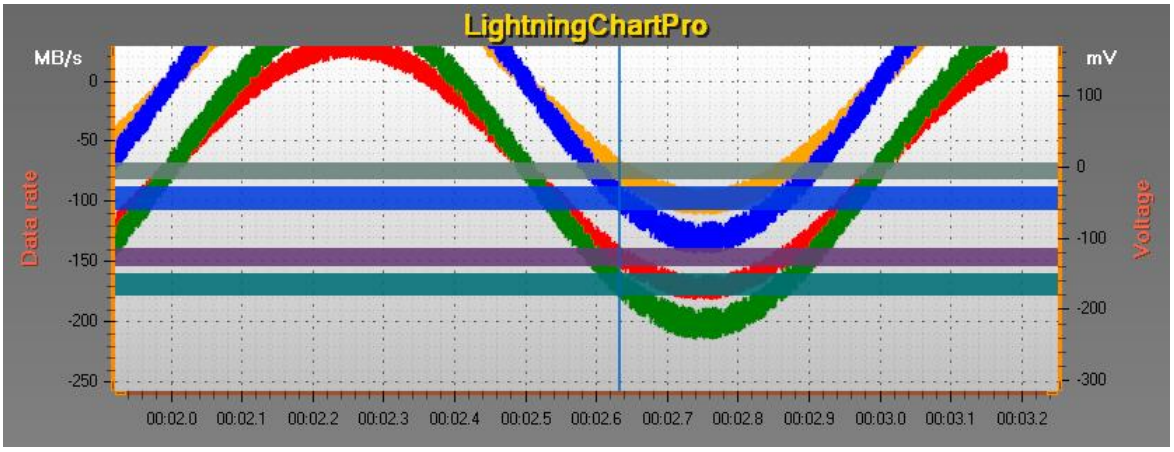


Figure 6-94. Hair-cross cursor with Y range indicator

6.27.1 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.

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).

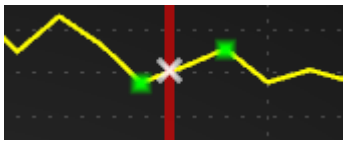


Figure 6-95. SolveYValueAtXValue interpolates the value between adjacent data points when SnapToPoints is disabled.

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 6.19

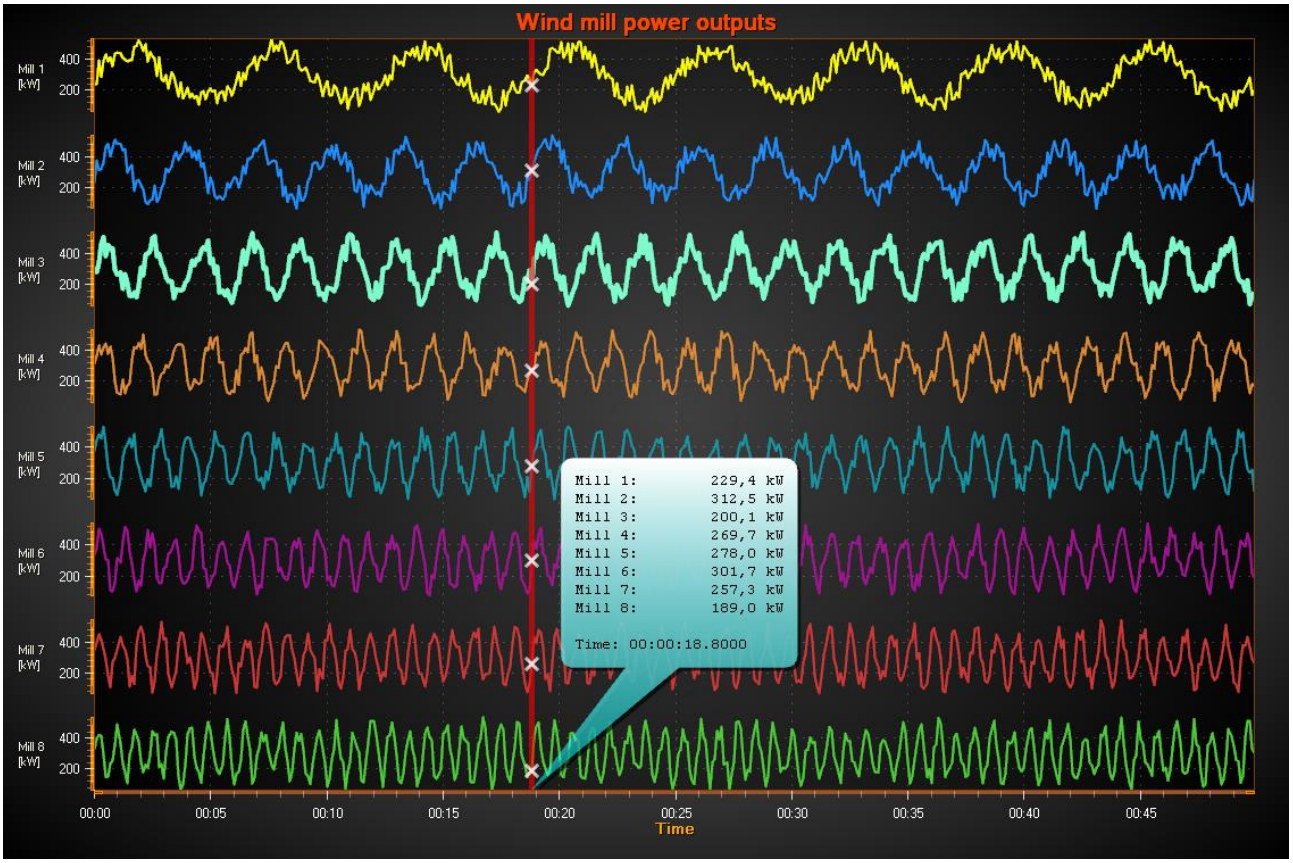


Figure 6-96. LineSeriesCursor used to track PointLineSeries. The values are shown in an AnnotationXY object.

6.28 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.

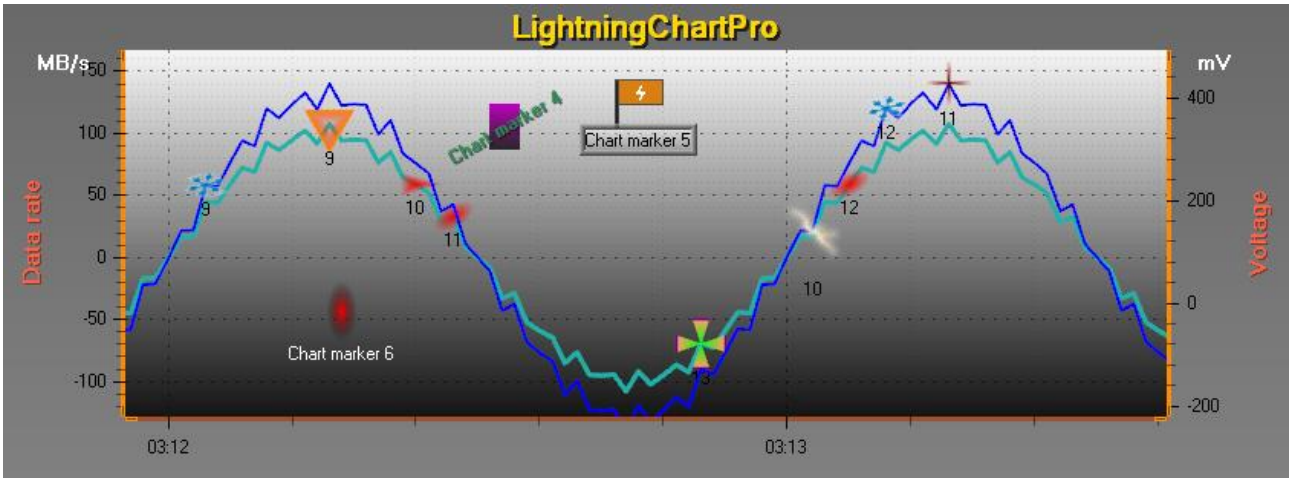


Figure 6-97. Some chart event markers and series event markers

6.28.1 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.

6.28.2 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.

6.29 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.

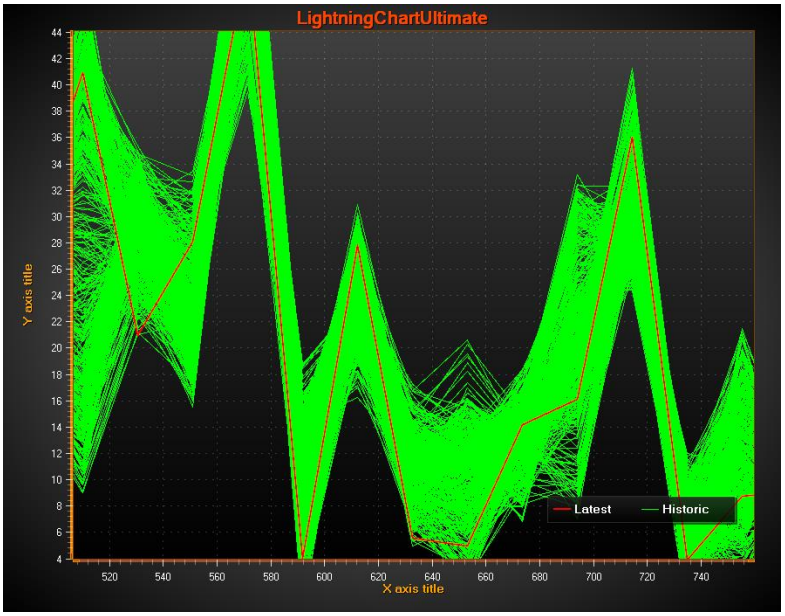


Figure 6-98. Persistent layer shows historical traces, in green color. A regular PointLineSeries is shown over it, in red color.

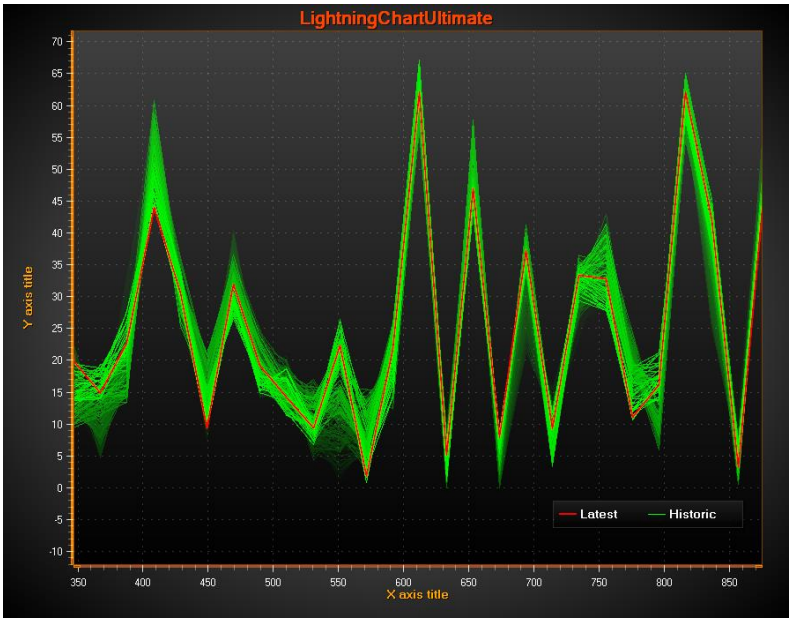


Figure 6-99. Persistent layer shows historical traces, in green color. MultiplyAlpha method is called before updating new data in the rendering layer, so it fades the oldest traces away.

6.29.1 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.LightningChartUltimate.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.

6.29.2 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));

6.29.3 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%.

6.29.4 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 6.5.4 for ***PointLineSeries***, 6.6.2 for ***SampleDataSeries***,6.7 for ***FreeformPointLineSeries***, ***6.9.4*** for ***HighLowSeries and 6.10.1*** 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).

6.29.5 Disposing the layer

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

6.29.6 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.

6.29.7 Getting list of layers

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

6.29.8 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.

6.30 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 6.29), 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.

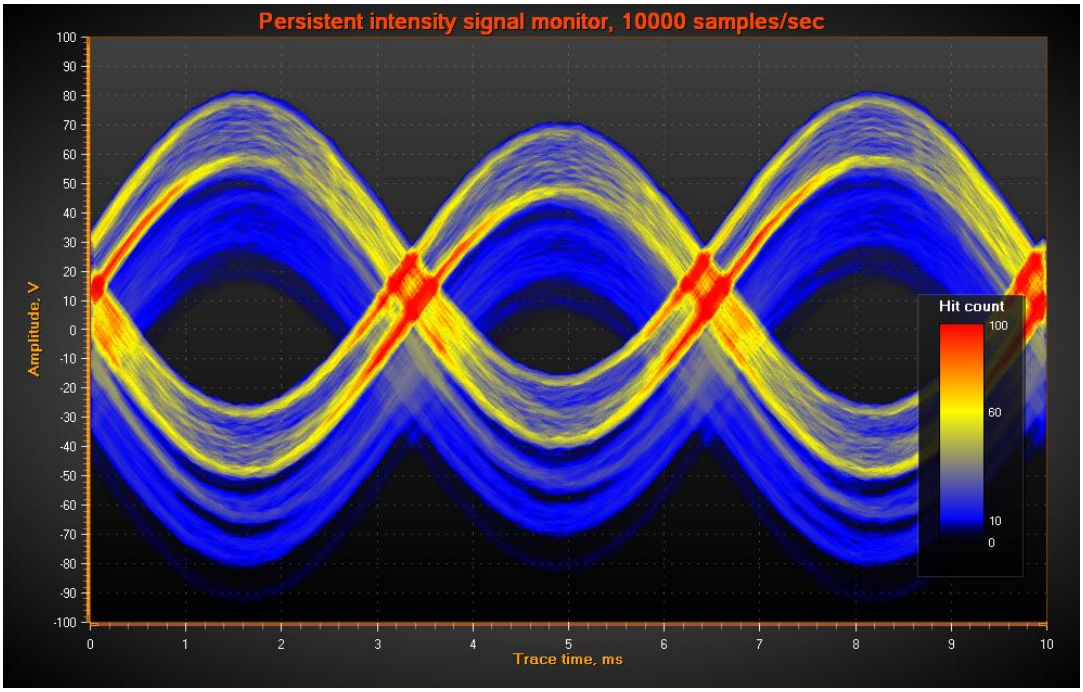


Figure 6-100. Persistent intensity layer highlights in where the activity has concentrated, here in yellow and red.

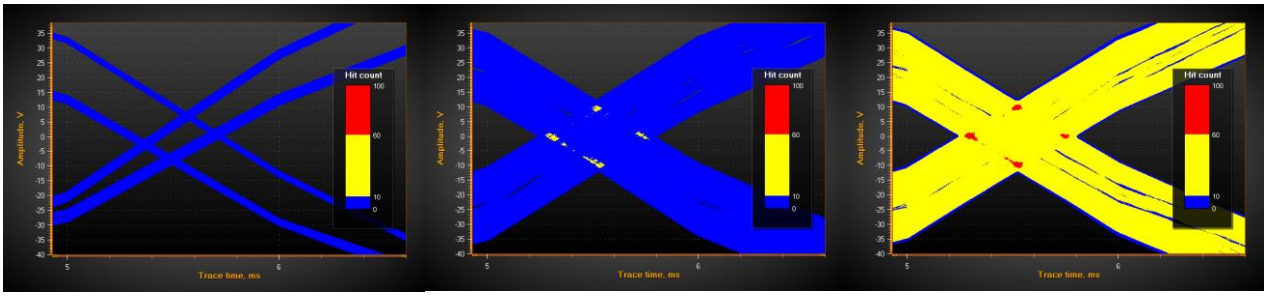


Figure 6-101. Repetitive signal trace is rendered in the same region. On the left, only a couple of traces have been rendered on the layer, so it shows all colors in blue. In the picture on the center, a lot of traces have been rendered, but mostly on different coordinates. In the intersections of the 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.

6.30.1 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
PersistentSeriesRenderingIntensityLayerngLayer(m_chart.ViewXY,
m_chart.ViewXY.XAxes[0]);
```

6.30.2 Clearing the layer

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

6.30.3 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.

6.30.4 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.

6.30.5 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.

6.30.6 Disposing the layer

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

6.30.7 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.

6.30.8 Getting list of layers

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

7. View3D

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

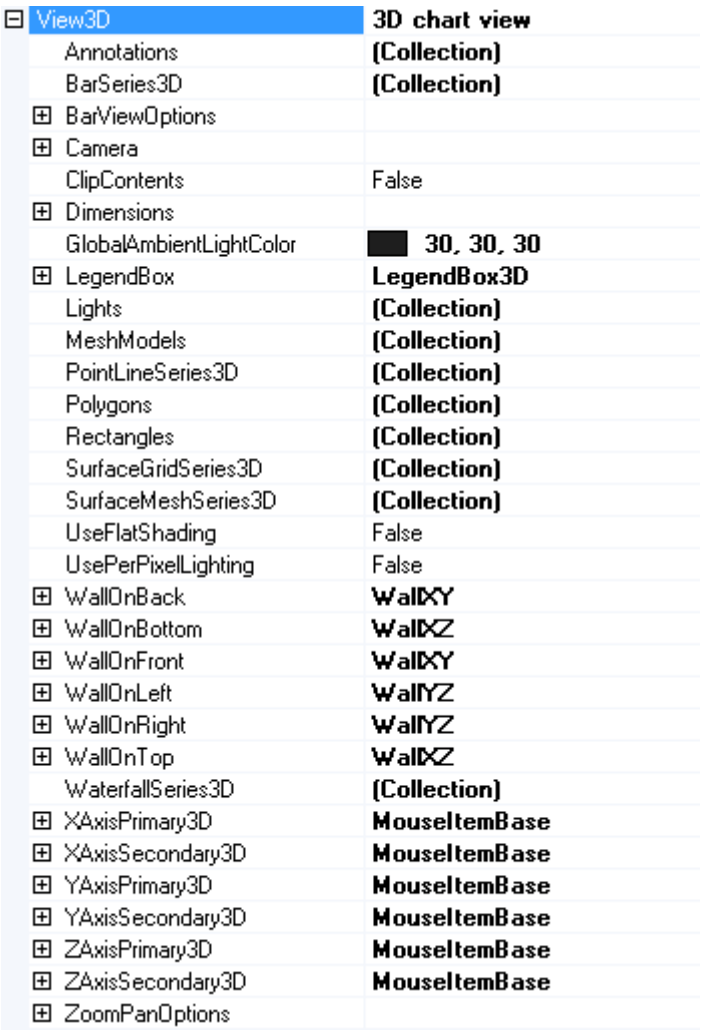


Figure 7-1. View3D object main tree.

7.1 3D model and dimensions

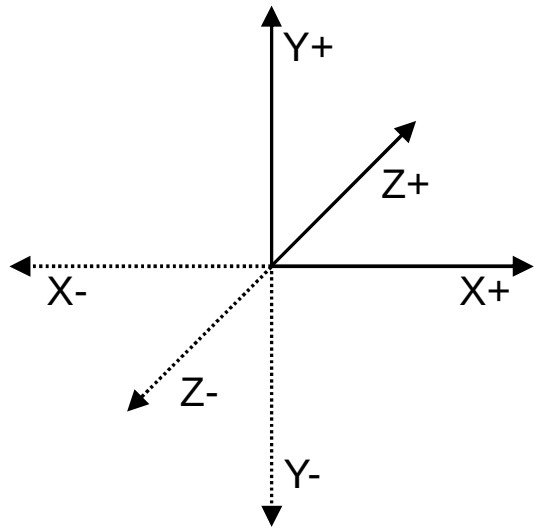


Figure 7-2. 3D model positive and negative directions.

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.

7.1.1 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.

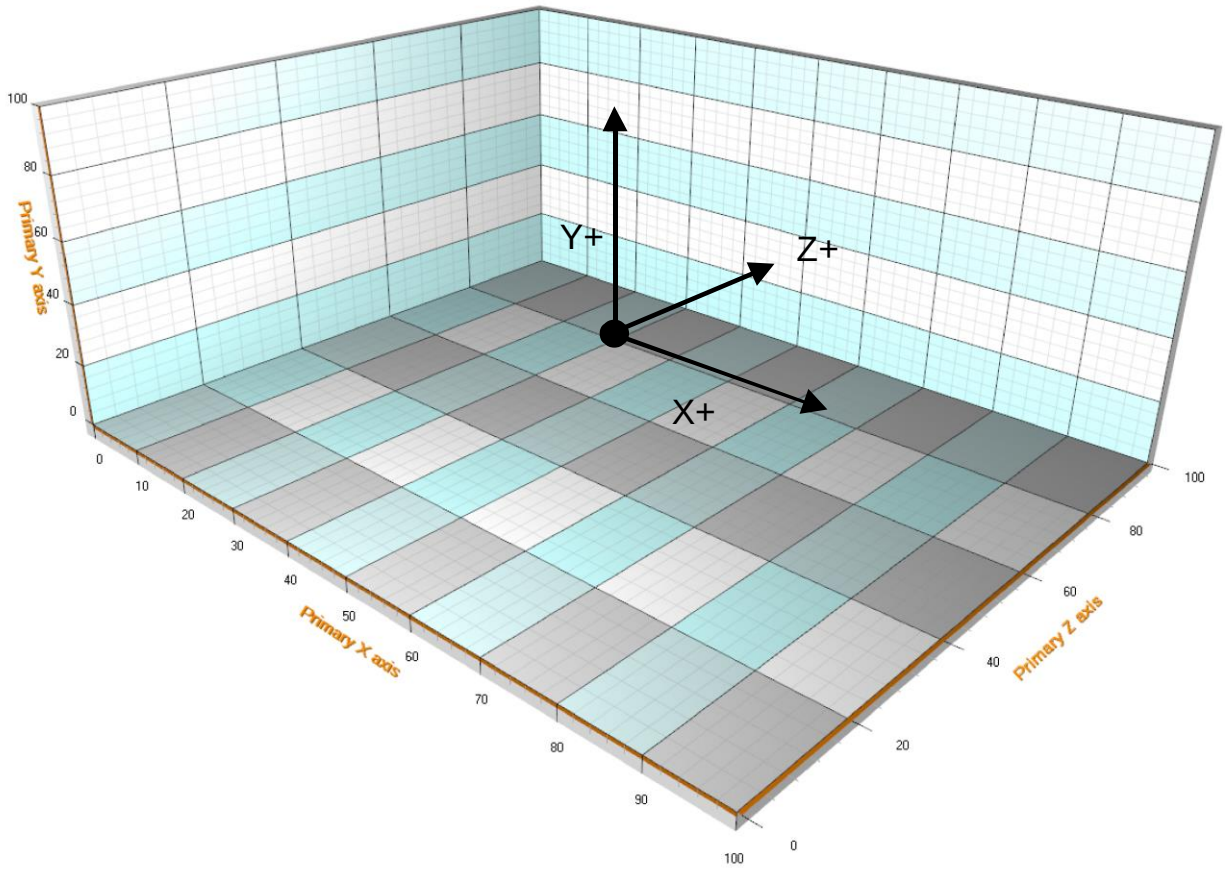


Figure 7-3. Example of 3D view setup. Dimensions are set to X=100, Y=40, Z=80. Walls visible: left, bottom, back. Perspective camera is used.

7.2 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.

7.3 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 7-4. 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 orghographic camera, the ***ViewDistance*** does not effect. Perspective camera shows the model in real-life way, and orthographic camera is better for certain technical applications.

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

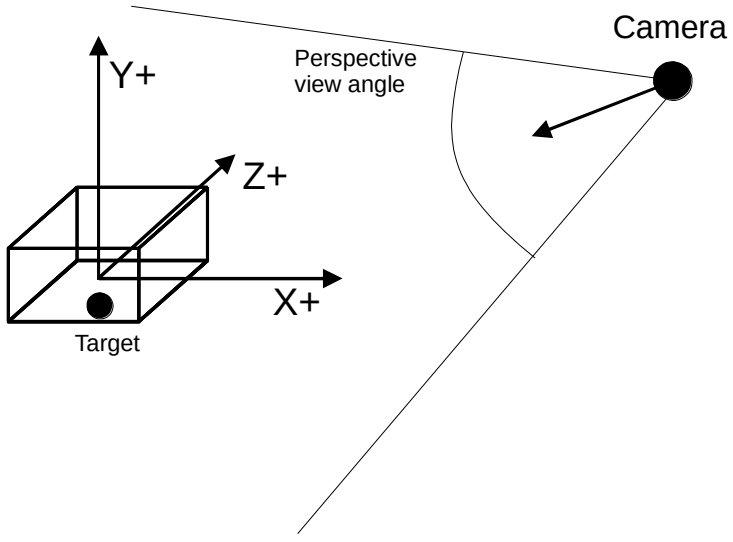


Figure 7-5. Perspective camera presentation in 3D space.

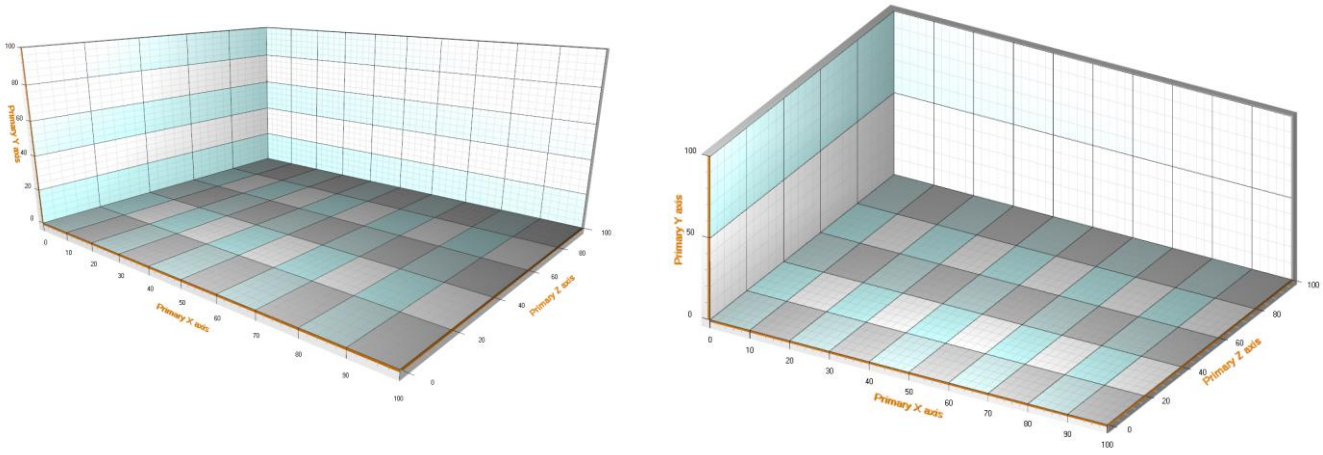


Figure 7-6. Perspective camera view and orthographic camera view in 3D space.

7.3.1 Predefined cameras

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

7.4 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***.

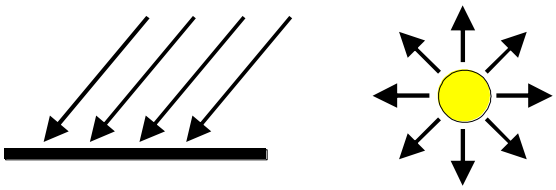


Figure 7-7. Directional light and point of light.

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.

7.4.1 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.

7.4.2 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.

7.4.3 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.

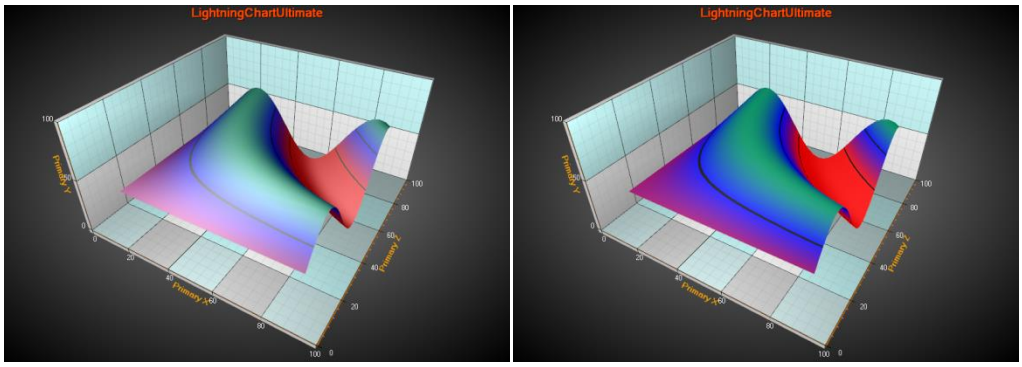


Figure 7-8. The picture on the left has ColorSaturation = 50%. The picture on the right has ColorSaturation = 85%.

7.4.4 Predefined lighting schemes

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

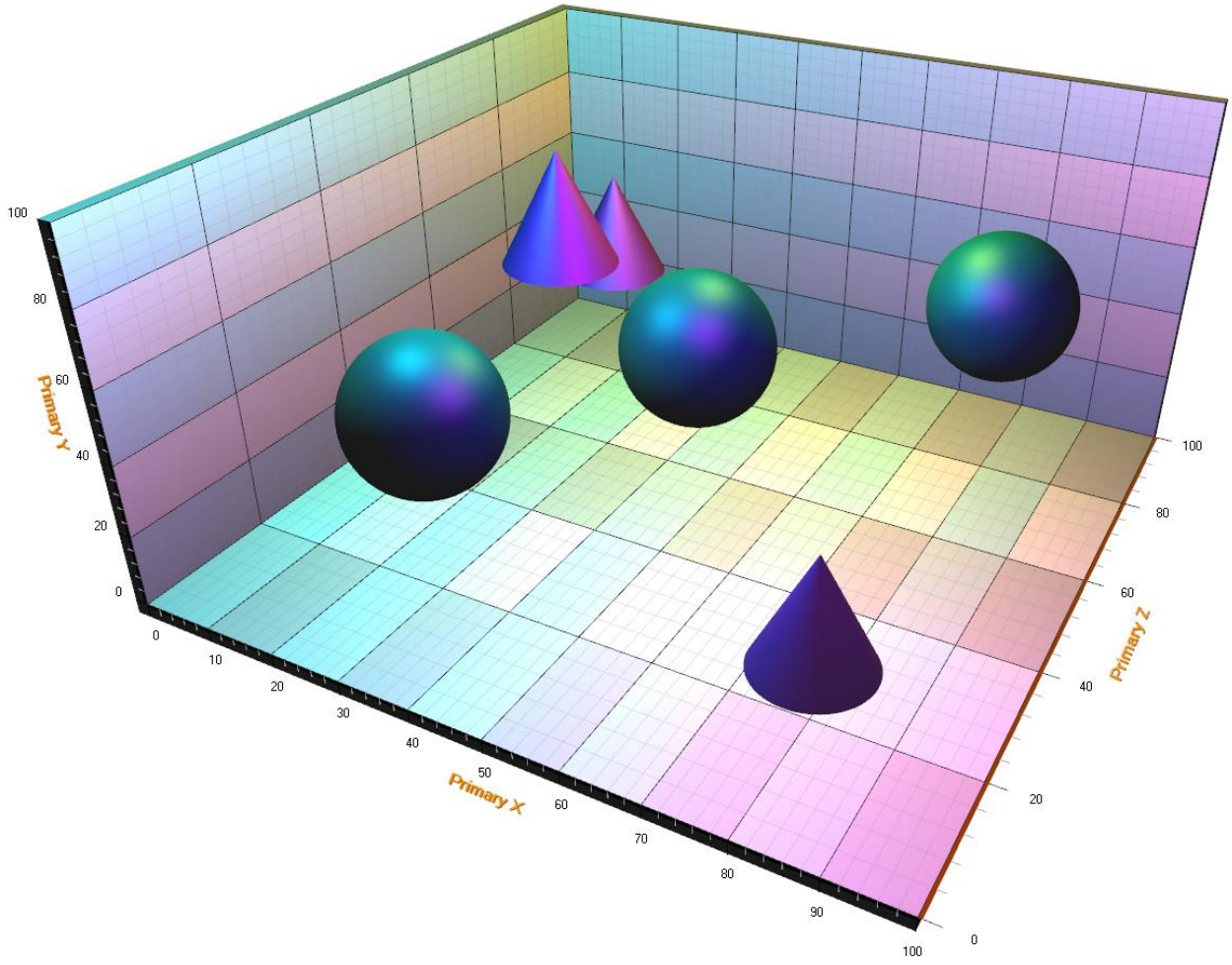


Figure 7-9. Predefined 'DiscoCMY' scheme in use. The scheme is composed from three differently colored PointOfLights near the ceiling. The spheres and cones are made with a couple of series of type PointLineSeries3D.

7.5 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.

7.5.1 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***.

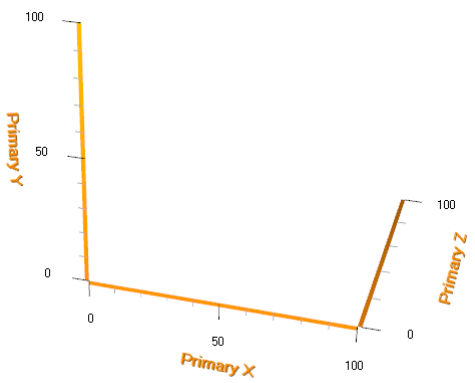


Figure 7-5. Default axis location setup, XAxisPrimary at BottomFront, YAxisPrimary at FrontLeft and ZAxisPrimary in BottomRight.

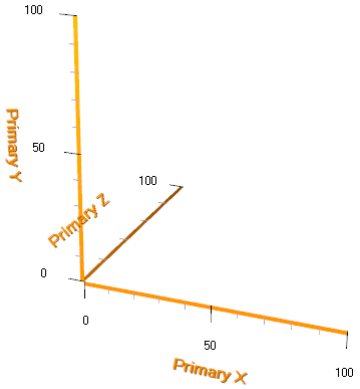


Figure 7-5. ZAxisPrimary location set to BottomLeft.

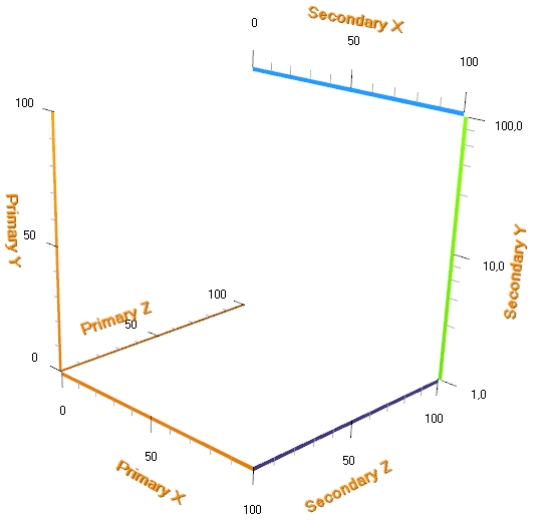


Figure 7-10. Secondary axes set visible and their locations and colors set arbitrarily. Secondary Y axis ScaleType set to Logarithmic.

7.5.2 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

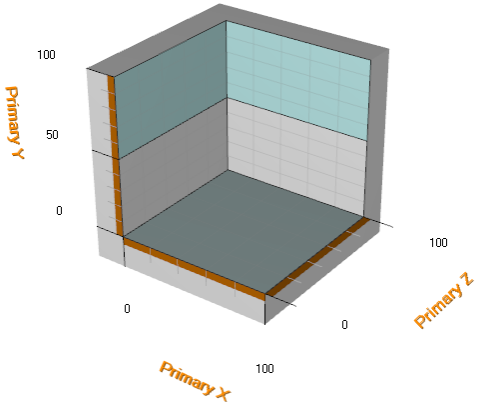


Figure 7-5. X axis orientation is set to XY, Y axis orientation to XY, Z axis orientation to XZ.

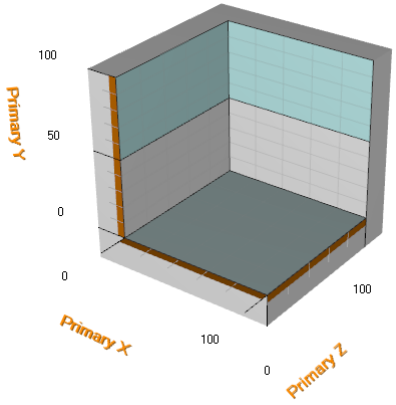


Figure 7-5. Y axis orientation stays same, but X axis orientation is changed to XZ and Z axis orientation is changed to ZY plane.

7.5.3 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.

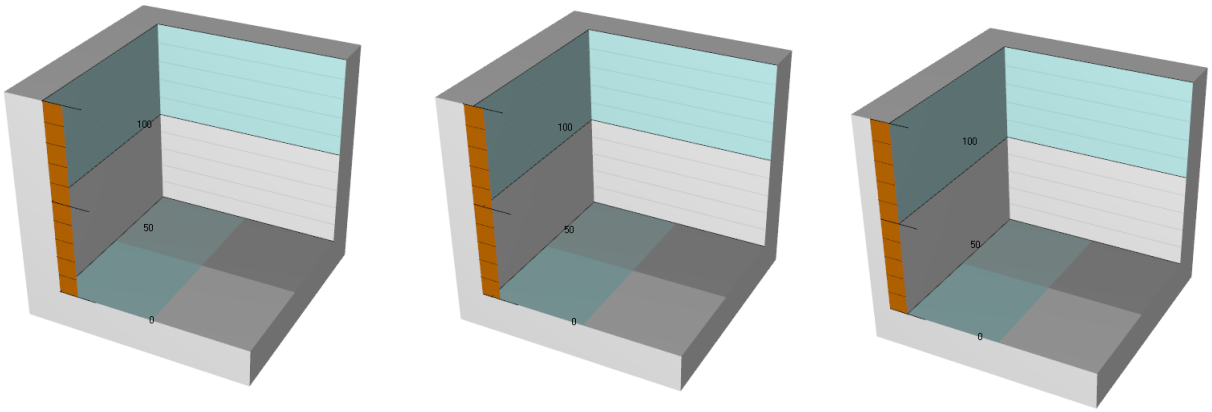


Figure 7-11. First figure: Only Y axis is visible in this example. Y Axis CornerAlignment is set to Inside. Alignment properties in MajorDivTickStyle and MinorDivTickStyle are set to Near. Second figure: CornerAlignment is set to AtCorner. Third picture: CornerAlignment is set to Outside.

7.6 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.

7.7 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.

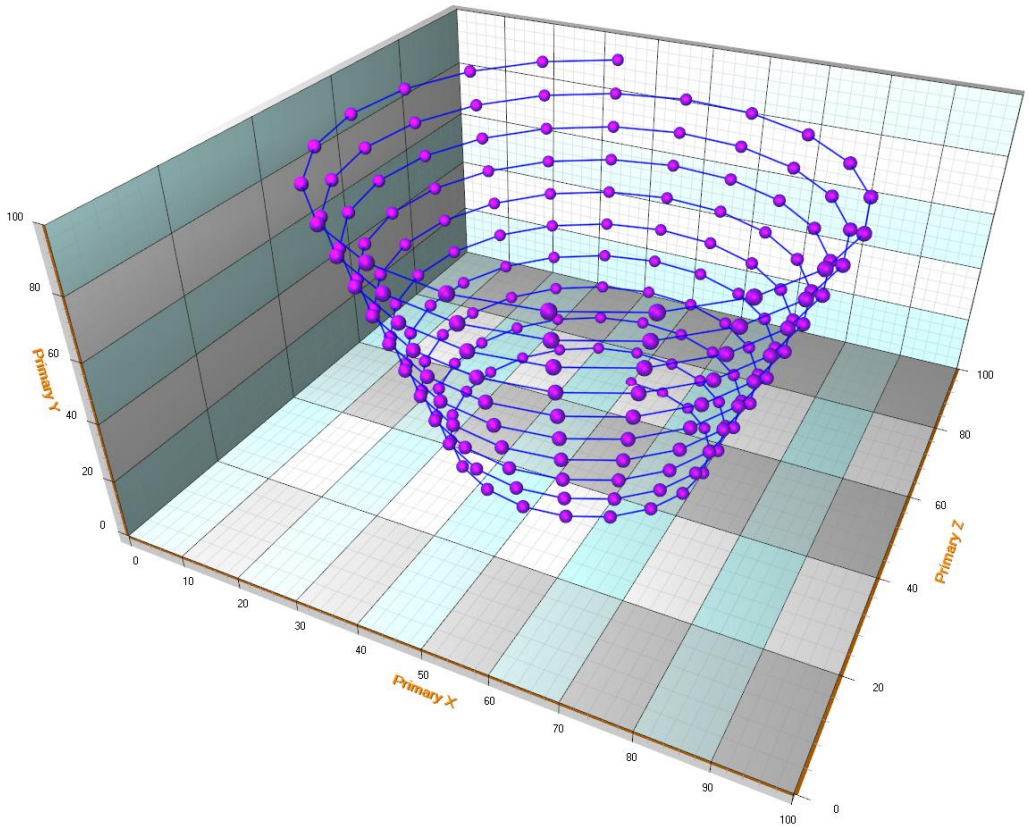


Figure 7-12. A PointLineSeries3D example. PointStyle's Shape is set to Sphere.

7.7.1 Point styles

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

PointStyle	
DetailLevel	30
Rotation3D	
Shape2D	
Angle	**45**
Antialiasing	True
BitmapAlphaLevel	255
BitmapImage	**(none)**
BitmapImageTintColor	White
BodyThickness	**3**
BorderColor	**128, 0, 0, 0**
BorderWidth	0
Color1	**Red**
Color2	**Black**
Color3	Black
GradientFill	Edge
Height	**13**
LinearGradientDirection	Down
Shape	**Cross**
UseImageSize	True
Width	**13**
Shape3D	Box
ShapeType	**Shape2D**
Size3D	

Figure 7-13. PointStyle property tree. ShapeType is the switch between 2D and 3D shapes.

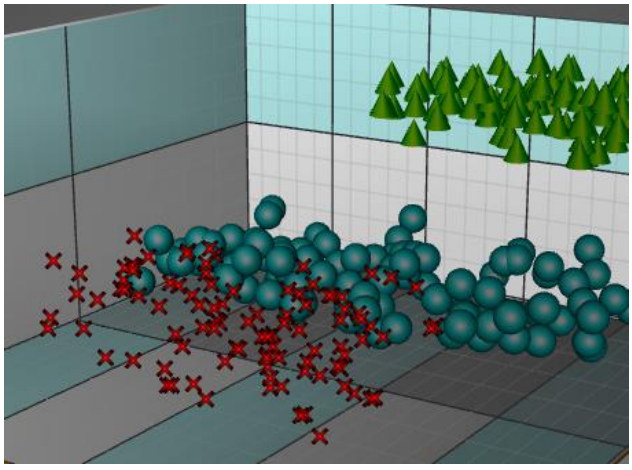


Figure 7-14. Red crosses are have ShapeType = Shape2D. Teal and Green objects have ShapeType = Shape3D.

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.

7.7.2 Line styles

LineStyle	
AntiAliasing	**Normal**
Color	Yellow
LineOptimization	**Hairline**
Pattern	Solid
PatternScale	1
Width	**0.2**

Figure 7-15. 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 recommeded, otherwise performance is impaired.

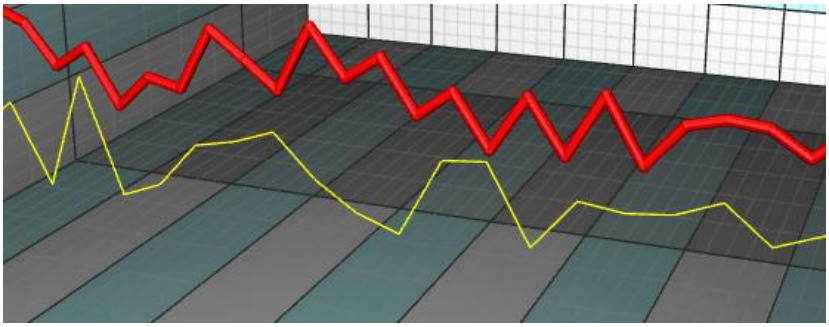


Figure 7-16. Yellow line: LineStyle.LineOptimization = Hairline. Red Line: LineStyle.LineOptimization = NormalShaded.

7.7.3 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.

Points

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

double X:	X axis value
double Y:	Y axis value
double Z:	Z axis value
Color color:	individual data point color, only applies when ***IndividualPointColors*** is enabled, or ***MultiColorLine*** is enabled.
double sizeFactor:	size factor multiplies the size defined by ***PointStyle.Size*****.** Only applies when using ***IndividualPointSizes*** is enabled.
object Tag:	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
```

PointsCompact

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

SeriesPointCompact3D structure consists of the following fields:

float X:	X axis value
float Y:	Y axis value
float Z:	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
```

7.7.4 Coloring points individually

By setting ***IndividualPointColors*** = ***True***, the color fields of points apply instead of ***Material.DiffuseColor.***

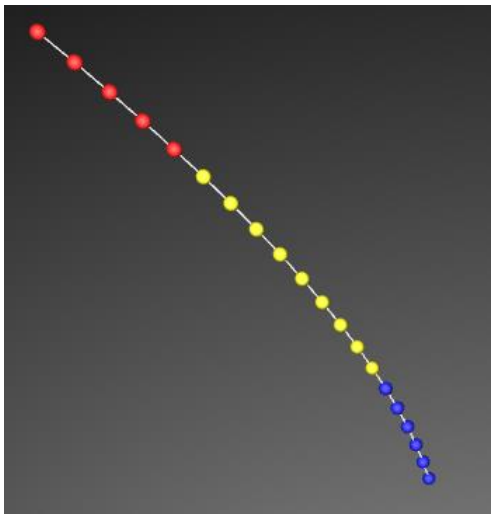


Figure 7-17. IndividualPointColors in use.

Note! Individual point coloring is not supported when having PointsType = PointsCompact.

7.7.5 Setting points sizes individually

By setting ***IndividualPointSizes = True***, ***sizeFactor*** fields from the points take effect. The factor multiplies the size defined in ***PointStyle.Size***.

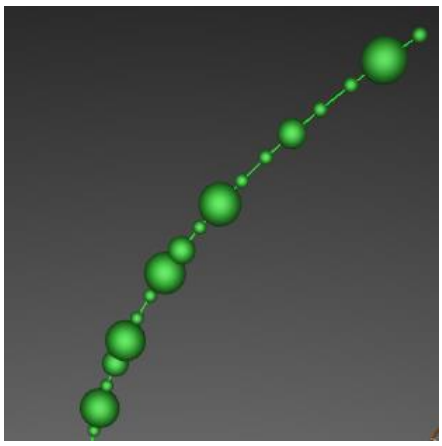


Figure 7-18. IndividualPointSizes in use.

Note! Individual point sizes is not supported when having PointsType = PointsCompact.

7.7.6 Multi-coloring line

To color the line with given data point colors, set ***MultiColorLine = True.*** The chart interpolates the color gradients between adjacent points.

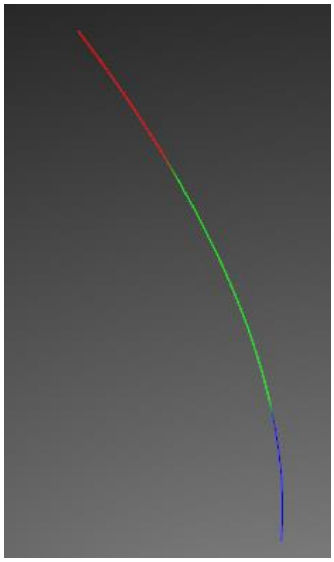


Figure 7-19. MultiColorLine enabled.

Note! MultiColorLine is not supported when having PointsType = PointsCompact.

7.7.7 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.

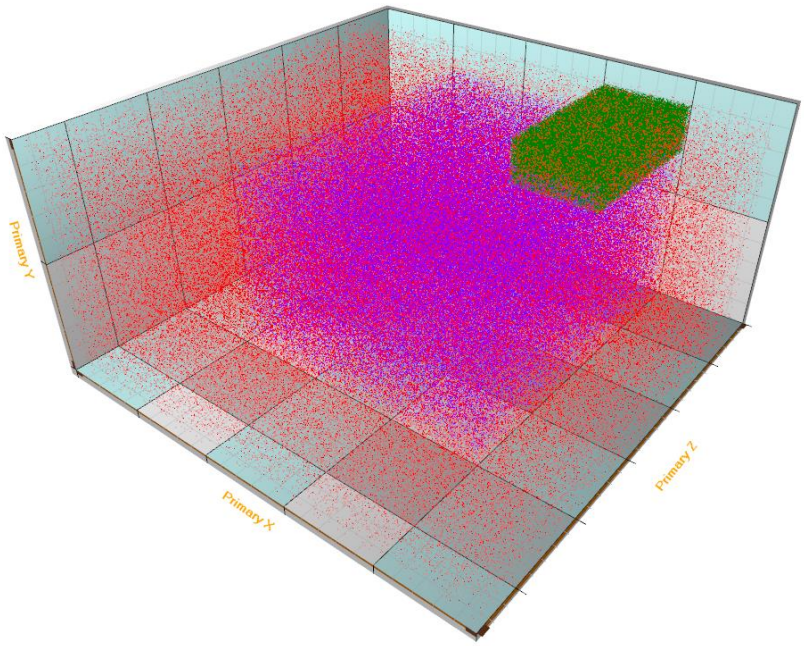


Figure 7-20. Millions of scatter points. LineVisible = False, PointsVisible = True, PointsOptimization = Pixels.

7.8 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.

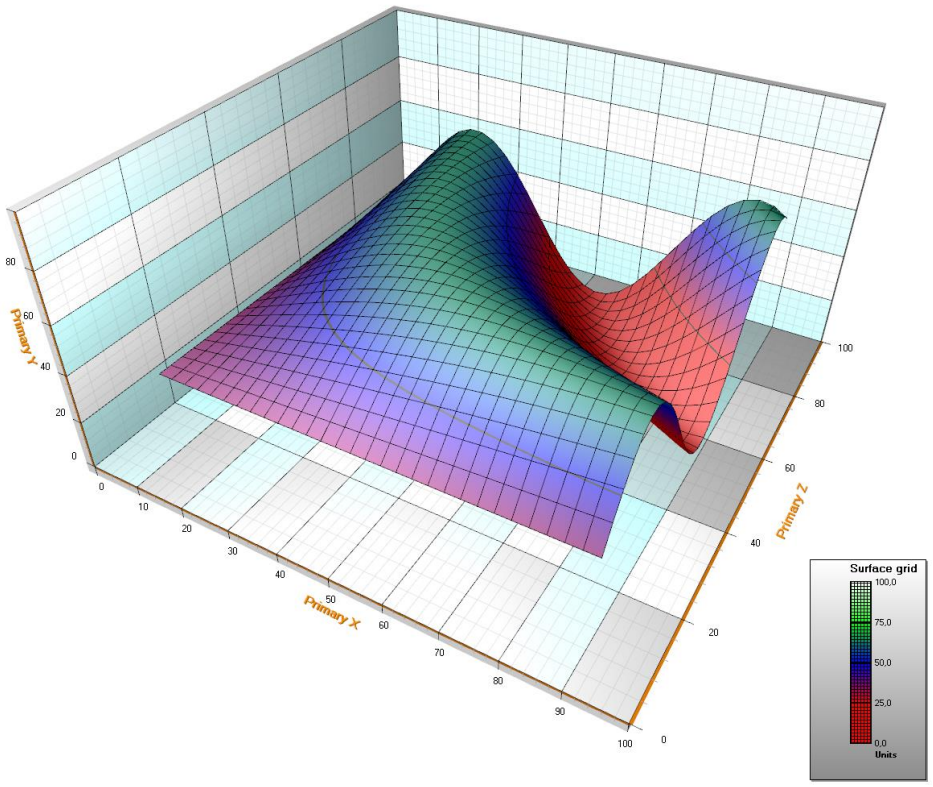


Figure 7-21. Surface grid series with default style. Height data is made with a sine formula. Legend box shows the height coloring intervals.

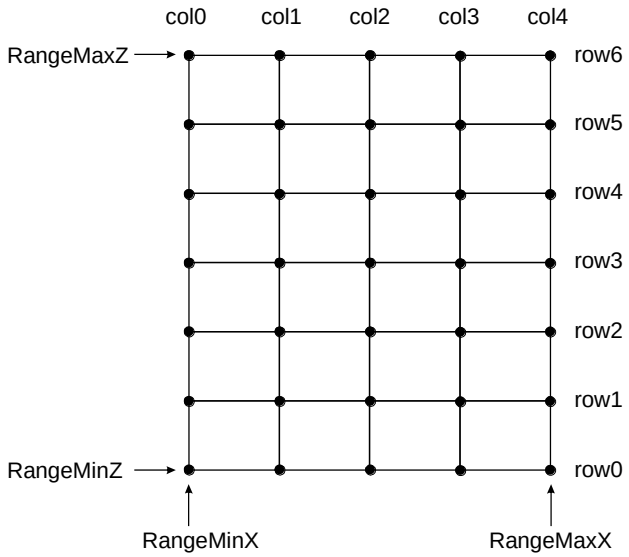


Figure 7-22. Surface grid nodes. SizeX = 5, SizeZ = 7.

Node distances are automatically calculated as

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

$$node\ distance\ Z = \frac{\text{RangeMaxZ} - \text{RangeMinZ}}{\text{SizeZ} - 1}$$

7.8.1 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 ís 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
```

7.8.2 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.

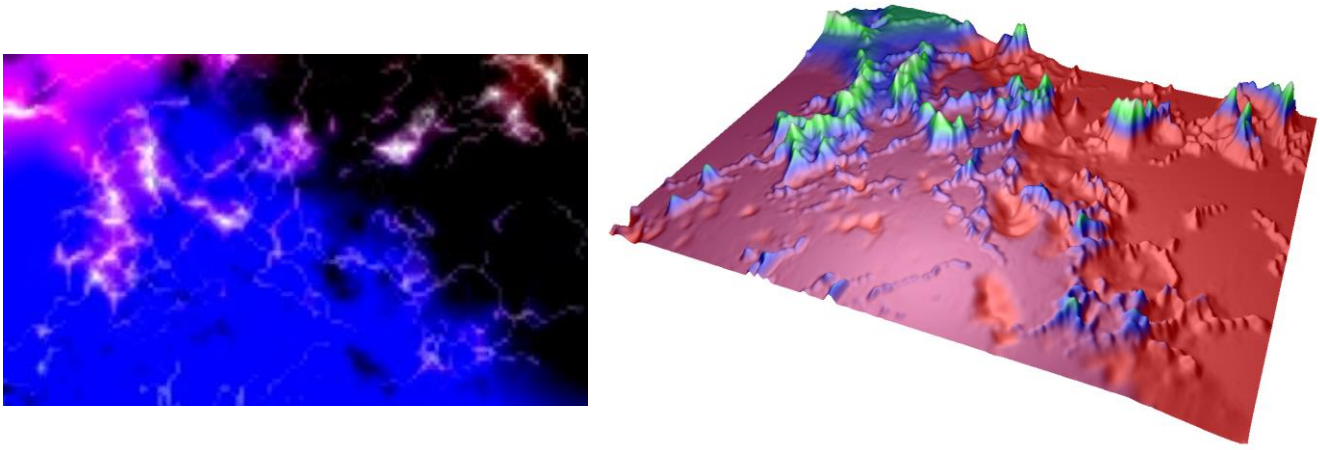


Figure 7-23. Source bitmap and calculated surface height data. Dark values stay low and bright values get higher in the surface.

7.8.3 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 7.8.4.
- ***PalettedByValue***: Coloring by **SurfacePoint's Value** fields by palette, see section 7.8.4.
- ***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.

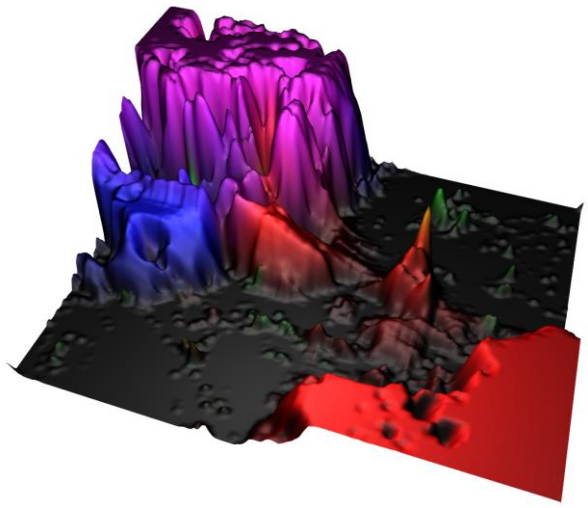


Figure 7-5. FromSurfacePoints fill. Color per data point.

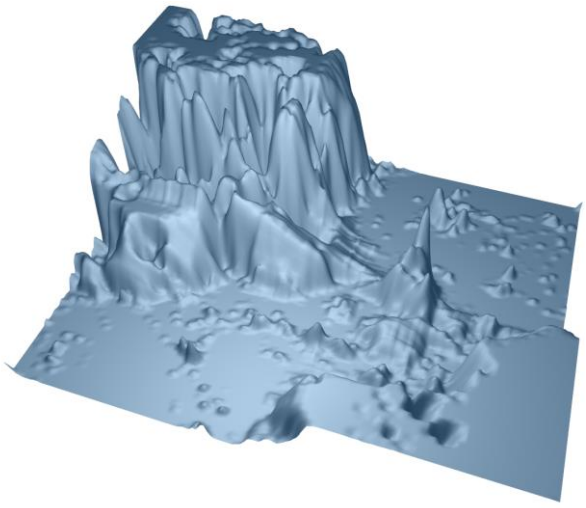


Figure 7-5. Toned fill.

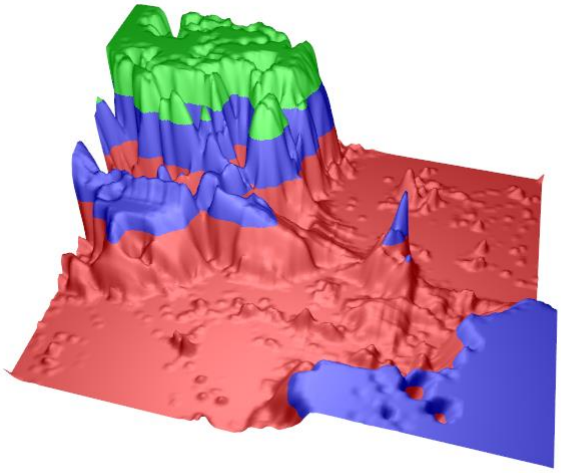


Figure 7-5. PalettedByY

Figure 7-5. Bitmap fill.

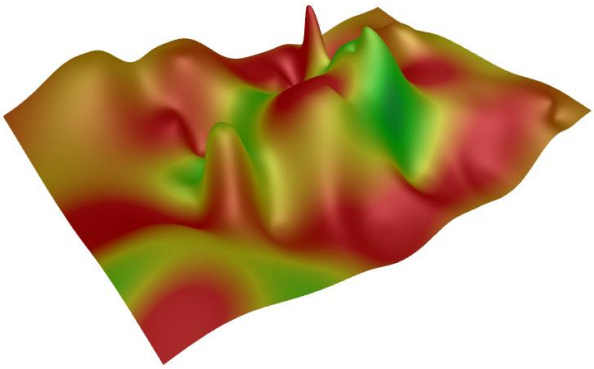


Figure 7-5. PalettedByValue.

7.8.4 Contour palette

Figure 7-5. PalettedByYValue

With ***ContourPalette*** property, you can define color steps for height coloring. ***ContourPalette*** can be used for

- ***Fill*** (see section 7.8.3)
- ***Wireframe mesh*** (see section 7.8.5)
- ***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.

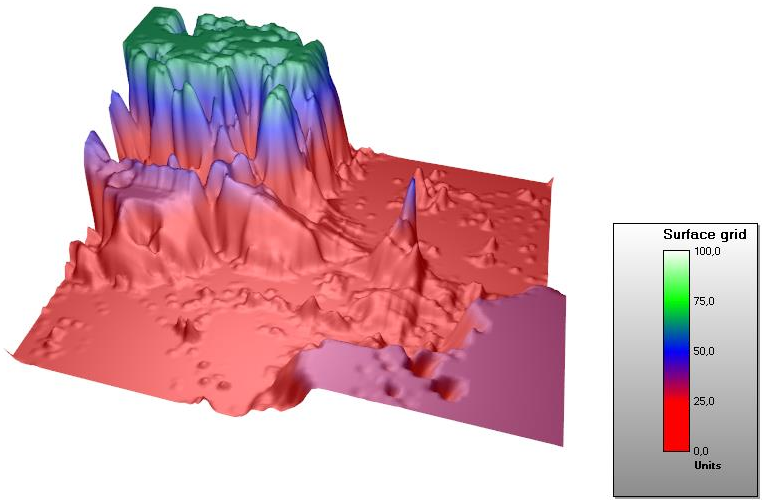


Figure 7-24. Surface grid series contour palette Type is set to Gradient.

7.8.5 Wireframe mesh

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

- ***None***: no wireframe
- ***Wireframe***: a solid color wireframe. Use ***WireframeLineType.Color*** to set the color
- ***WireframePalettedByY***: the wireframe coloring follows SurfacePoint's **Y** field ***ContourPalette*** (see section 7.8.4)
- ***WireframePalettedByY***: the wireframe coloring follows SurfacePoint's **Value** field, ***ContourPalette*** (see section 7.8.4)
- ***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***.

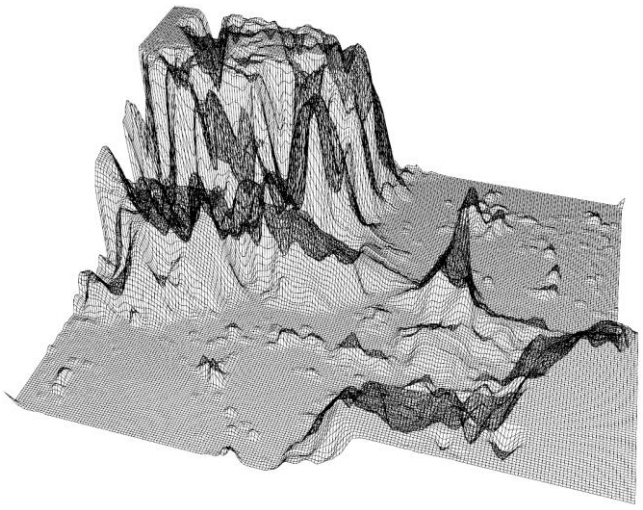


Figure 7-26. WireframeType = Wireframe.

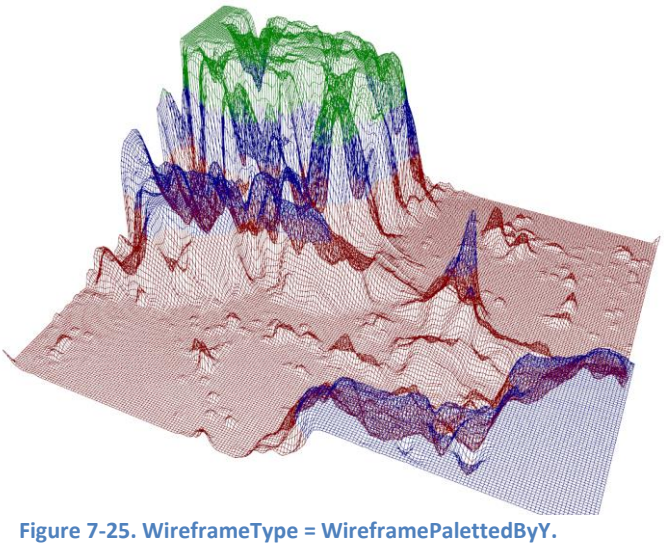


Figure 7-25. WireframeType = WireframePalettedByY.

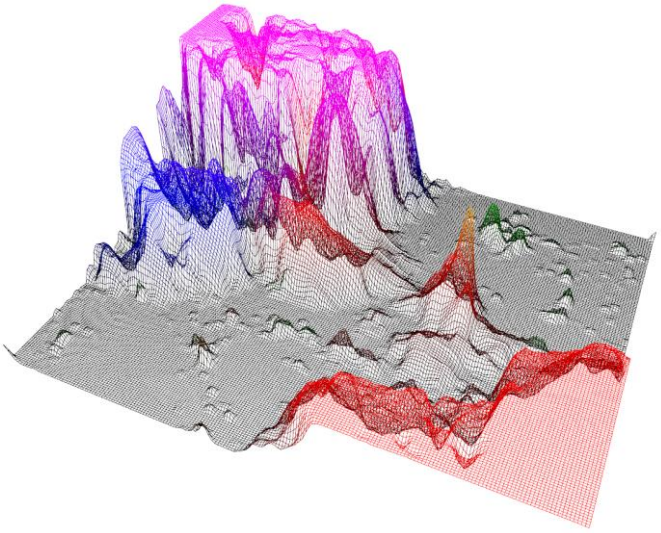


Figure 7-28. WireframeType = SourcePointColored.

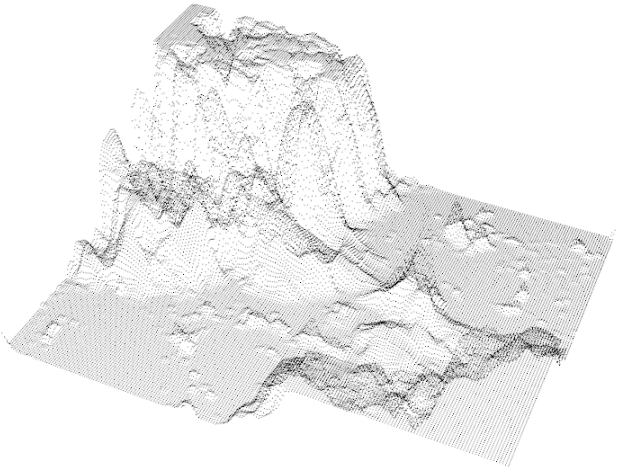


Figure 7-27. WireframeType = Dots.

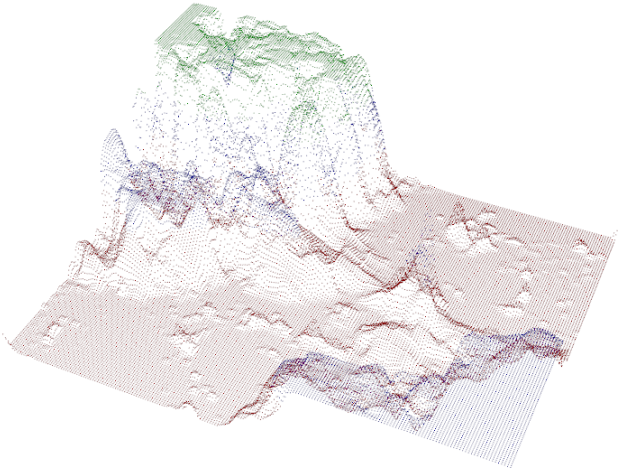


Figure 7-30. WireframeType = DotsPalettedByY.

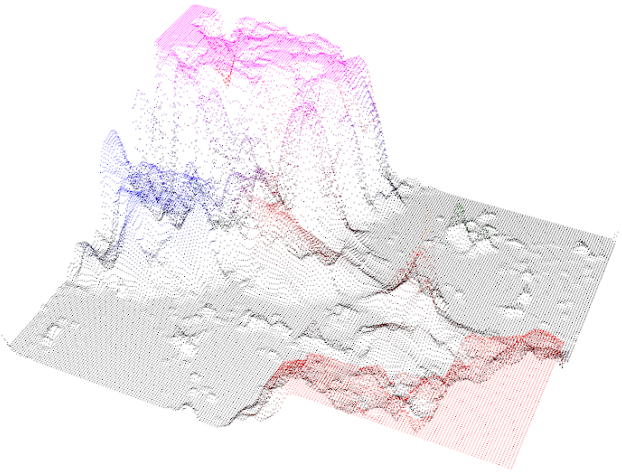


Figure 7-29. WireframeType = DotsSourcePointColored.

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.

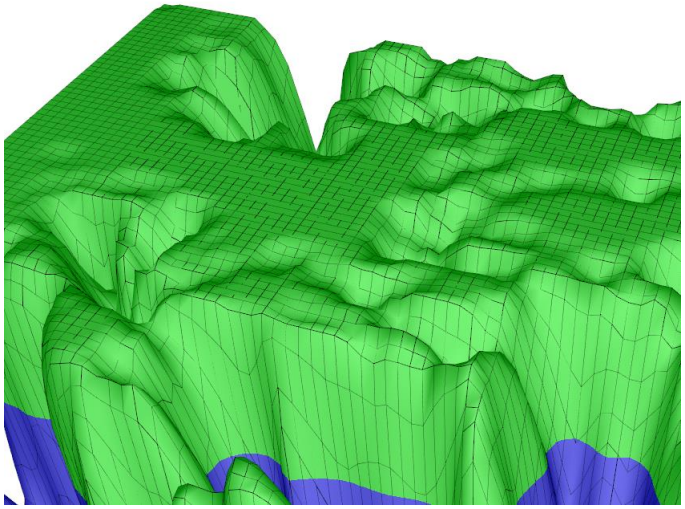


Figure 7-31. Surface grid wireframe with filling. Z-fighting appears as broken wireframe lines.

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.

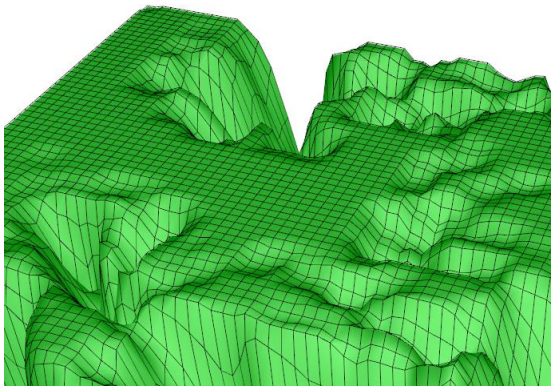


Figure 7-32. WireframeOffset = (X=0; Y=0.1; Z=0).

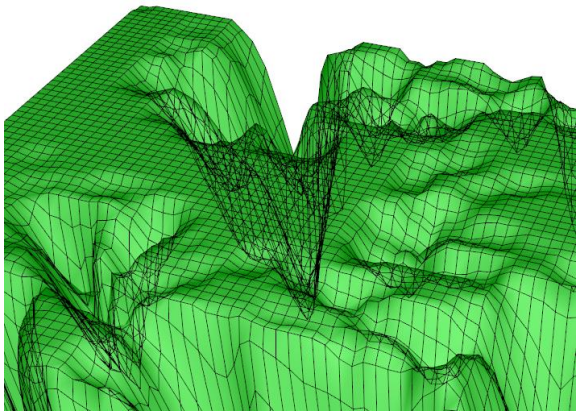


Figure 7-33. DrawWireframeThrough is enabled.

7.8.6 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 ***ContourLineType*** 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 7.8.4)
- ***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 7.8.4)

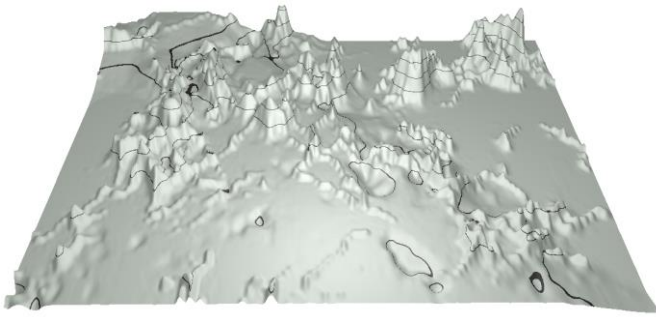


Figure 7-34. ContourLineType = FastColorZones.

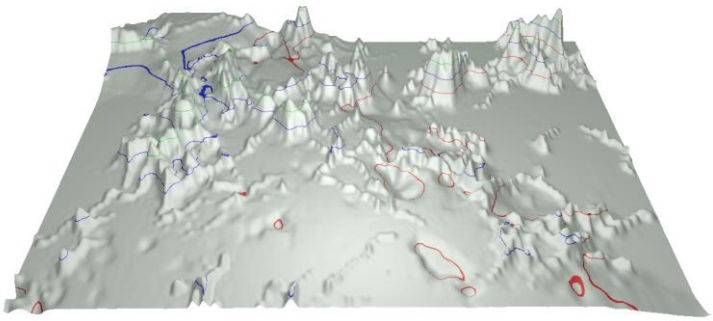


Figure 7-35. ContourLineType = FastPalettedZones.

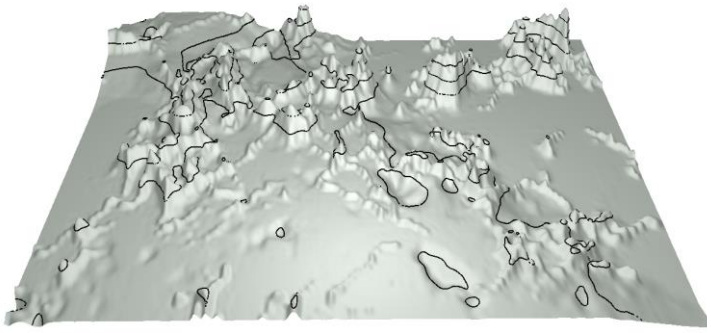


Figure 7-36. ContourLineType = ColorLine.

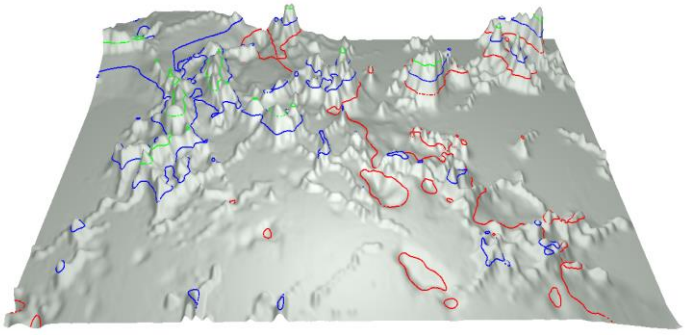


Figure 7-37. ContourLineType = PalettedLine.

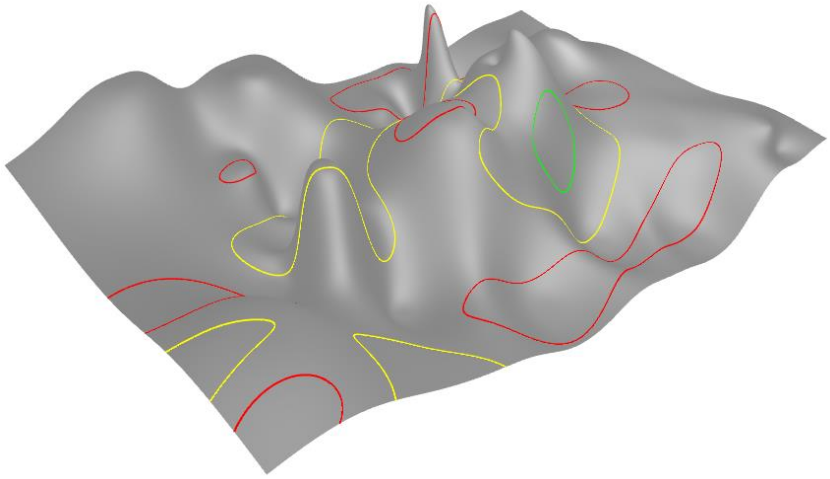


Figure 7-38. ContourLineType = PalettedLineByValue.

7.8.7 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.

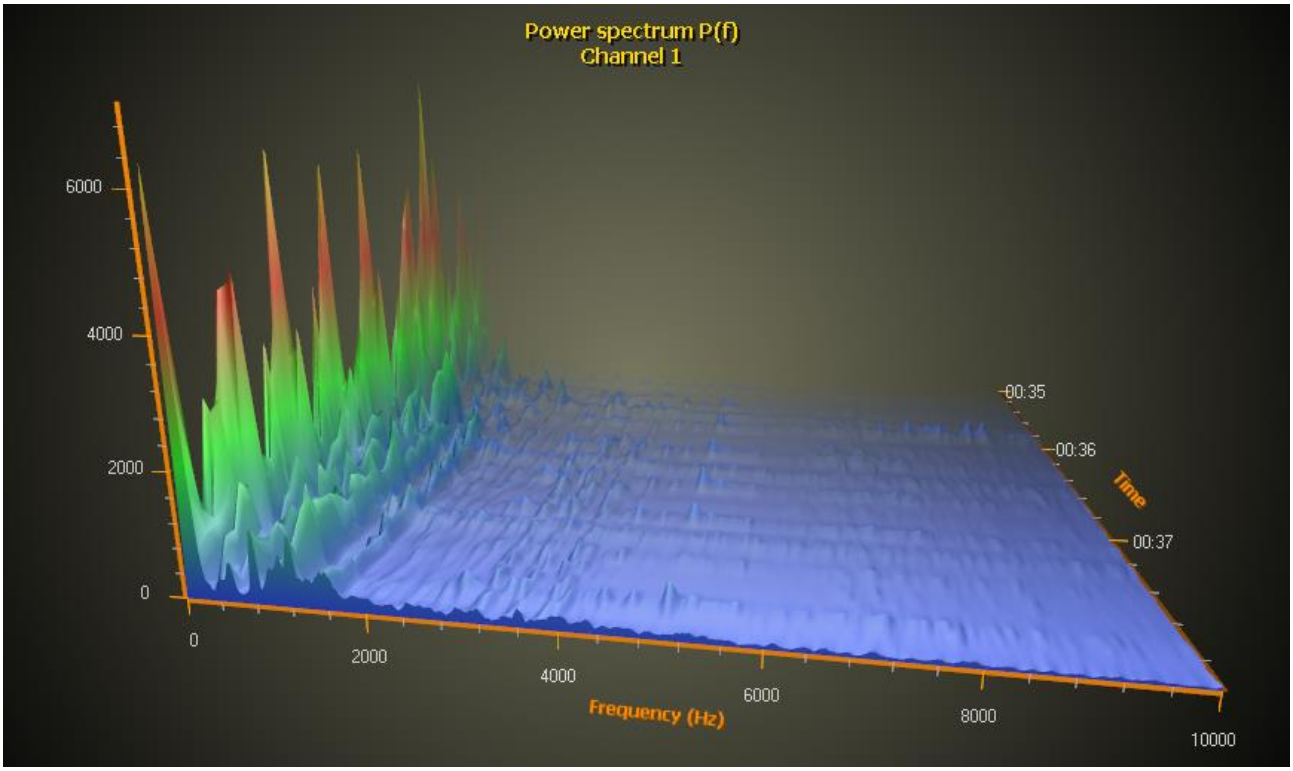


Figure 7-39. Presenting 3D spectrum with surface grid. InsertRowBackAndScroll method is used for performance optimized data adding. Fadeaway property is 100 to make the surface smoothly fade away towards back. A perspective camera is used.

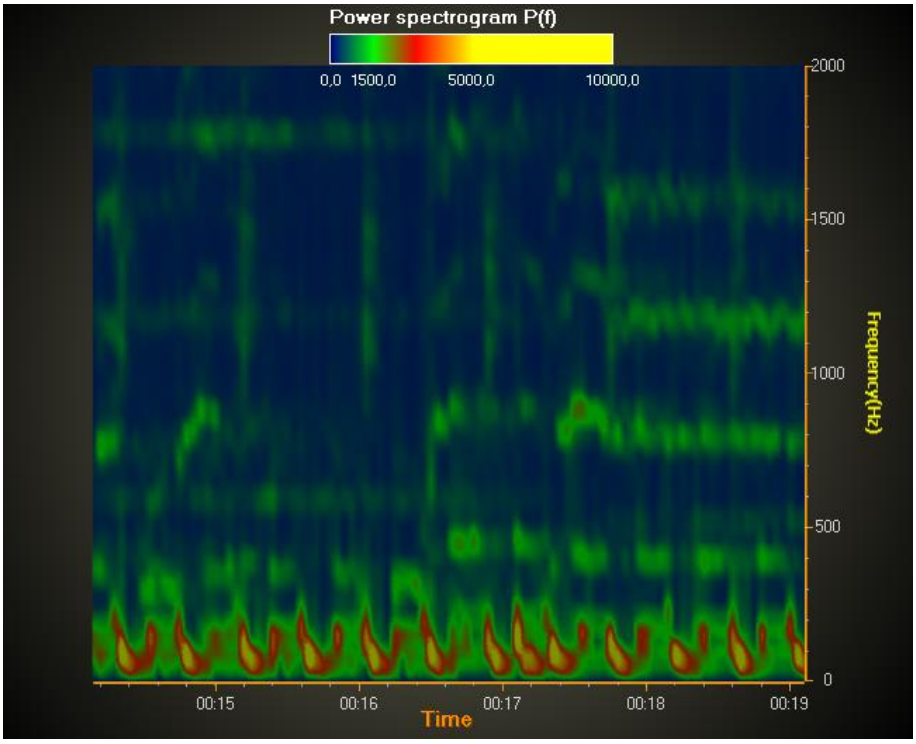


Figure 7-40. Presenting spectrogram with surface grid. InsertColumnBackAndScroll method is used. An orthographic camera above the model is used to give straight and perpendicular projection. SuppressLighting is enabled for removing the light reflections. Fadeaway = 0 for making the grid series fully visible.

7.9 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.

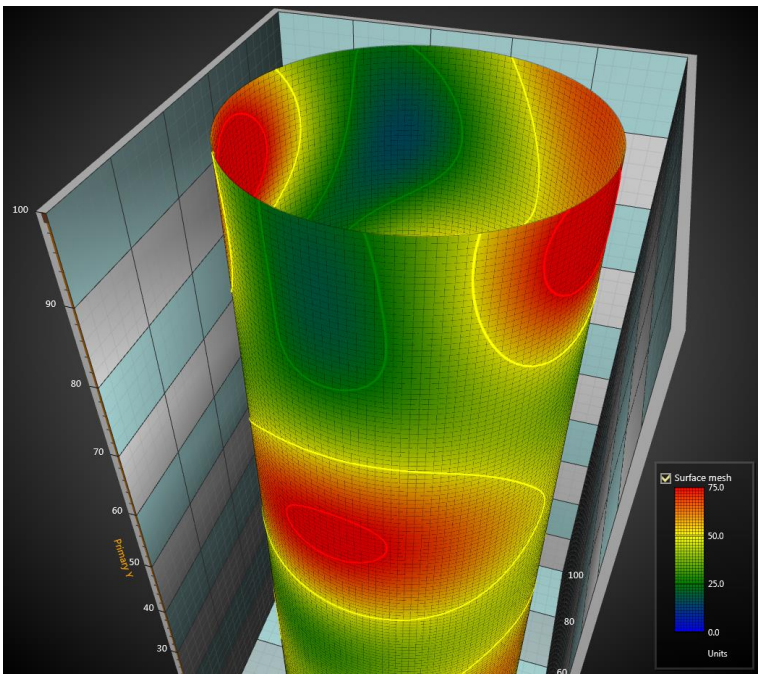


Figure 7-41. SurfaceMeshSeries3D, geometry made as a pipe.

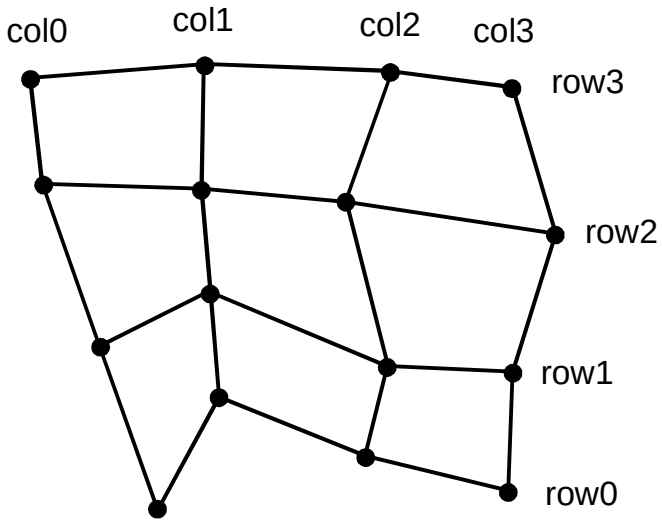


Figure 7-42. Surface mesh nodes. SizeX = 4, SizeZ =4.

7.9.1 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
```

7.9.2 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***

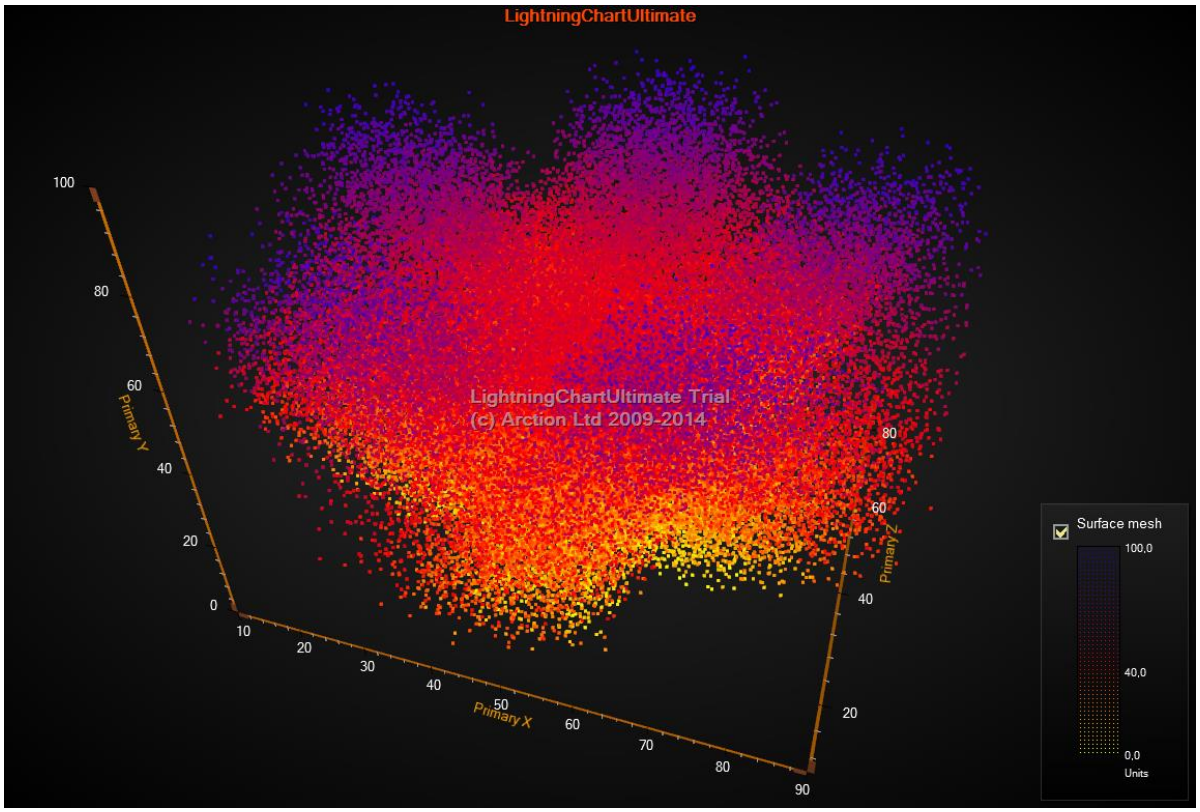


Figure 7-43. Point cloud visualization with palette coloring of points.

7.10 WaterfallSeries3D

With ***WaterfallSeries3D***, the data is visualized in area strips. Areas can be filled, wire-framed and contour-lined like ***SurfaceGridSeries3D***, see section 7.8. In Y-dimension, area starts from ***BaseLevel*** property value. The node data can be set like in ***SurfaceMeshSeries3D***, see section 7.9.1.

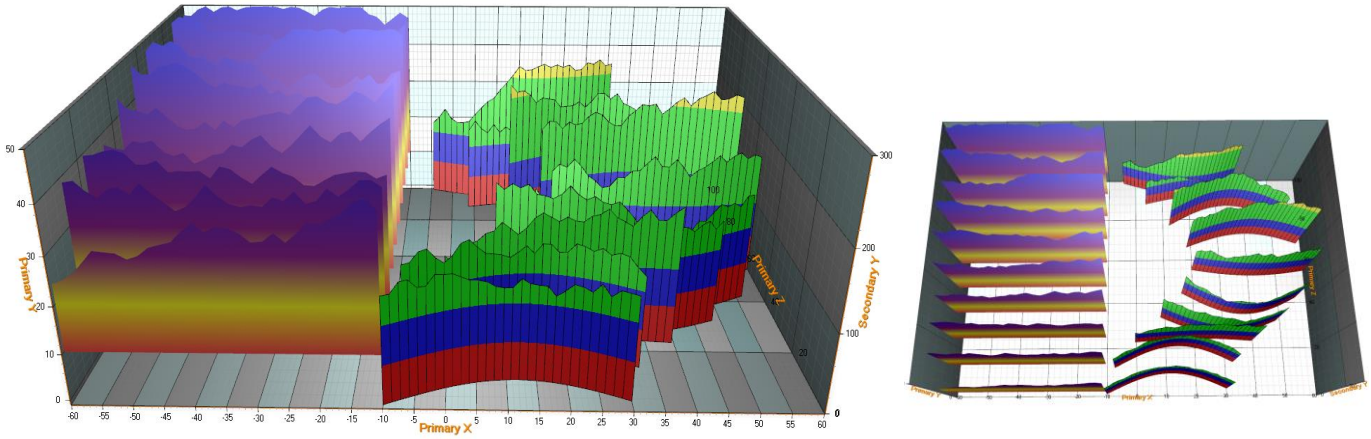


Figure 7-44. Two waterfall series. On the left violet series, X and Z are in rectangular form. BaseLevel = 10. On the right red-green-blue series, X and Z values are bent, and each row is placed in different horizontal location.

WaterfallSeries3D is especially handy for presenting traditional 3D spectrum.

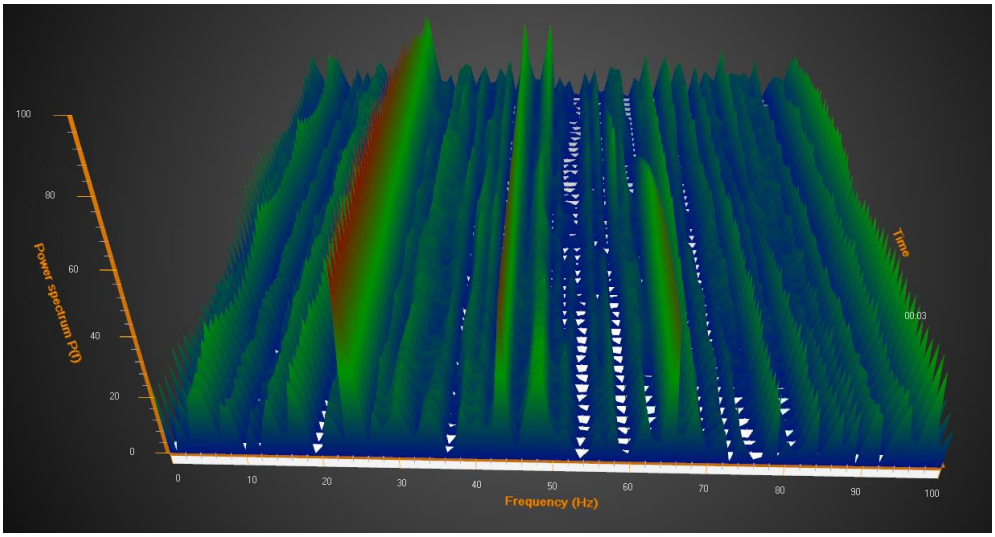


Figure 7-45. Waterfall series used for traditional spectrum presentation.

7.11 BarSeries3D

BarSeries3D allow bar data visualization in 3D.

7.11.1 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.

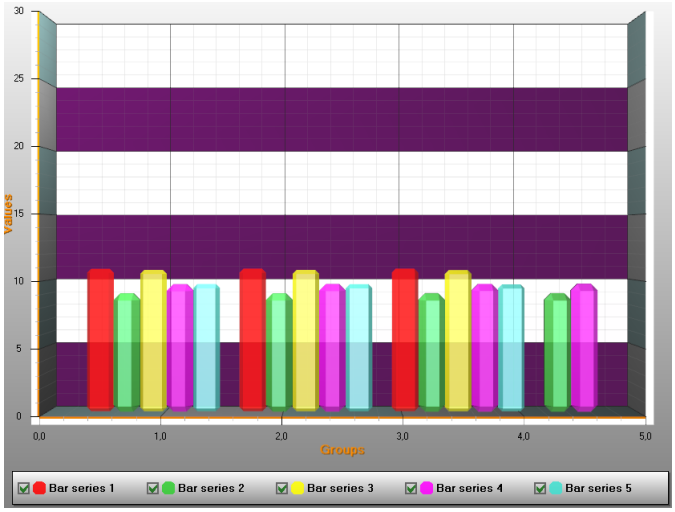


Figure 7-46. BarViewOptions.ViewGrouping = GroupedIndexedFitWidth. Bar widths and group gaps are arranged to fit the width nicely.

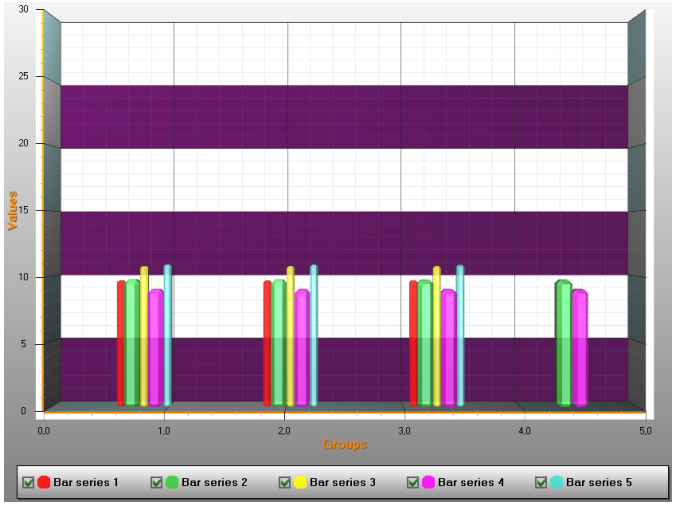


Figure 7-47. BarViewOptions.ViewGrouping = GroupedIndexed. Original bar widths apply and groups are arranged to fit the chart width.

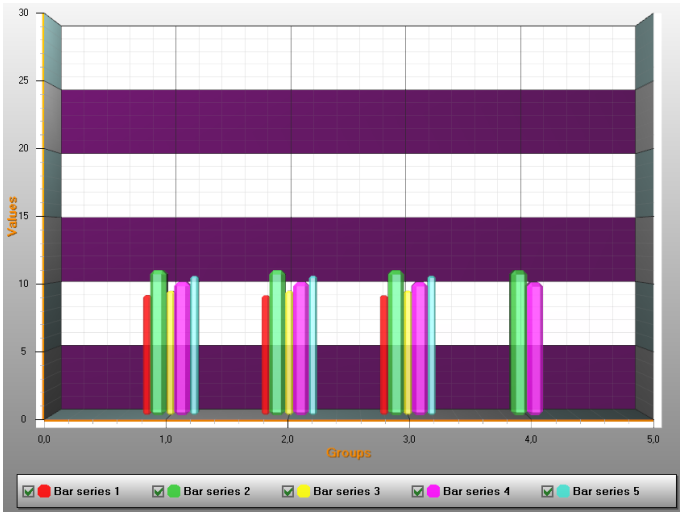


Figure 7-48. BarViewOptions.ViewGrouping = GroupedByXValue. Bar X values apply.

Figure 7-49. BarViewOptions.ViewGrouping = StackedIndexed. All bars having same index are stacked.

Figure 7-50. 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 7-51. BarViewOptions.ViewGrouping = StackedStretchedToSum. All bars having same X value are stacked and stretched to StackSum.

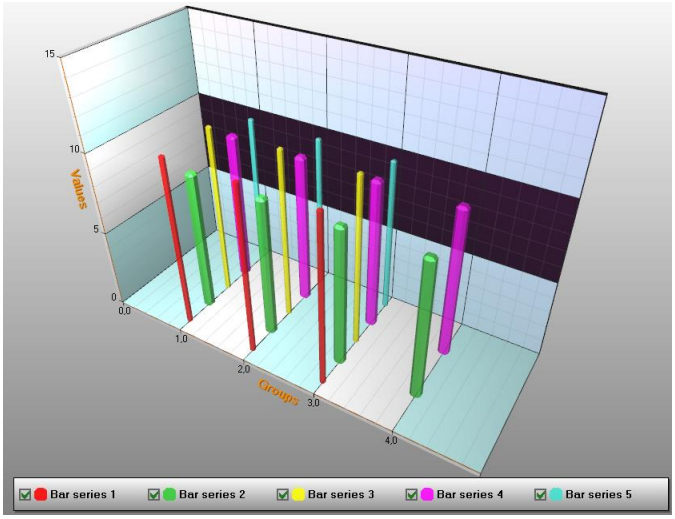


Figure 7-52. BarViewOptions.ViewGrouping = Manhattan. The first series values are shown nearest to camera and the last series farthest. Bar X values control the bar position in X dimension.

7.11.2 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.

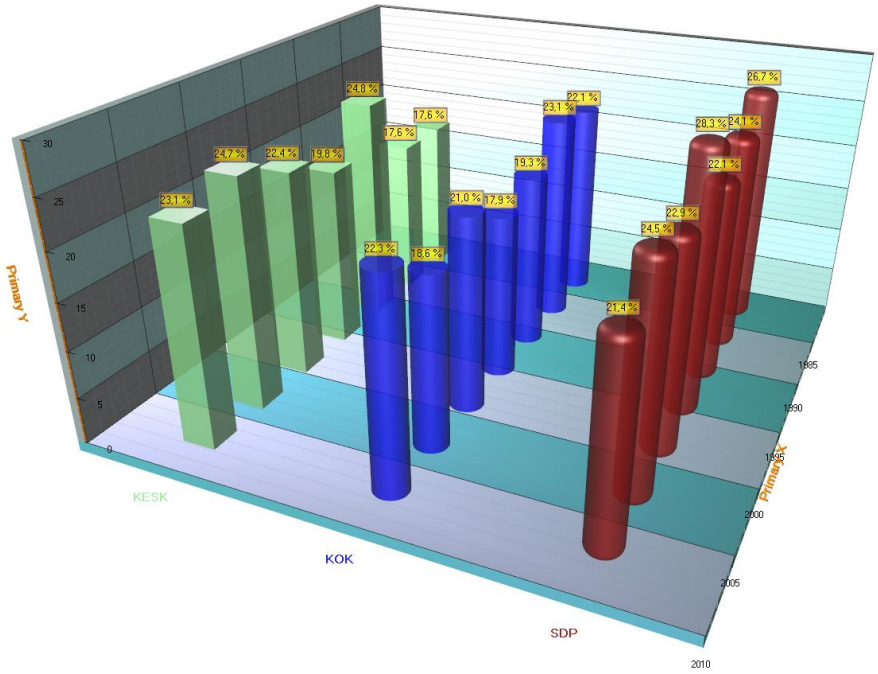


Figure 7-53. Shapes: Simple, Cylinder and RoundedCylinder.

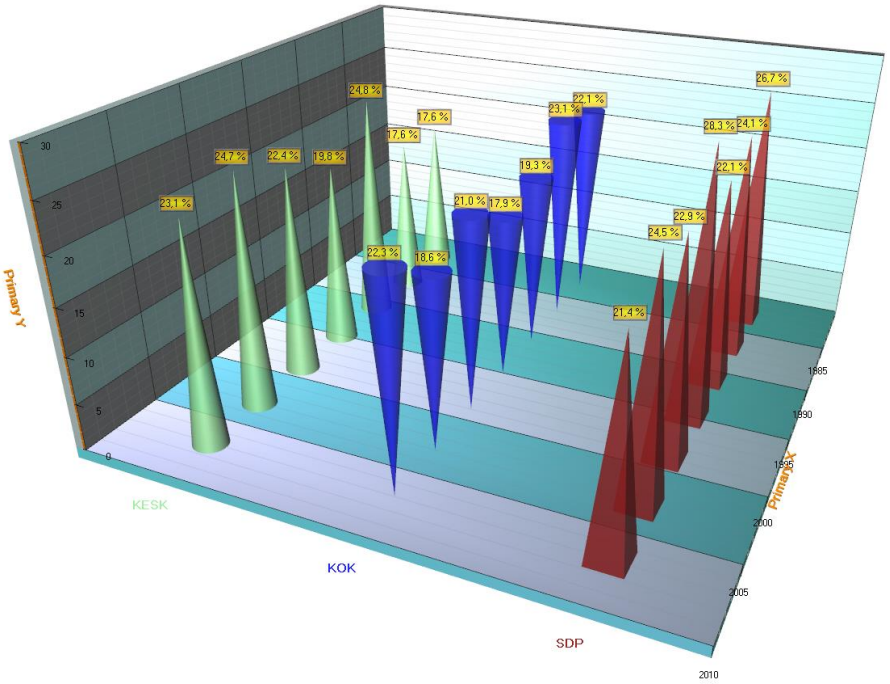


Figure 7-54. Shapes: Cone, ReversedCone and Pyramid.

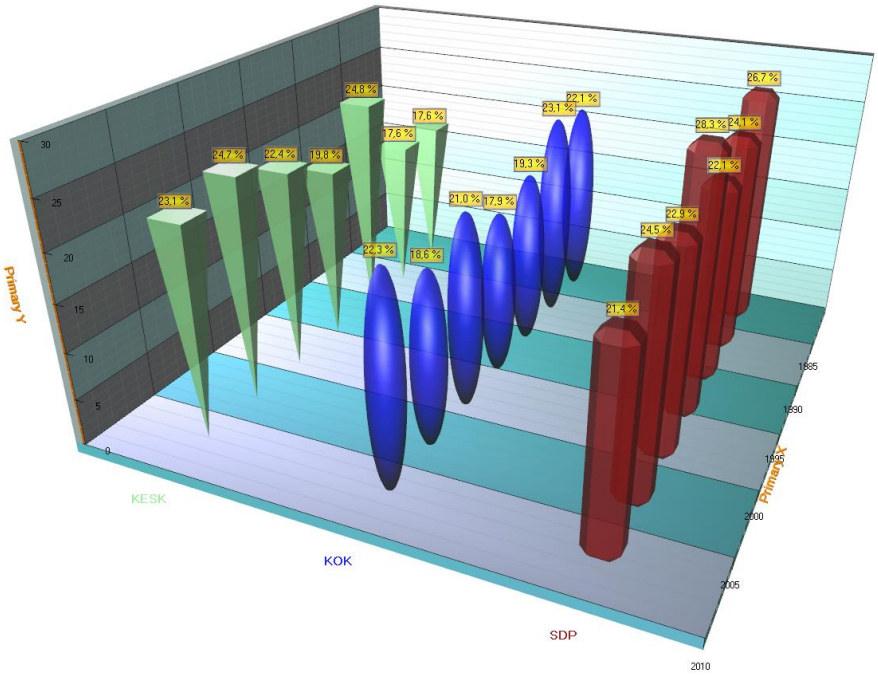


Figure 7-55. Shapes: ReversedPyramid, Ellipsoid and Beveled.

7.11.3 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);
```

7.11.4 Showing bars horizontally

Bars are drawn in Y axis direction. To show the bars vertically, rotate the camera 90 degrees.

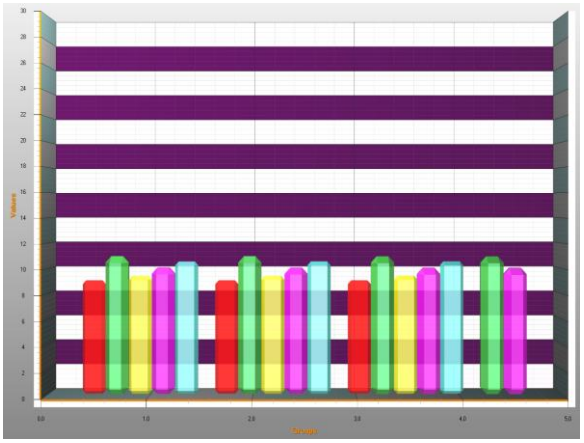


Figure 7-5: Vertical bars view.

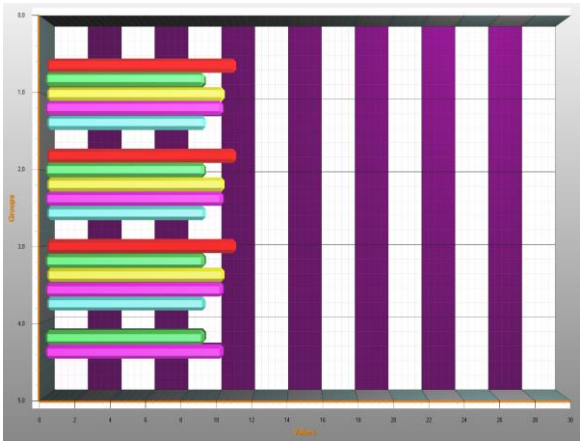


Figure 7-5: Horizontal bars view.

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();
```

7.12 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 general format in 3D modeling applications and game engines. To load a model from file, set the path and file name into ***FileName*** 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.

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.

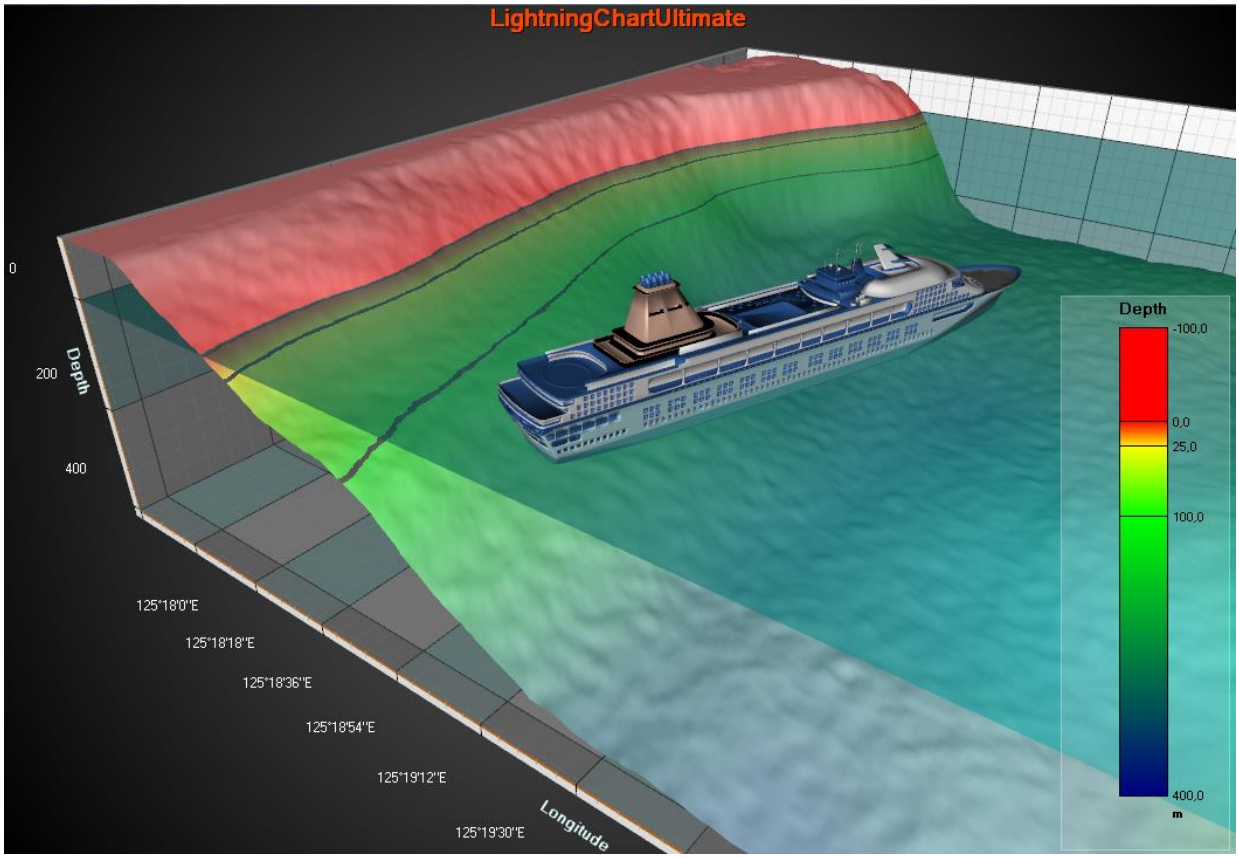


Figure 7-56. Cruise ship loaded into a MeshModel object.

NOTE! LightningChart v.7 onwards does not support Direct3D X-format files (*.x) anymore, since DirectX 11 does not support it.

7.13 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.TileCountWidth*** properties.

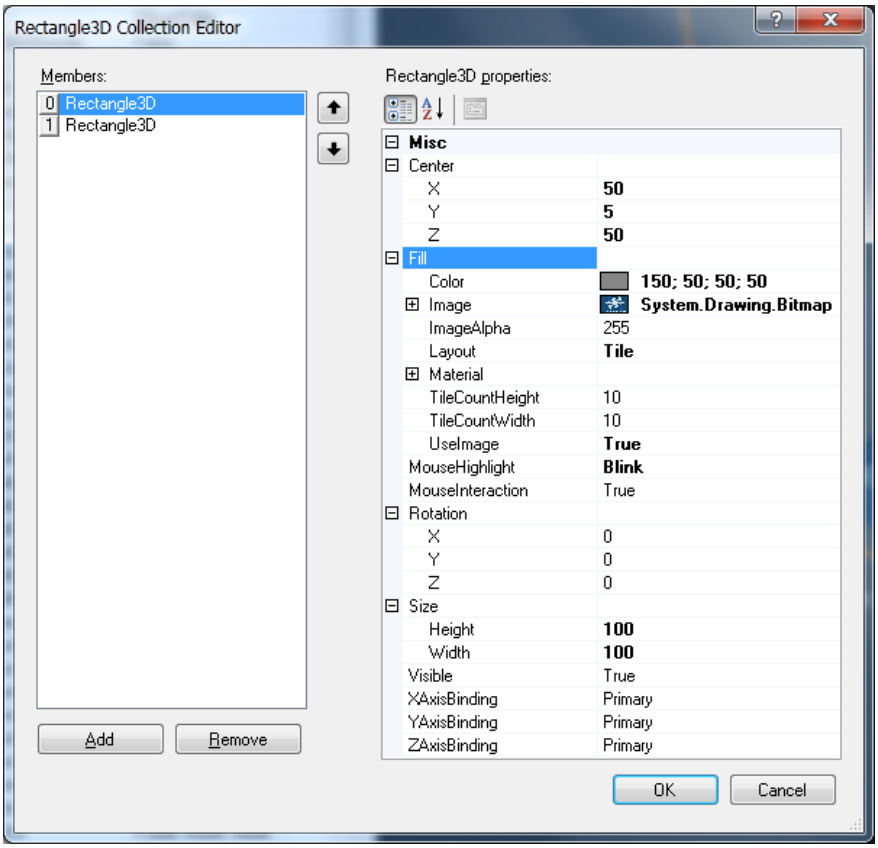


Figure 7-57. Properties of Rectangle3D objects.

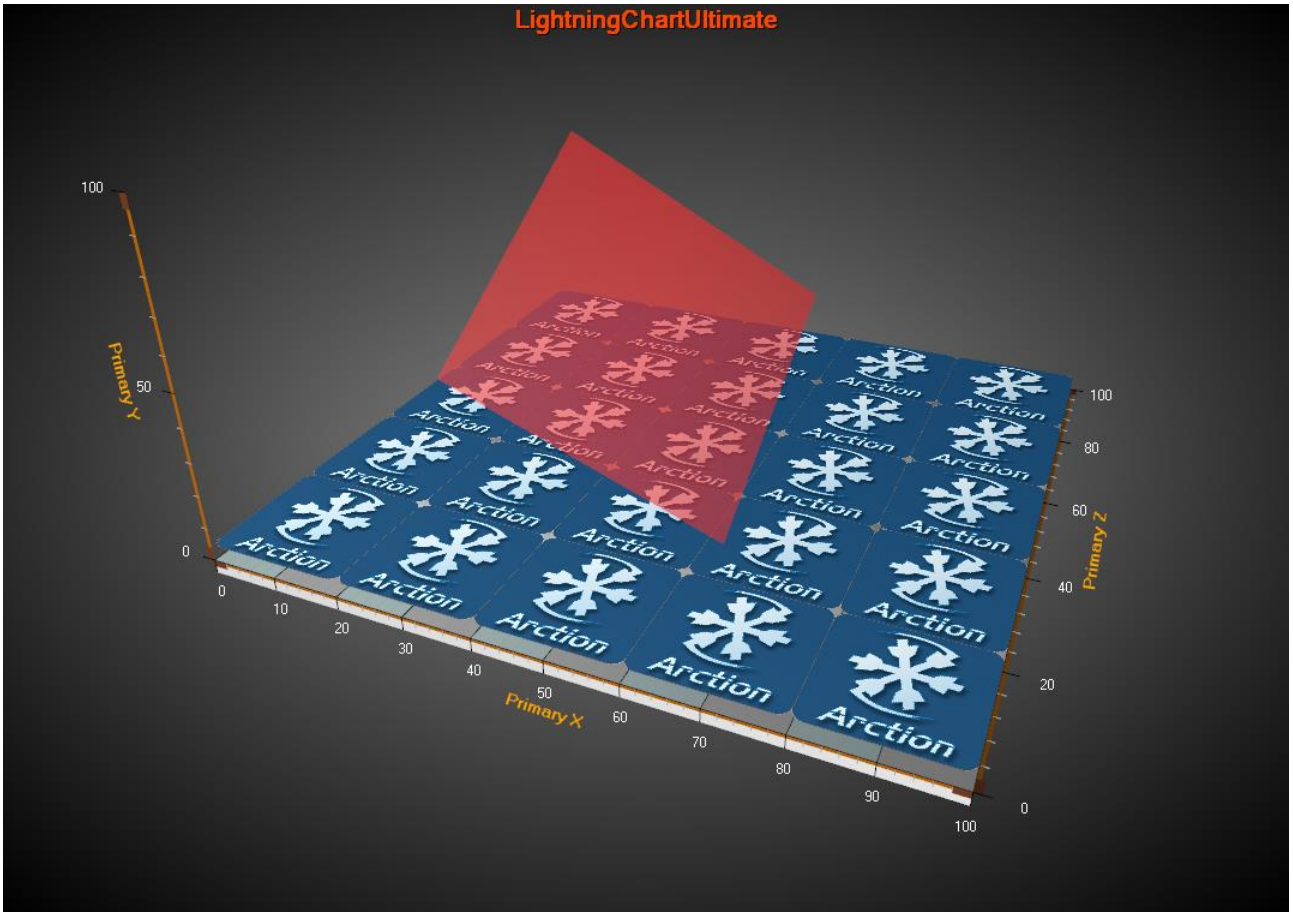


Figure 7-58. Two Rectangle3D objects in View3D. The blue one in the bottom shows a bitmap fill, with Layout = Tile. Red rotated rectangle on the top is configured with translucent color.

7.14 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.

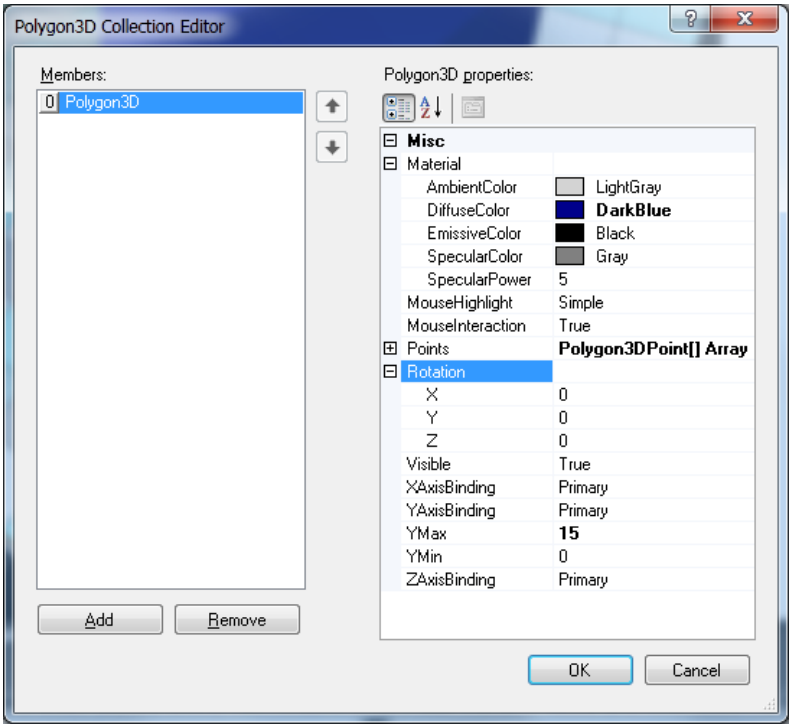


Figure 7-59. Properties of Polygon3D objects.

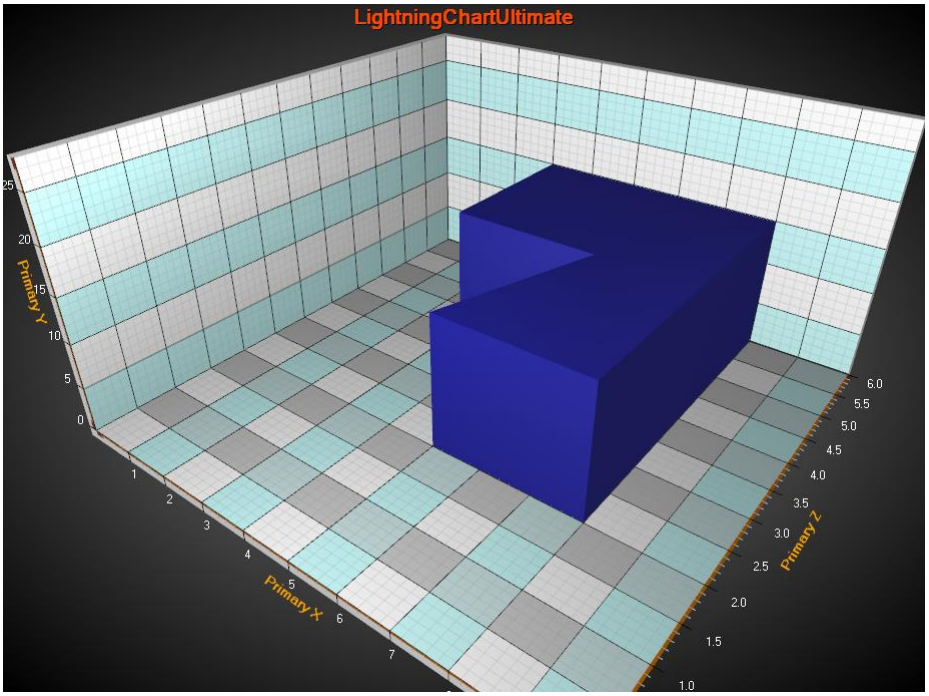


Figure 7-60. A 6-point polygon ranging from YMin = 0, YMax = 15.

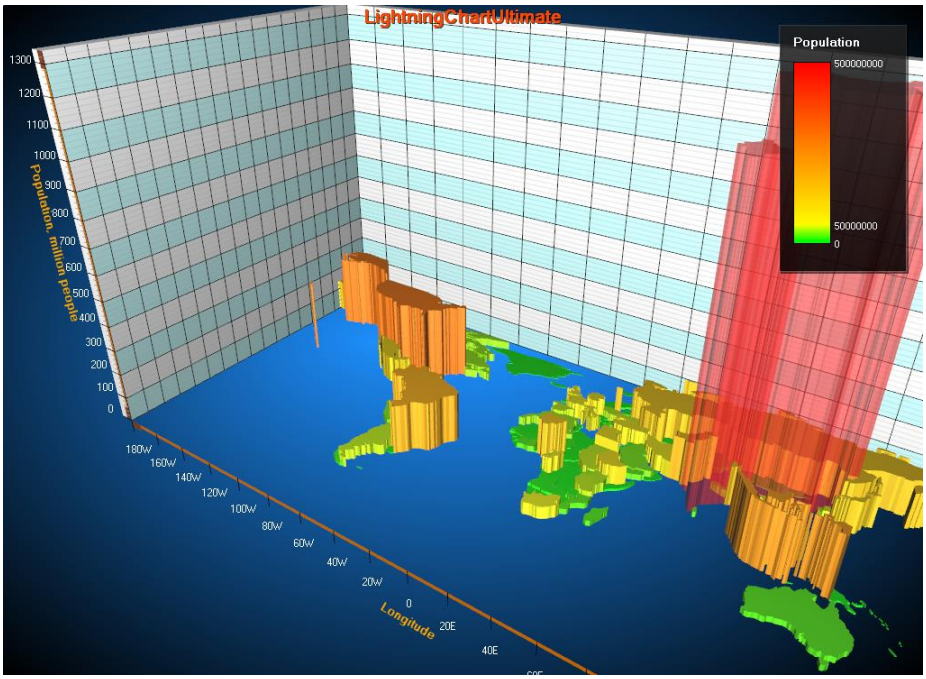


Figure 7-61. World population shown with Polygon3D objects. A Polygon3D object for each region of map data. Population value of a country is used to color the polygon and to set the YMax of it. China and India are shown with translucent colors because of high population of theirs.

7.15 Zooming, panning and rotating

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

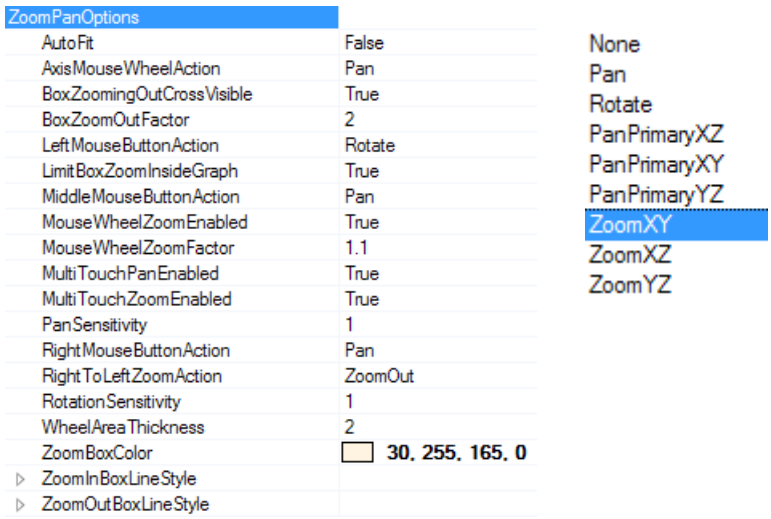


Figure 7-62 ZoomPanOptions properties and sub-properties, with LeftMouseButton / MiddleMouseButtonAction / RightMouseButtonAction options on the right.

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.

7.15.1 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.

7.15.2 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.

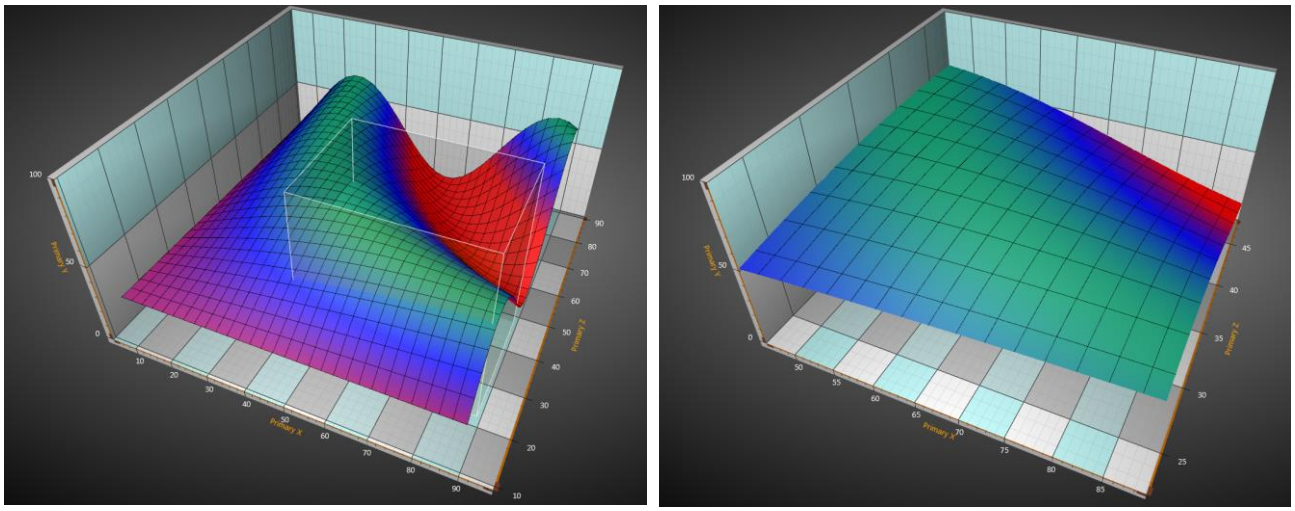


Figure 7-63 XZ-plane box zooming in progress and after mouse button released. X and Z axis ranges are modified, Y axis range not.

Zoom in by dragging box from left to right. The zoomed ranges are applied to axes related to the selected plane.

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***.

7.15.3 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.

7.15.4 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***.

7.15.5 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***.

7.15.6 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.

7.15.7 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.

7.16 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.

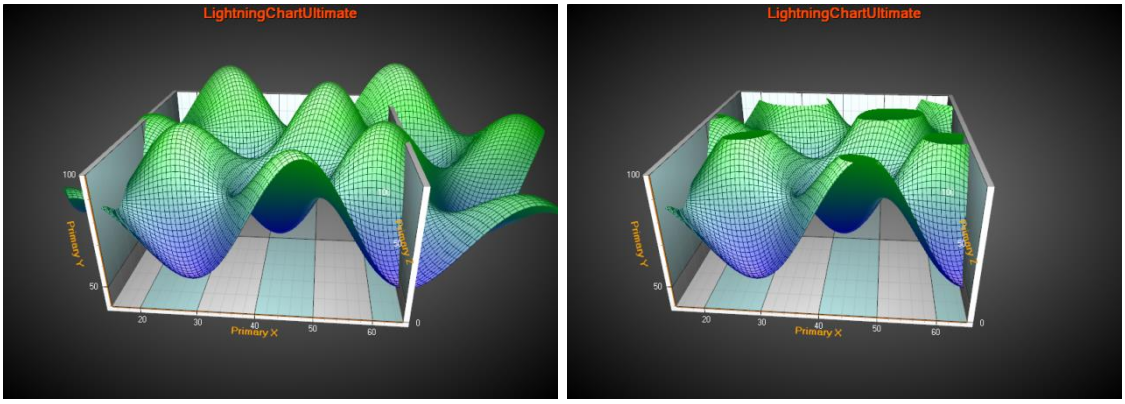


Figure 7-64. On the left, ClipContents is not used. Series render outside axis ranges. On the right, ClipContents is enabled,

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.

7.17 Annotations

Annotations collection allows adding annotations into the 3D scene. In general, they are very similar to ViewXY's ***Annotations*** (see section 6.19), but ***Target*** and ***Location*** control support 3D.

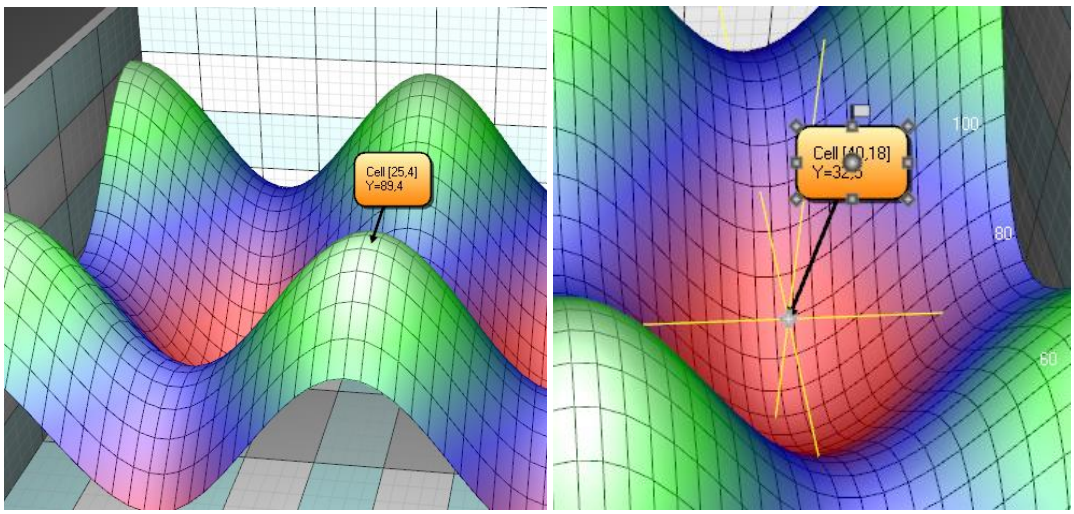


Figure 7-65. Annotation3D object displaying value of 3D series. Crosshair cursor can be used to aid the target movement.

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***.

8. ViewPie3D

ViewPie3D presents data as pie and donut charts, in 3D.

ViewPie3D	**3D pie/donut view**
Annotations	**(Collection)**
▷ Camera	
DonutInnerPercents	50
ExplodePercents	10
▷ LegendBox3DPie	**LegendBoxPie3D**
LightingScheme	DirectionalFromCamera
Lights	**(Collection)**
▷ Material	
Rounding	**40**
StartAngle	0
Style	Pie
Thickness	25
TitlesNumberFormat	**0 USD**
TitlesStyle	**Values**
Values	**(Collection)**
▷ ZoomPanOptions	

Figure 8-1. ViewPie3D object tree.

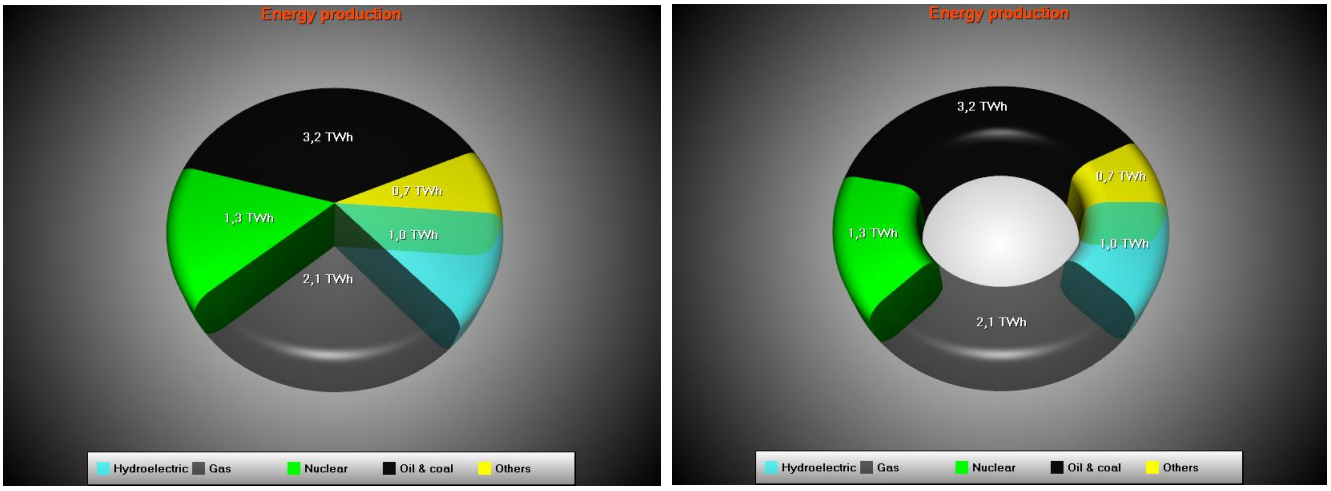


Figure 8-2. Example of Pie and Donut charts.

8.1 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 7).

Camera property controls the viewpoint, see section 7.3. Predefined lighting setup can be selected with ***LightingScheme*** property. Use Material property and its sub-properties to adjust general 3D surface appearance and shininess.

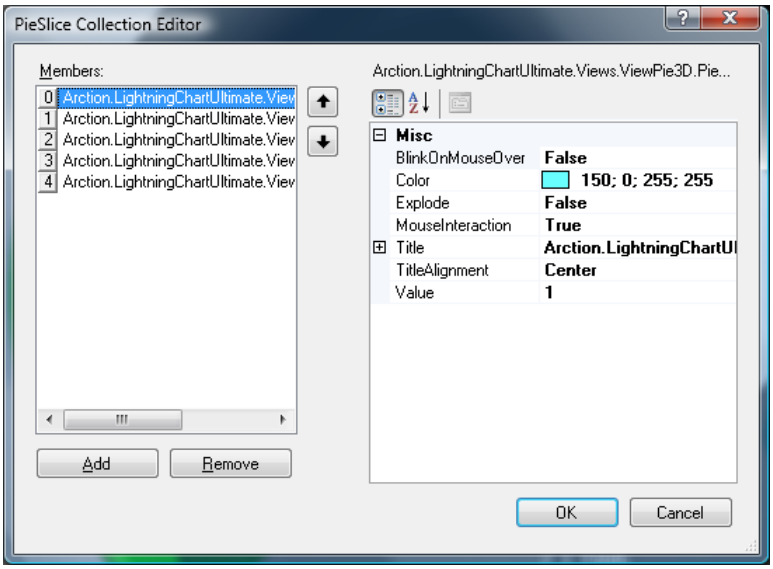
Use ***DonutInnerPercents*** to set the donut inner radius, ***Rounding*** to adjust edge rounding radius, ***StartAngle*** to rotate the pie, and ***Thickness*** to adjust pie thickness. ***ExplodePercents*** adjusts how far away the exploded pie slice is, when slice's ***Explode*** is true.

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 7.17.

8.2 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.

8.3 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;
```

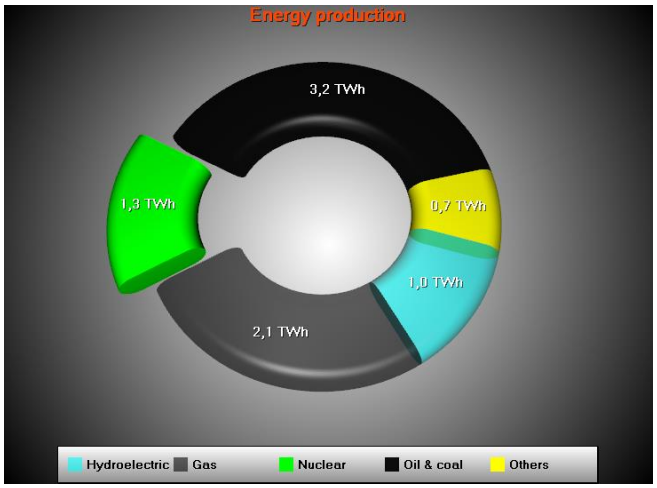
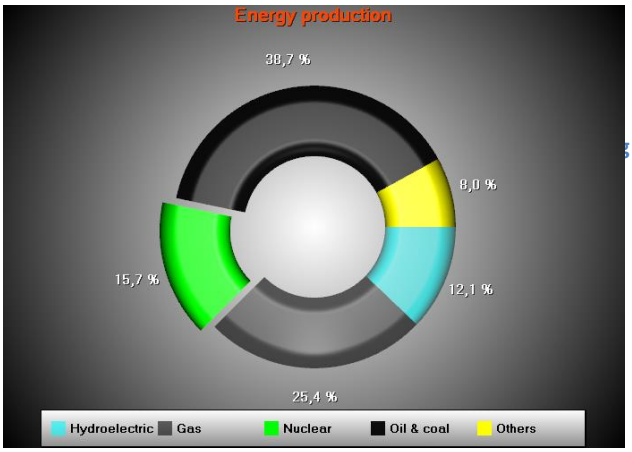


Figure 8-3. Data set into chart. Third slice is separated by using slice3.Explode = true.

8.4 Viewing pie chart in 2D

Set the camera as predefined camera from top.

```
chart.ViewPie3D.Camera.SetPredefinedCamera(PredefinedCamera.PieTop);
```



slice3.Explode = true.

Figure 8-5. Pie chart shown as 2D, with a predefined camera from top.

9. 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.

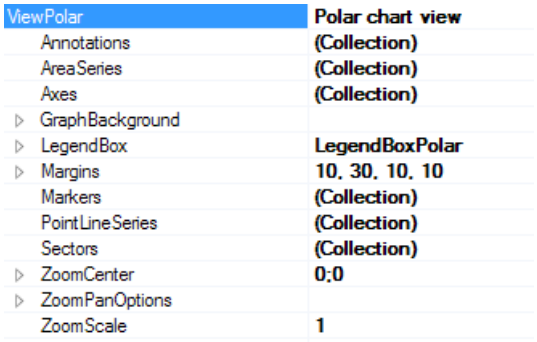


Figure 9-1. ViewPolar object tree.

9.1 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 6.2.

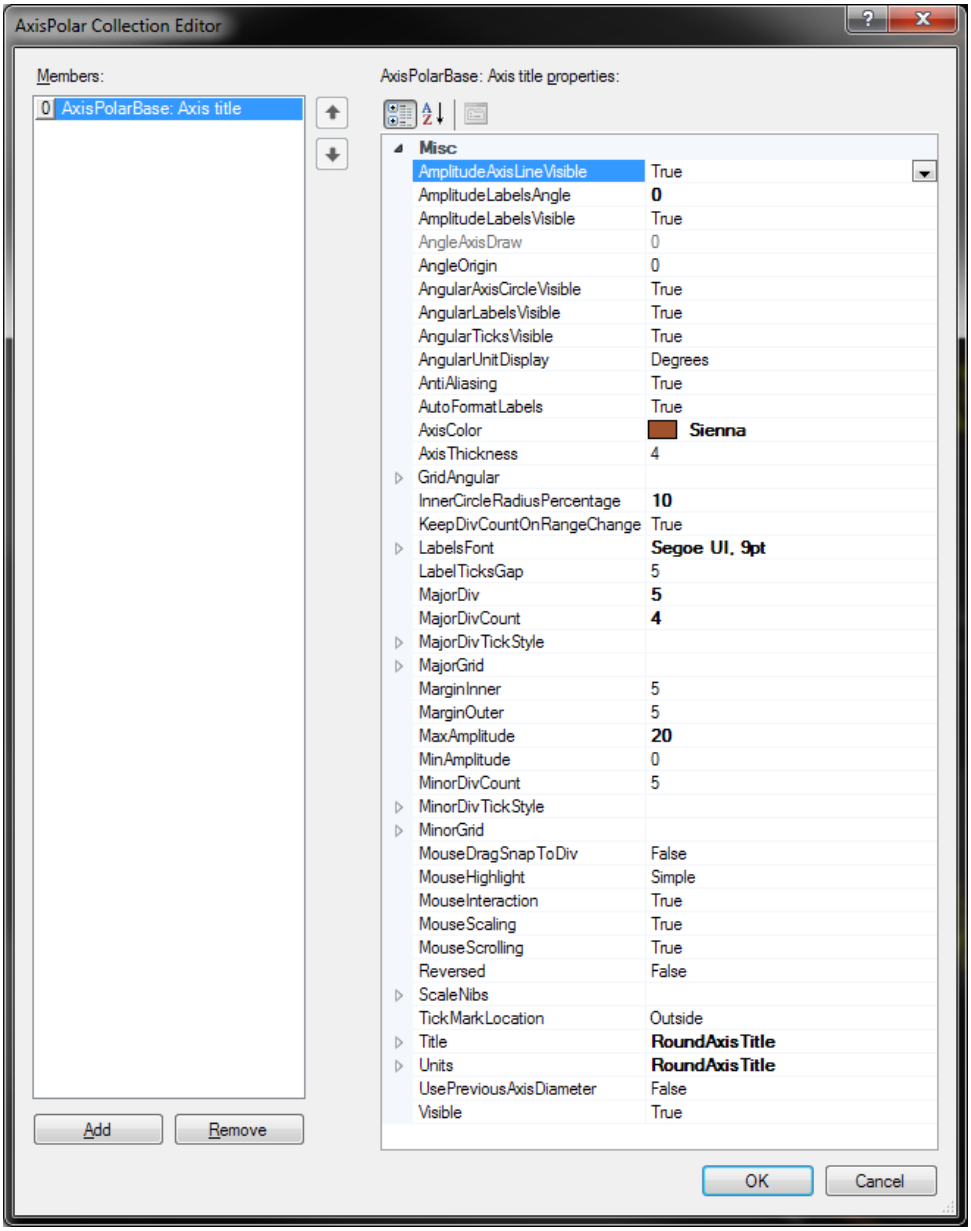


Figure 9-2. AxisPolar property tree

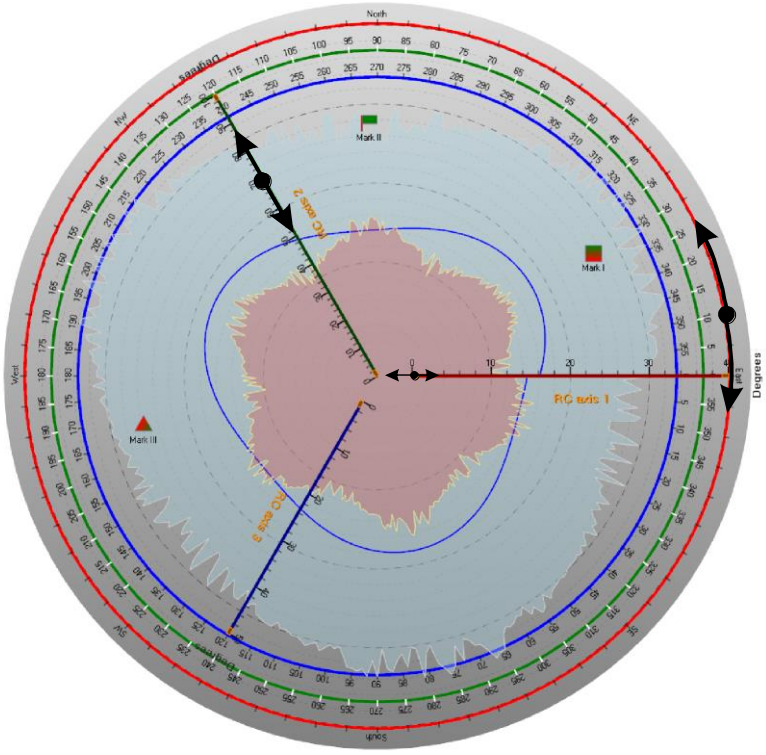


Figure 9-3. Three axes, first (red) in the outer circle, second (green) in the middle, and third (blue) closest to center. Axis AngleOrigin can be changed by dragging it over the axis circle. Amplitude range can be changed by dragging from the axis. Minimum or maximum of axis amplitude range can be changed by dragging from the small nib in the end of the axis.

9.2 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.

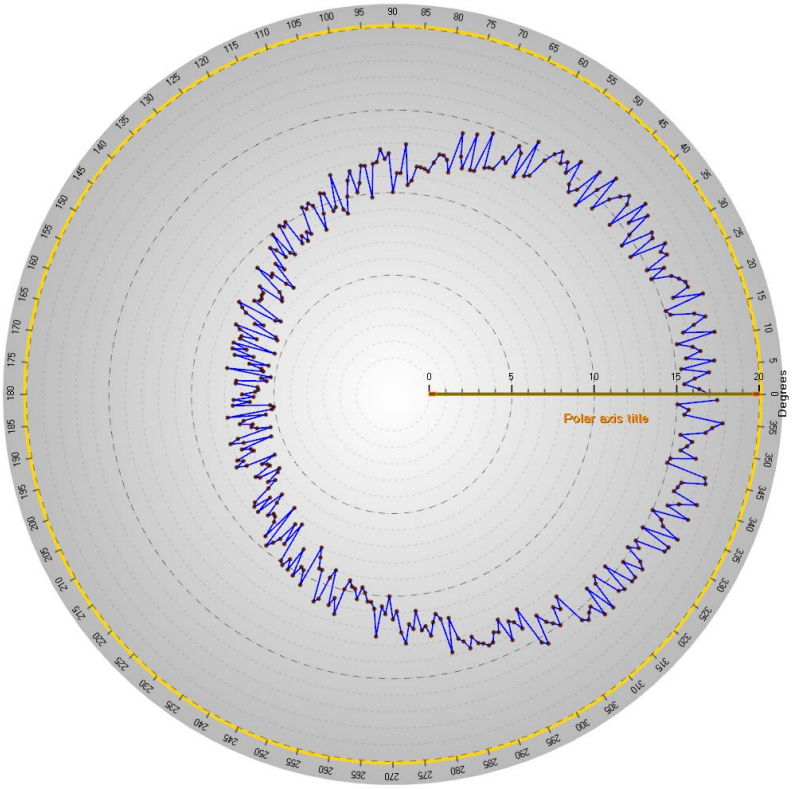


Figure 9-4. Some data presented with ViewPolar's PointLineSeries. Line and points are both visible.

9.2.1 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;
```

9.2.2 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

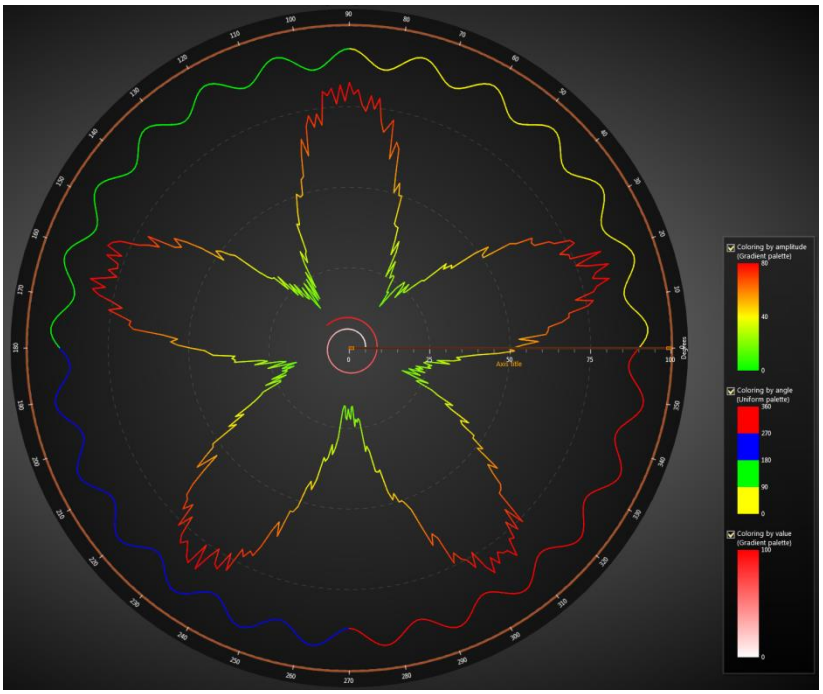


Figure 9-5. Palette coloring applied.

Use ***ValueRangePalette*** property to define the colors and value steps, it works in similar way than with ViewXYs' and View3D's series.

9.3 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.

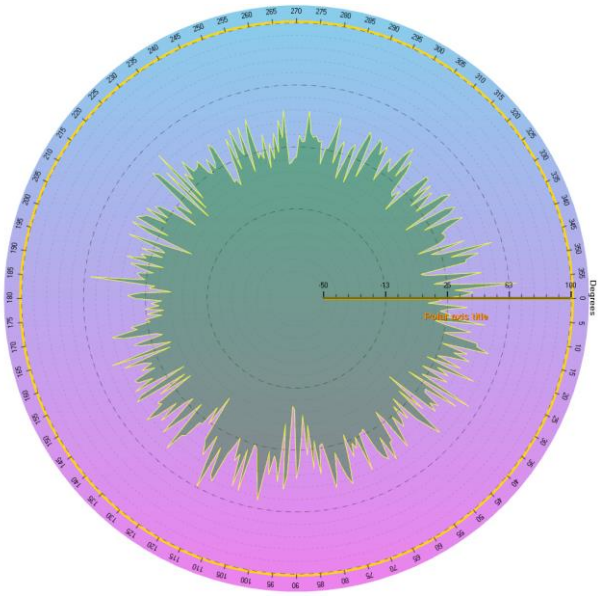


Figure 9-6. Some data presented with ViewPolar's AreaSeries.

9.3.1 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;
```

9.4 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.

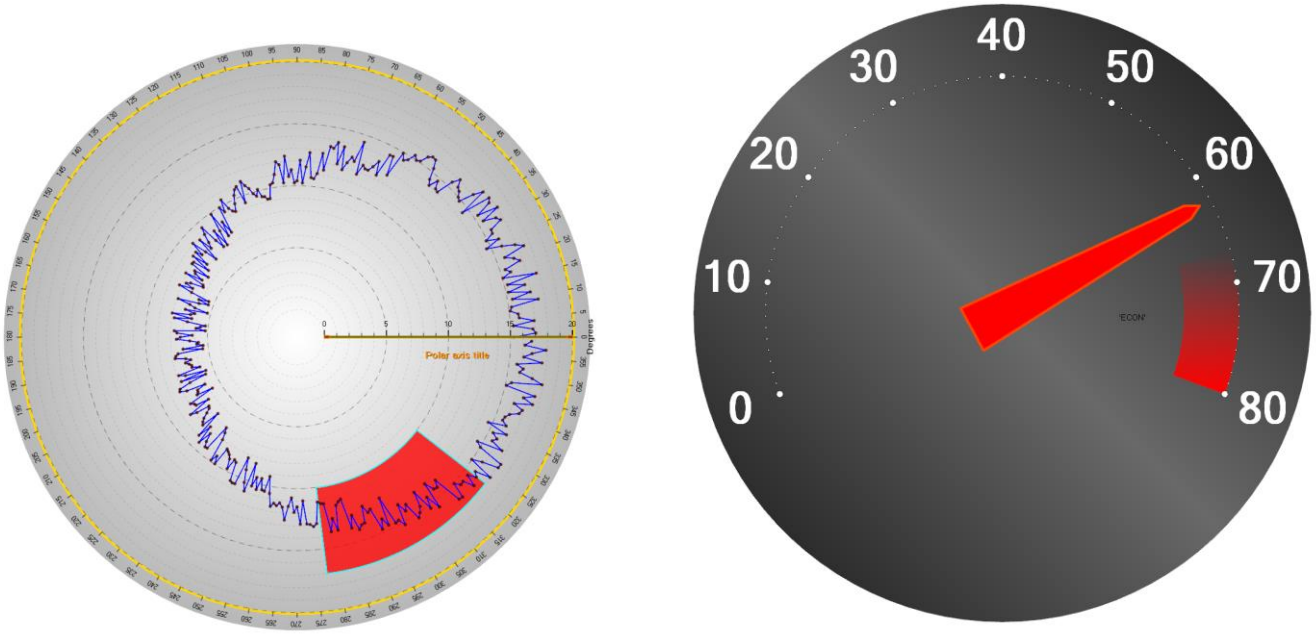


Figure 9-7. A couple of examples sectors are used in. In the first figure, there's a PointLineSeries and sector. In the second figure, a dial is made with AreaSeries and a sector represents RPM meter red zone.

9.5 Annotations

Annotations are almost similar to ViewXY's ***Annotations***, see section 6.19, 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***.

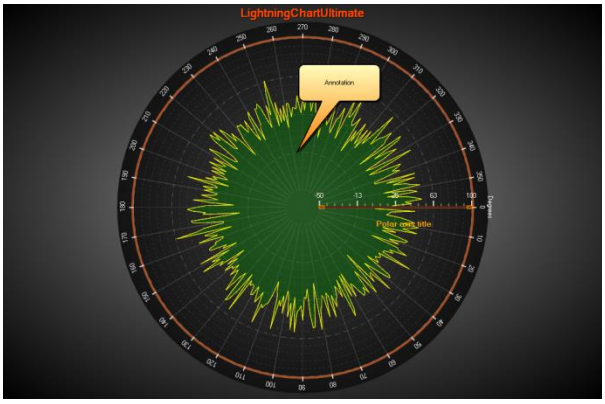


Figure 9-8. An annotation in Polar view.

9.6 Markers

Markers can be used 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 want, 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.

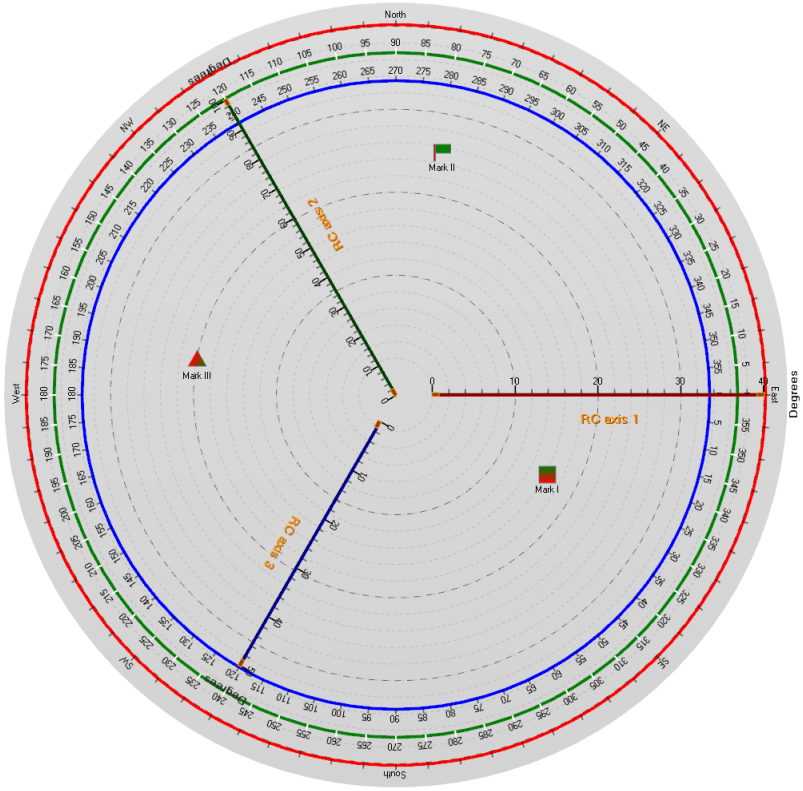


Figure 9-9. A couple of markers in a polar chart.

9.7 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.

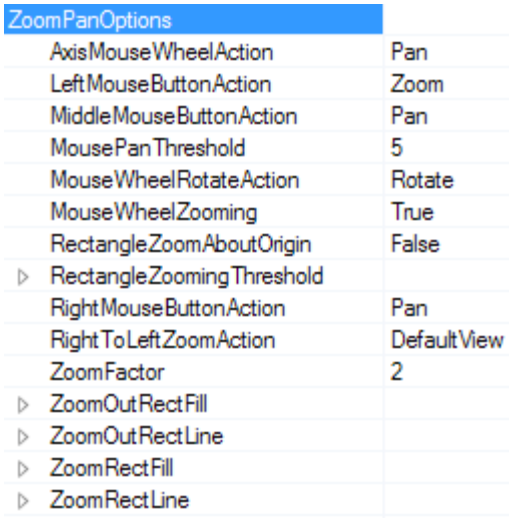


Figure 9-10. ViewPolar's ZoomPanOptions.

10. 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)** .

Terms

Impedance = **Z** = **R** + j**X**

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.

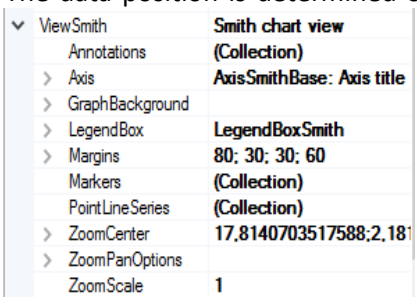


Figure 9-1. ViewSmith property tree.

10.1 Axis

The Smith chart has only one real axis, which can be configured via extended property tree ***Axis***, see figure below.

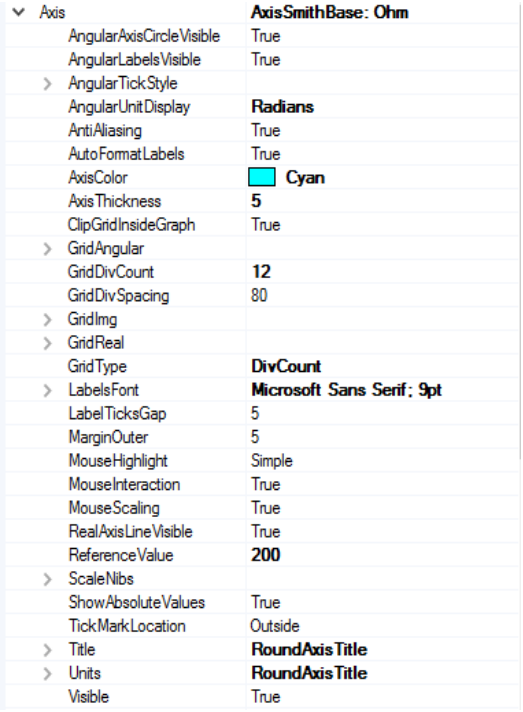


Figure 9-2. AxisSmith property tree.

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.

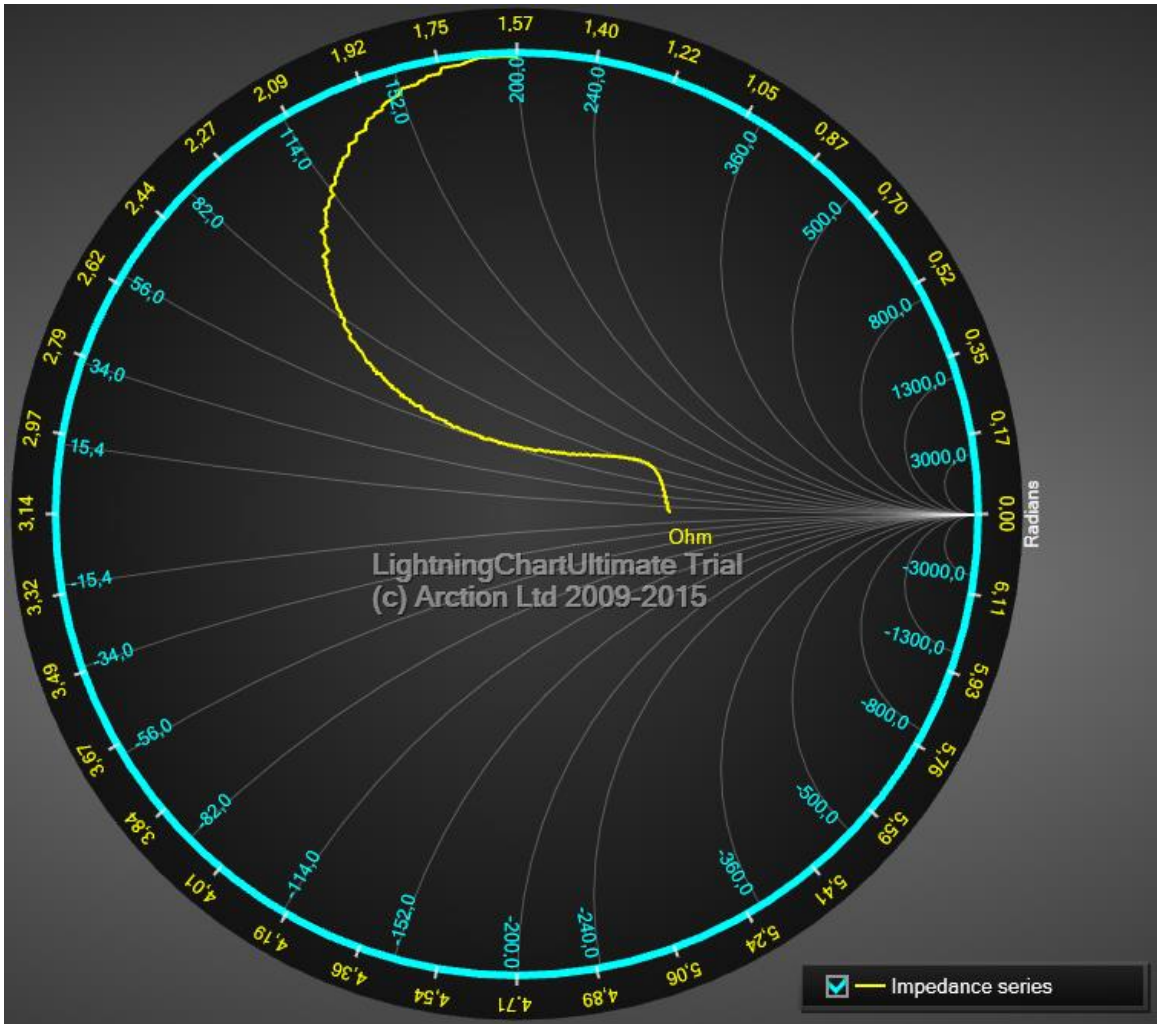


Figure 9-3. Real grid lines are hiden, Imaginary lines are visible.

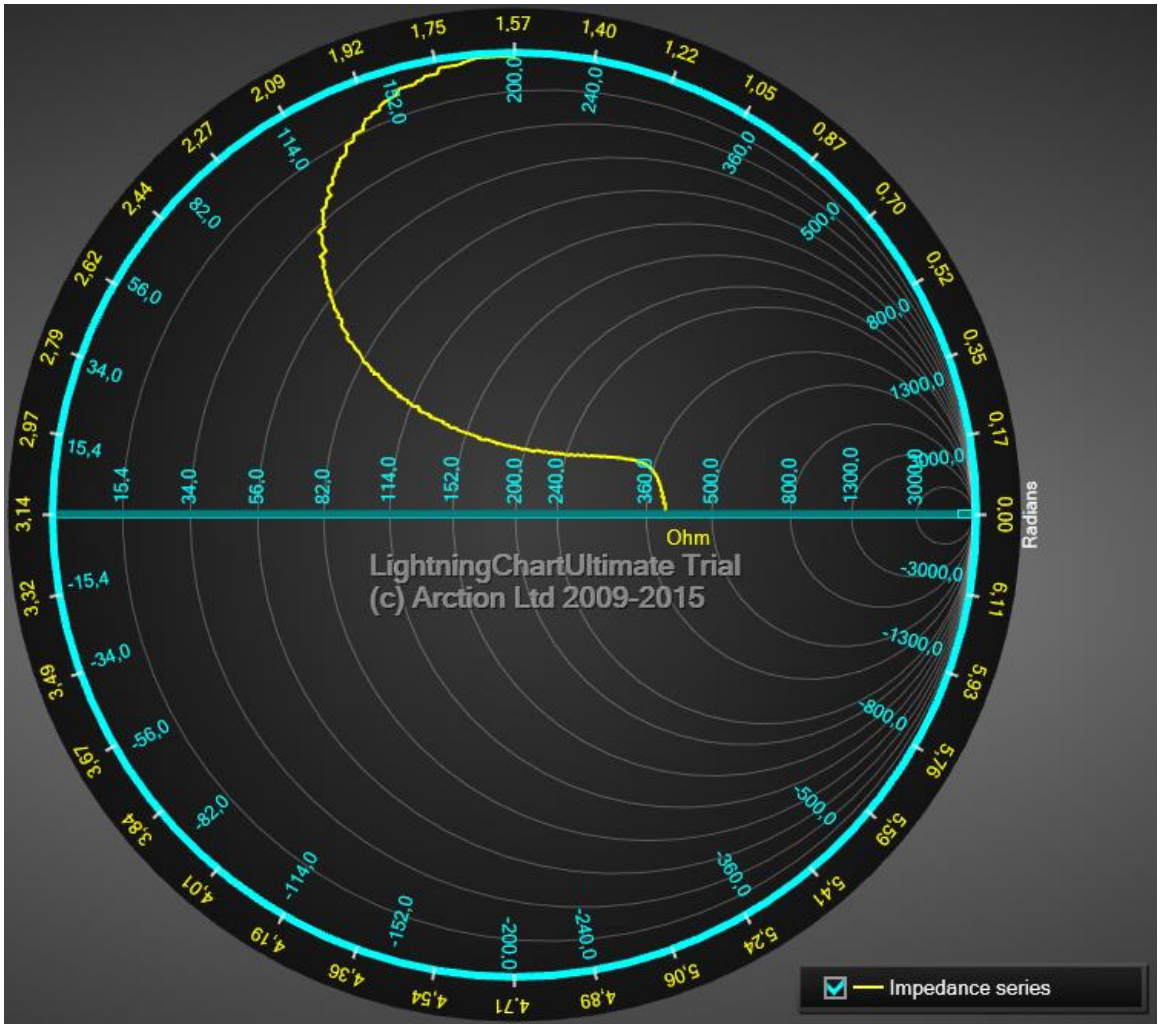


Figure 9-4. Imaginary grid lines are hiden, real lines are visible.

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.

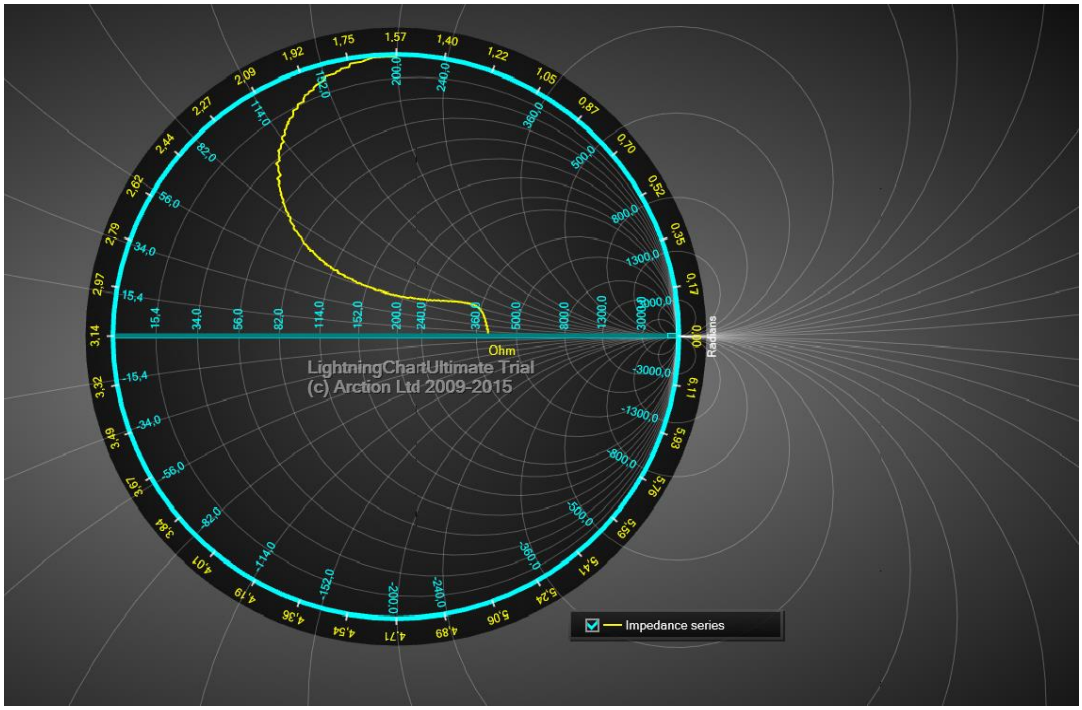


Figure 9-5. ClipGridInsideGraph = False.

The fully customized Smith chart, you can see below.

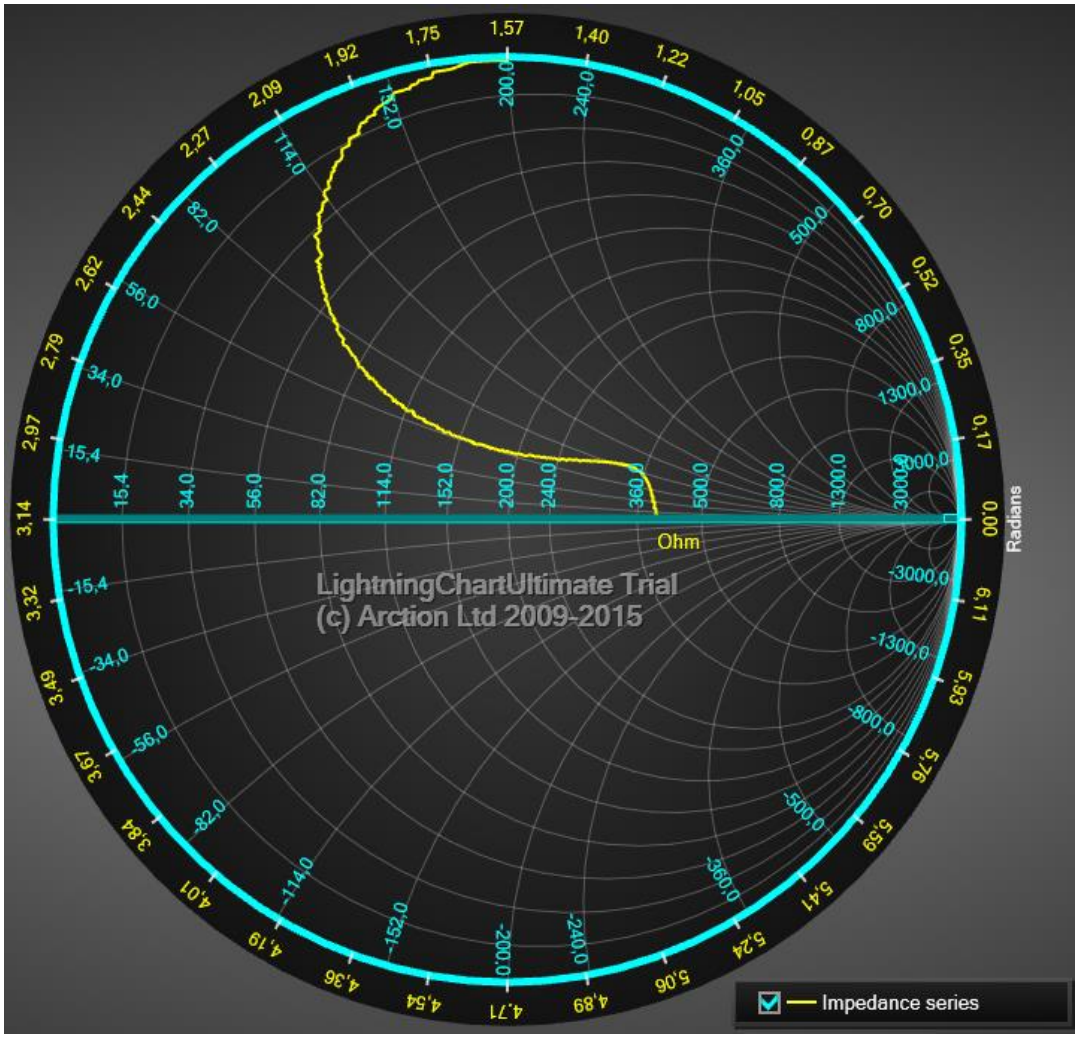


Figure 9-6. Customized Smith chart.

10.2 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.

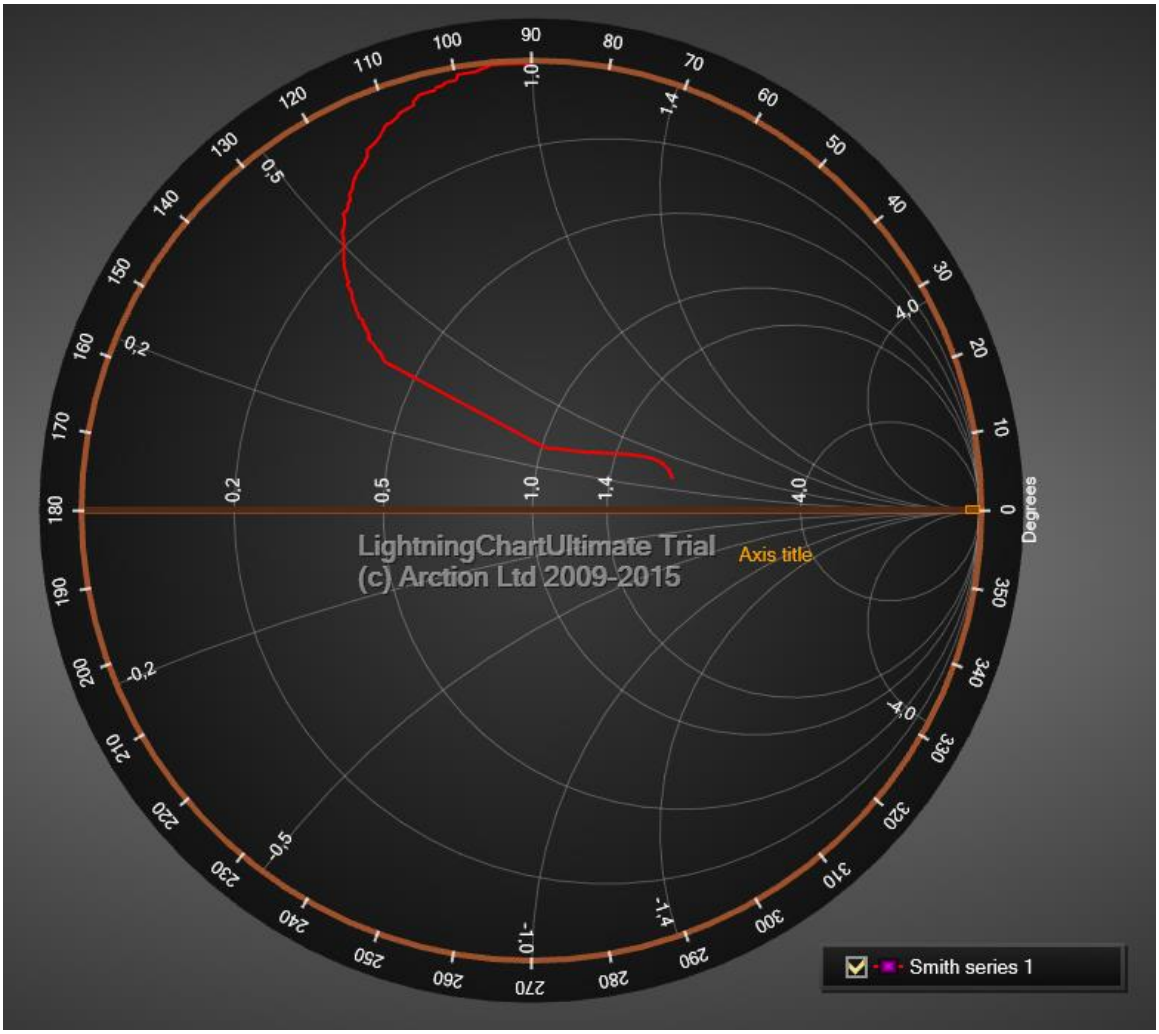


Figure 9-2. Smith data series.

10.3 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);
```

10.4 Annotations

Annotations are identical to ViewPolar's ***Annotations***, see section 9.5. ***Target*** is Location are defined in Polar axis values. ***Sizing*** property has only values ***Automatic*** and ***ScreenCoordinates***.

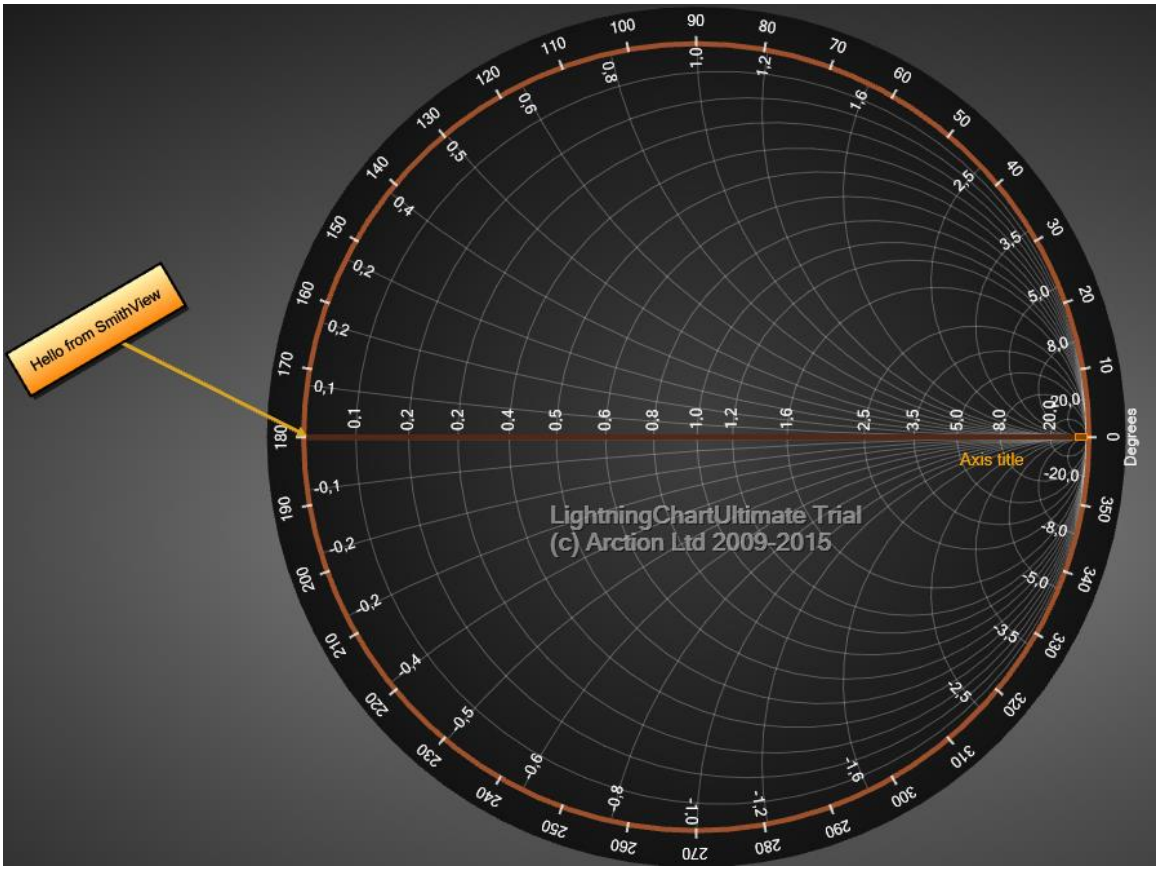


Figure 9-4. An annotation in Smith view.

10.5 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 9.6.

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.

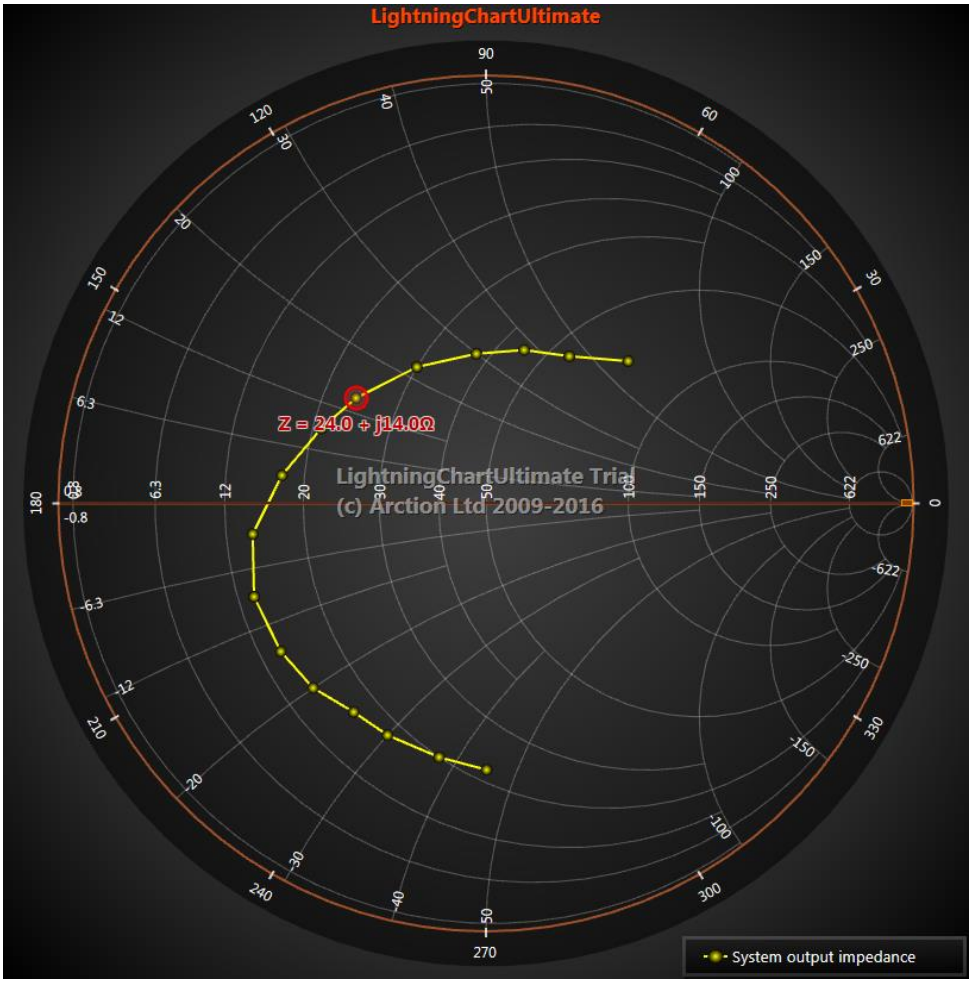


Figure 9-5. A marker tracking a series in Smith view.

11. 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 11-1. Different color themes in use. On the left, default Dark theme with some custom colors. On the right, LightBlue theme set.

12. 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***.

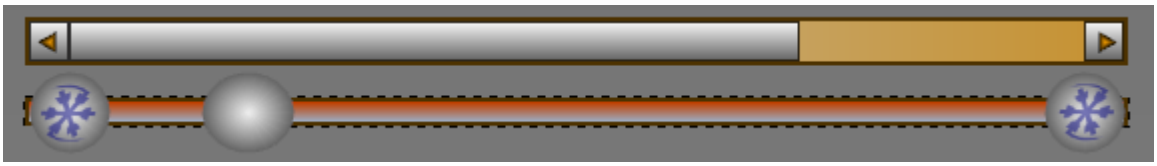


Figure 12-1. A couple of different looking scrollbars

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 repsesents 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 hander provides more information of how the scroll has been done.

13. Export and printing

13.1.1 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.

13.1.2 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 a such a plain color and simple shape. The vector output may also contain some bitmap elements.

13.1.3 Copy to clipboard

The chart can be copied to clipboard by calling ***CopyToClipboard(…)***. ViewXY, ViewPolar and ViewSmith can copied with ***CopyToClipboardAsEmf()*** method in vector format.

13.1.4 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.

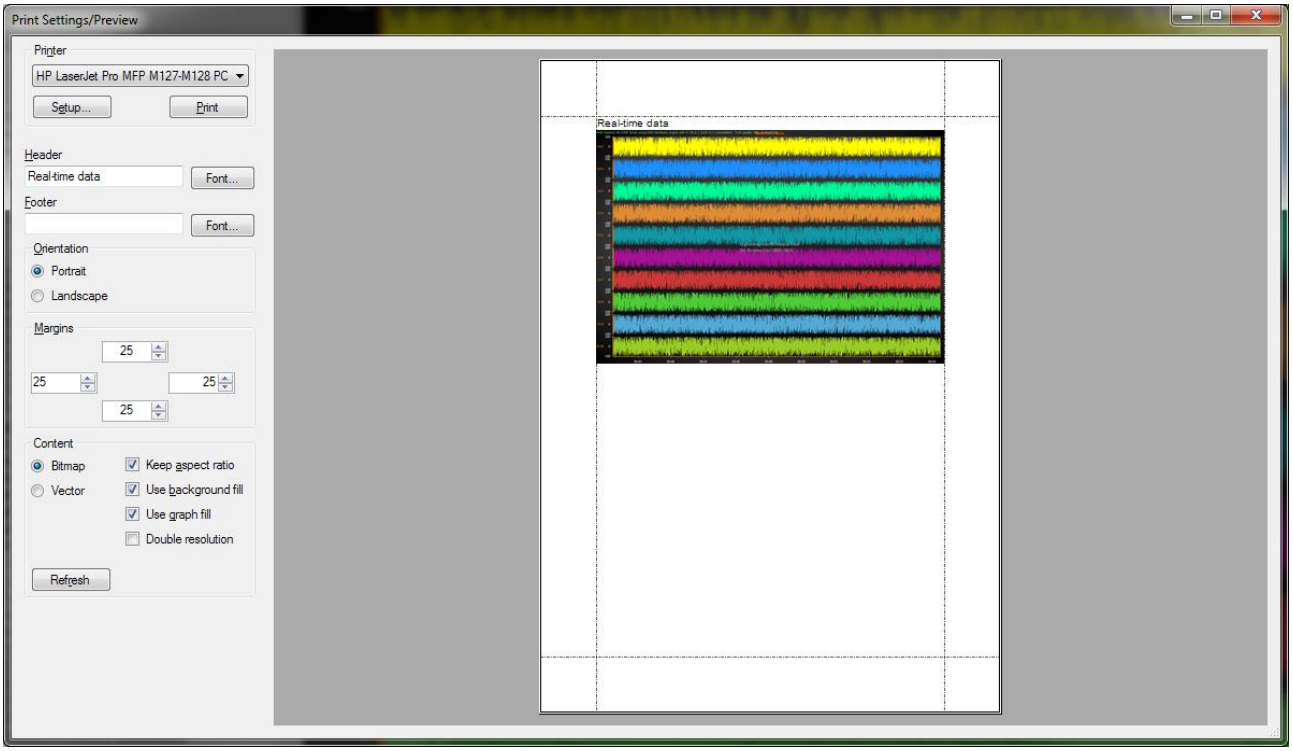


Figure 13-1. Print preview dialog.

14. LightningChart performance

14.1 Selecting the correct API edition

Select the chart edition instructed in section 2.1. Do not use series data binding features if not necessary.

14.2 Set the rendering options correctly

LightningChart's DirectX9 rendering engine is faster than DirectX11 engine, especially in WPF. DirectX11 in the other hand, gives better appearance. Prefer DirectX9 for maximum performance.

Fonts quality setting is important as well.

See section 5.6.

14.3 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. ***EndUpdate()*** redraws the control. If you don't want to redraw the control with ***EndUpdate()*** call, use ***EndUpdate(false)***. 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
```

14.4 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 ***MouseInteraction*** to false.

14.5 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.

14.6 3D surface series tips

Applies to: ***SurfaceGridSeries3D***, ***SurfaceMeshSeries3D***, ***WaterfallSeries3D***

- 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.
- 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.

14.7 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.

14.8 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.

15. LightningChart error and exception handling

Enable ***ThrowChartErrors*** 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.

16. ChartManager component

Add ChartManager control to your form. Then, assign the manager control to ***ChartManager*** property of all LightningChartUltimate controls.

16.1 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.

16.2 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.

17. 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.

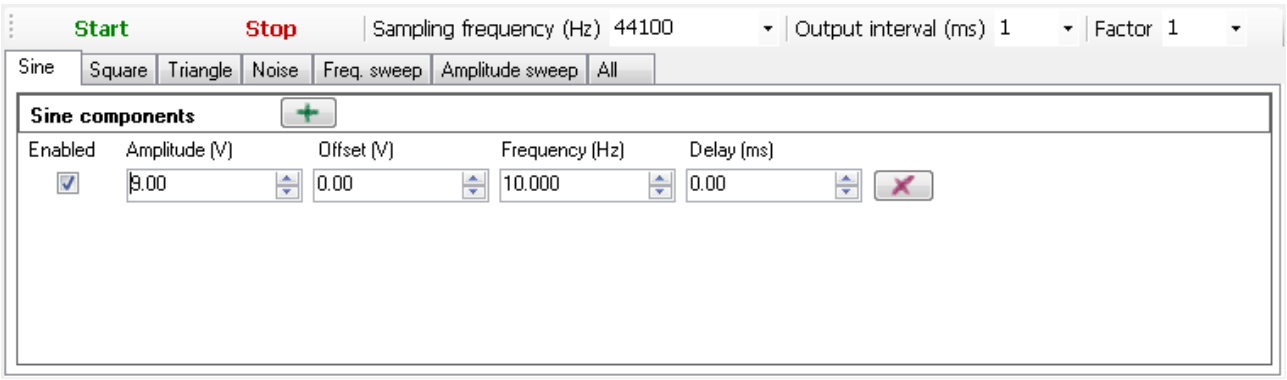


Figure 17-1. Signal generator component with Sine page selected.

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.

17.1 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.

17.2 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 * ***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

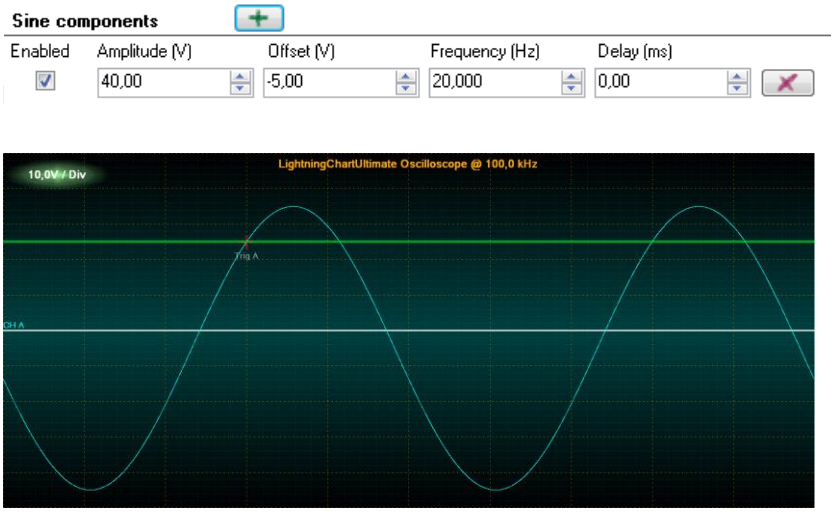
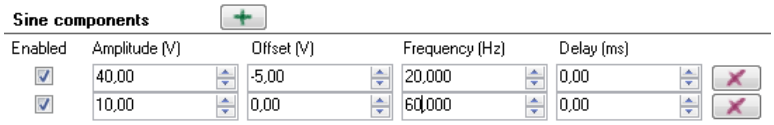


Figure 17-2. One sine waveform signal.

Respectively, two sine waveforms with the following settings



produces signal like the following figure.

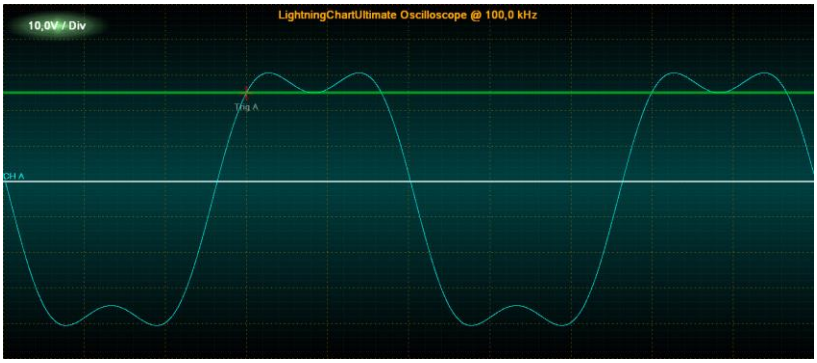
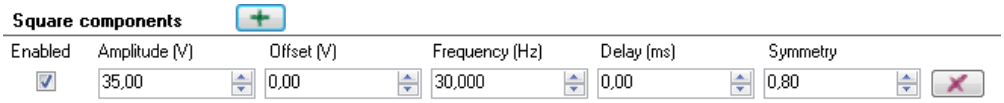


Figure 0-3. Two sine waveforms signal.

17.3 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



produces a signal like the following.

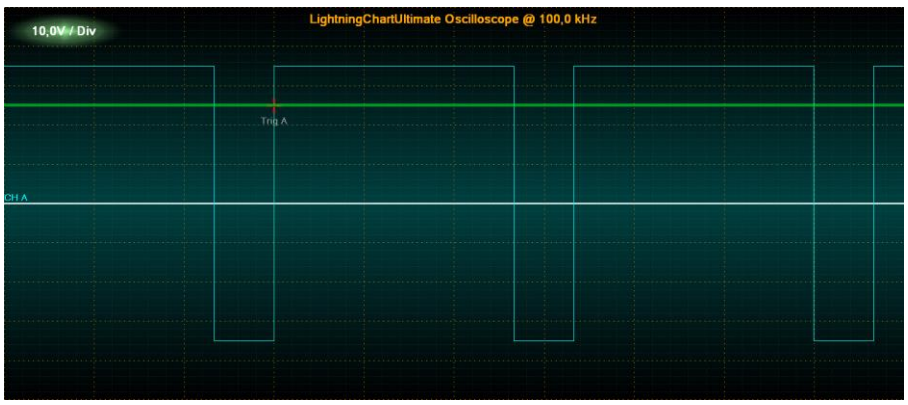
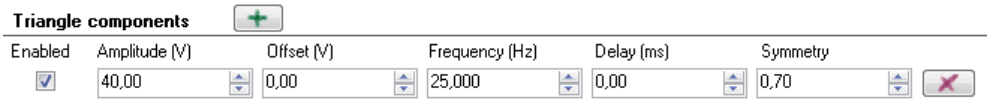


Figure 0-4. One square waveform signal, with symmetry of 0.8.

17.4 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



produces a signal like the following

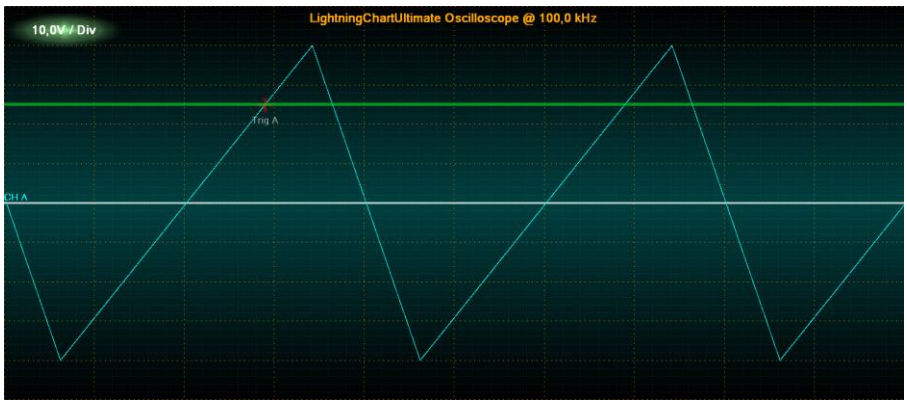


Figure 15-4. One triangle waveform signal, with symmetry of 0.7.

17.5 Noise waveforms

Noise waveform is a randomly generated signal. Points get randomized between –Amplitude and +Amplitude.

Noise waveform with the following settings

produces a signal like the following

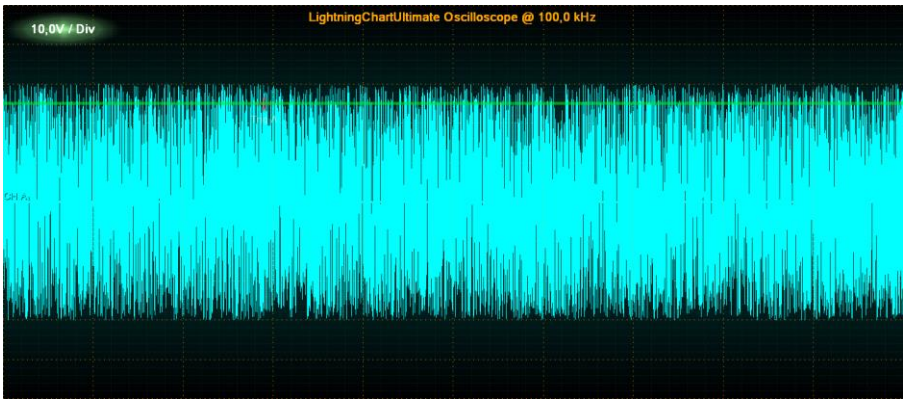


Figure 15-6. Noise waveform signal.

17.6 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.

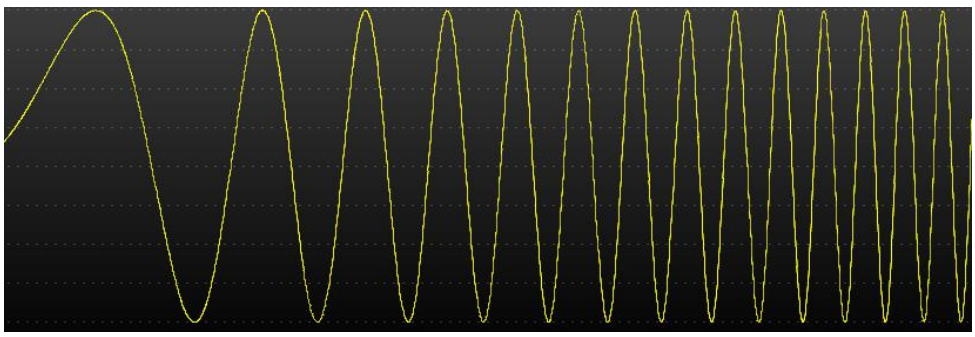


Figure 15-6. Frequency sweep.

17.7 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.

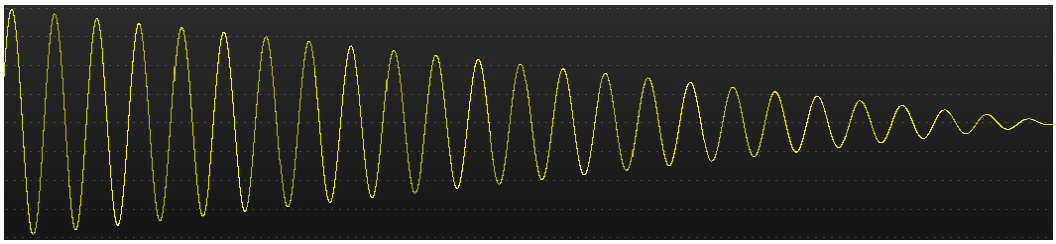


Figure 15-6. Amplitude sweep.

17.8 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.

17.9 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.

17.10 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.

18. 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 17.10.

SignalReader component currently supports a couple of formats, wav and sid.

18.1 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 17.1.

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.

18.2 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.

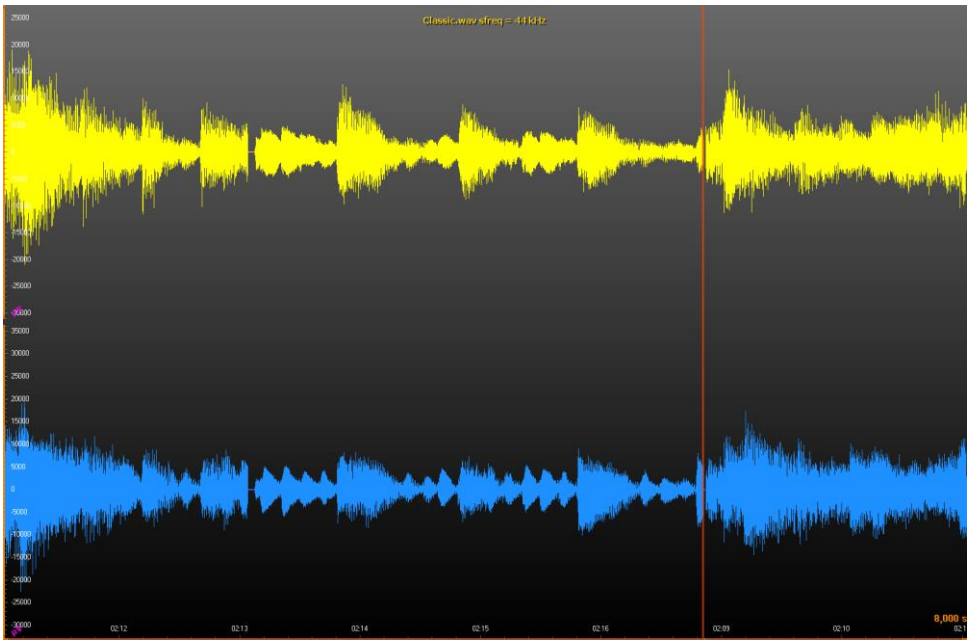


Figure 18-1. SignalReader reads a wav file and LightningChart SampleDataSeries draw the signal. A cursor line is used to mark current reading position and X-axis scroll position.

19. 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…

19.1 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.

19.2 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.

19.3 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.

19.4 Usage (WinForms)

This chapter describes the usage of WinForms version of AudioInput class. WPF version will be handled in chapter 17.5.

19.4.1 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.

19.4.2 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.

19.4.3 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.

19.4.4 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.

19.4.5 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.

19.5 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.

19.5.1 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.

20. 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.

20.1 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.

21. SpectrumCalculator component

SpectrumCalculator component allows conversion between time domain and frequency domain.

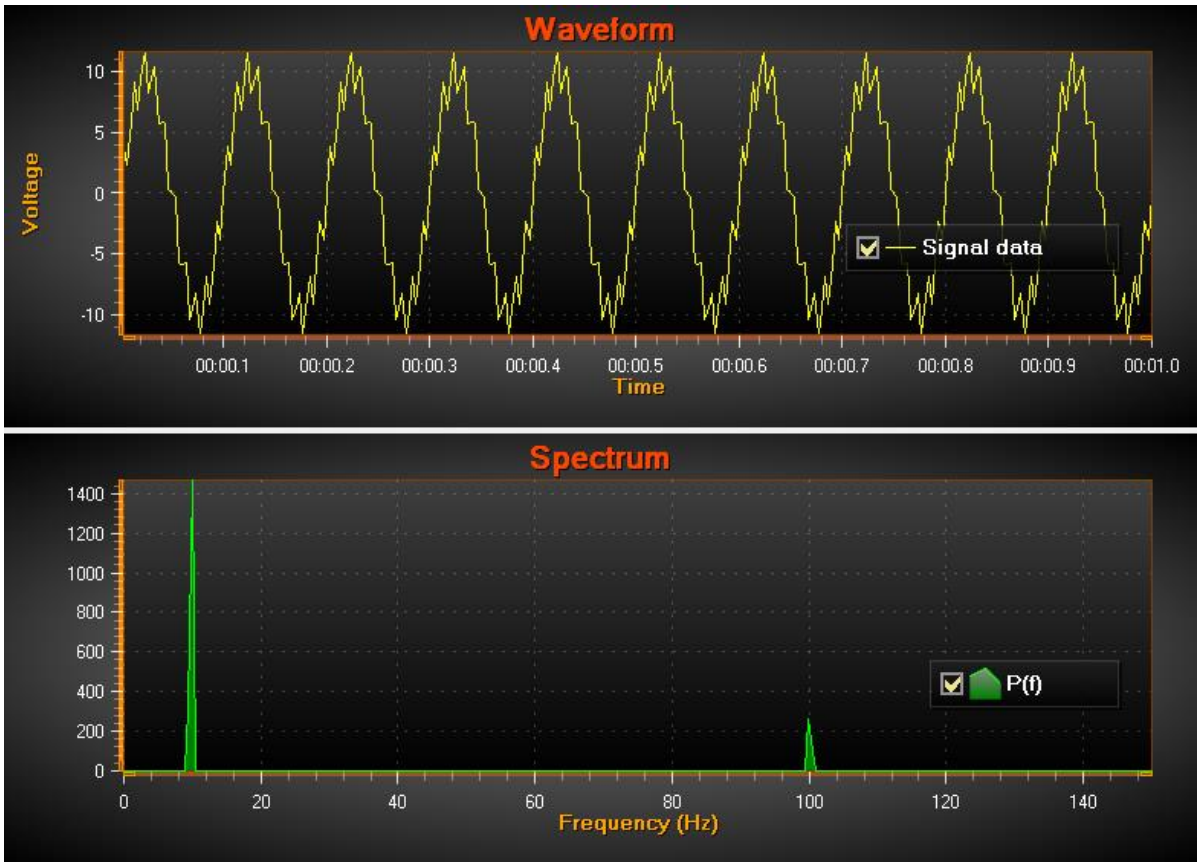


Figure 21-1. Example of source signal data (top) converted to frequency domain (bottom). Signal sampling frequency = 300 Hz, thus frequency scale is 300/2 = 150 Hz. The strong sine base line is 10 Hz (10 cycles / sec). Smaller signal of 100 Hz is added as noise. Both spikes are found in the power spectrum.

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)*

22. 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. Open it from *Project -> Properties* menu, and open ***Configuration Properties, General page***.

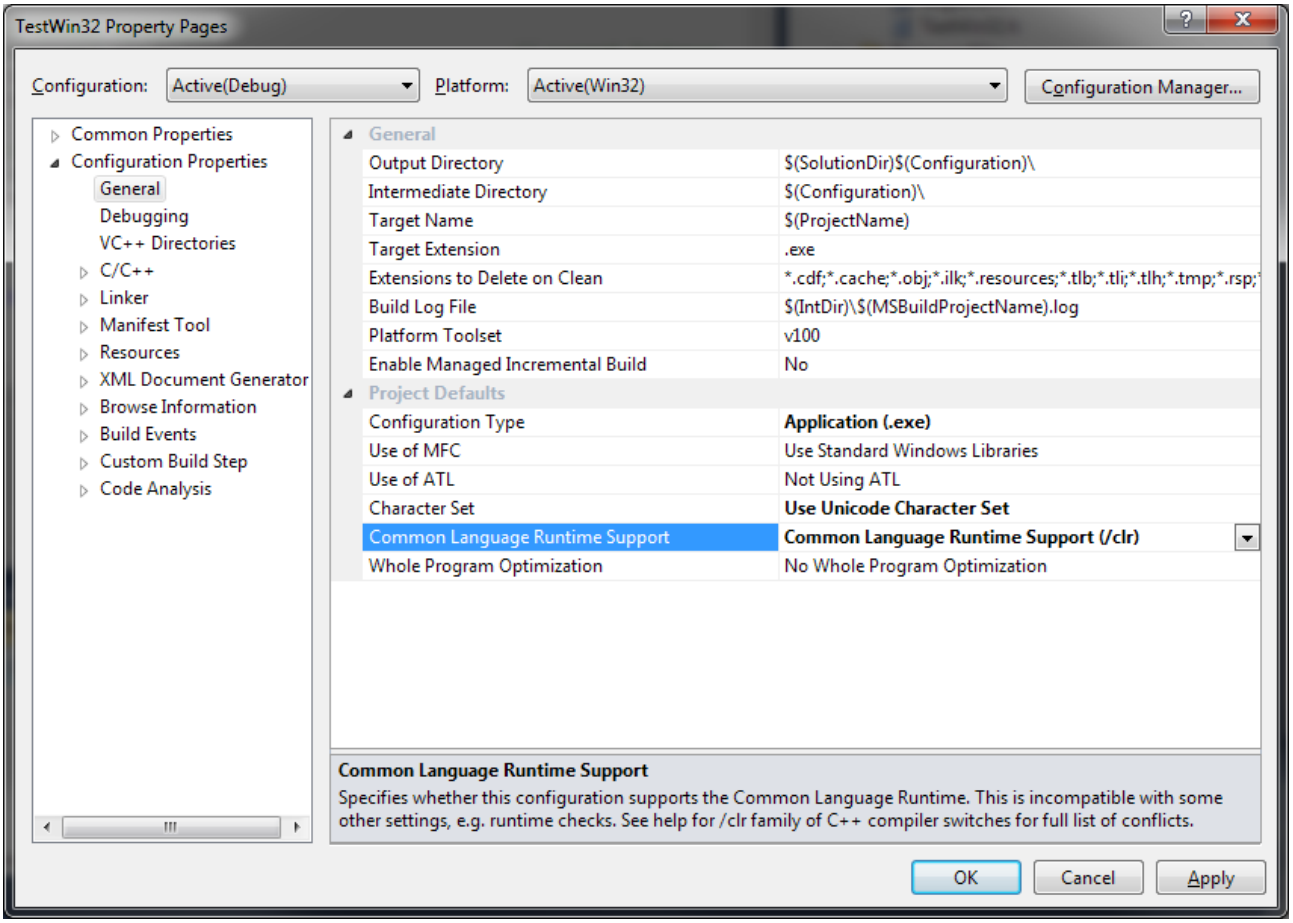


Figure 22-1 C++ application property pages in Visual Studio 2010. Setting of Common Language Runtime support.

The .NET library is now accessible with C++/CLI syntax, C++ with .NET extensions.

Add the .NET assembly references in property pages, ***Common properties, Framework and References***. Add the following .NET assemblies:

- **Arction.WinForms.Charting.LightningChartUltimate**(found in install folder or install folder -> LibNET4)
- **System**(framework assembly)
- **System.Windows.Forms** (framework assembly)

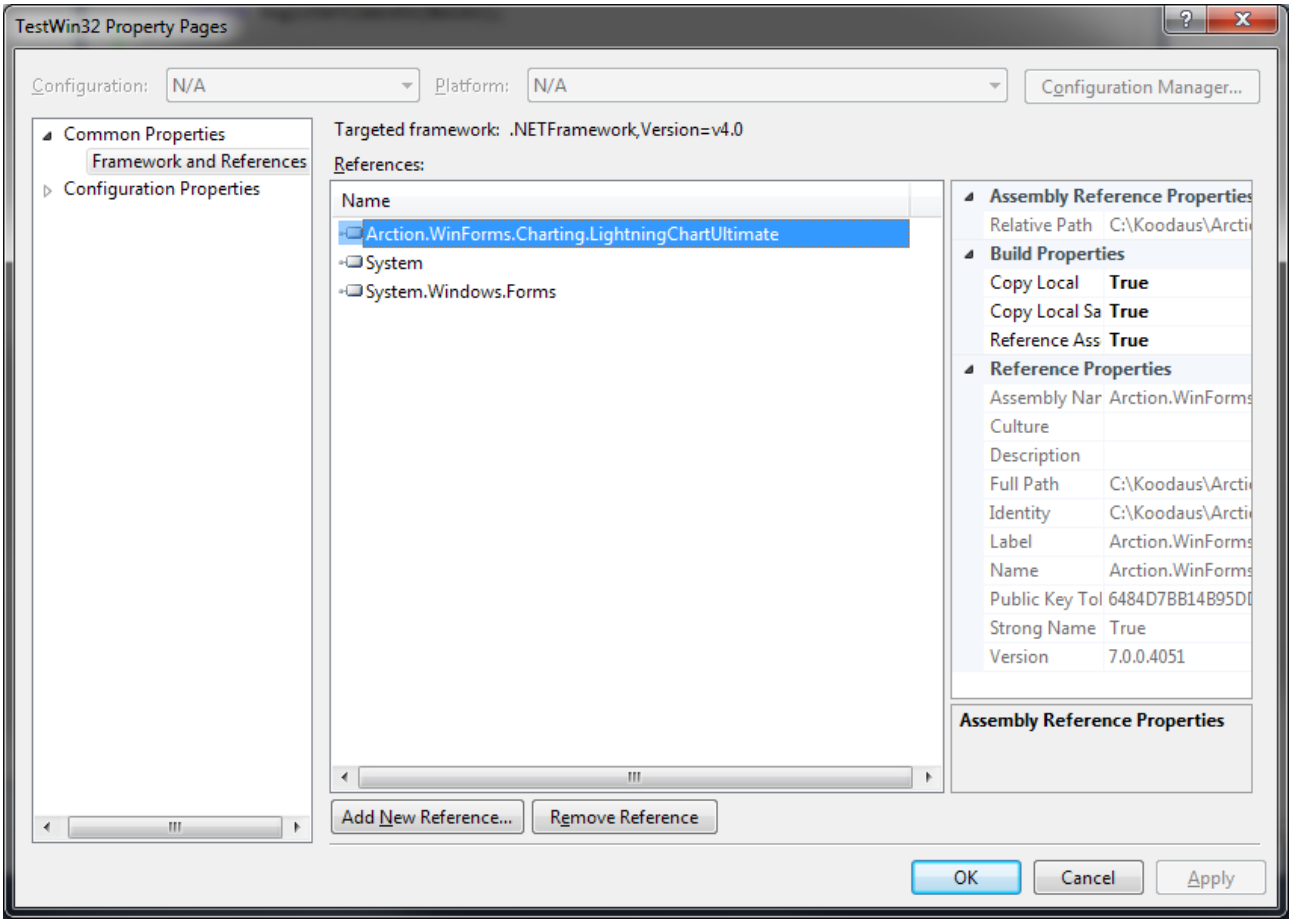


Figure 16-2 Adding .NET references.

In ***Configuration Properties, C/C++*** page, modify ***Resolve #using references*** path to location where the Arction assemblies can be found at development time.

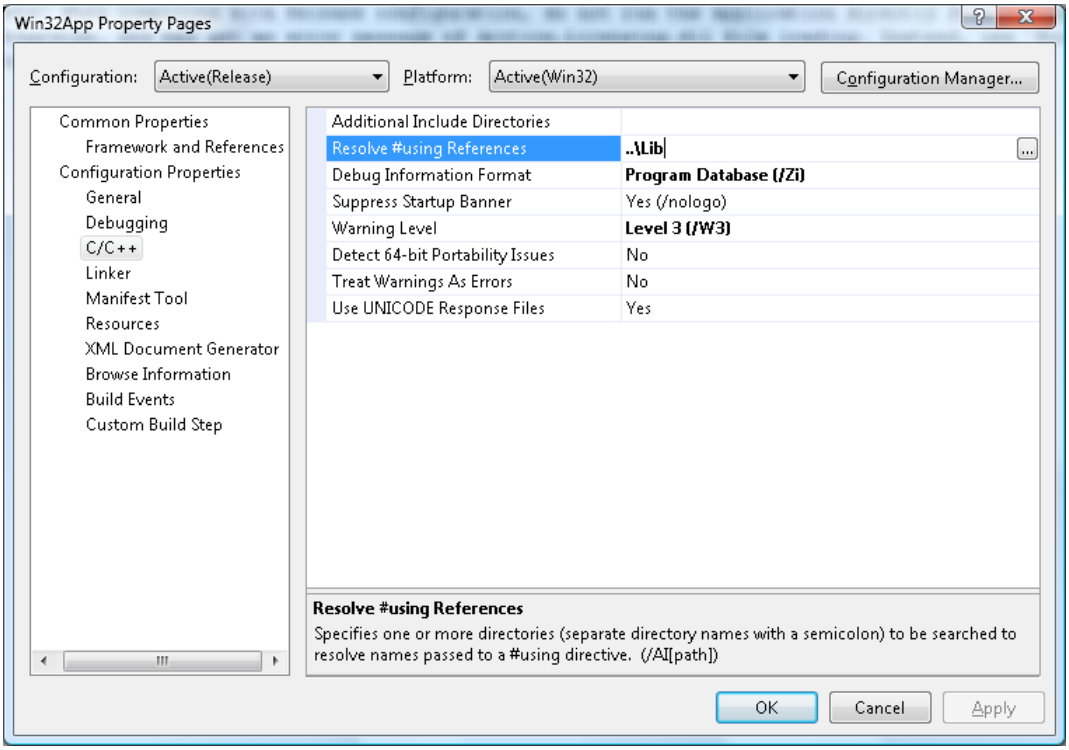


Figure 22-3 Setting "Resolve #using References" path.

Include the library and relevant namespaces after your include rows as follows:

```
#include "stdafx.h"
#include <stdlib.h>
…

//Include Arction DLLs and relevant namespaces
#using "Arction.WinForms.Charting.LightningChartUltimate.dll"
using namespace Arction::WinForms::Charting;
using namespace Arction::WinForms::Charting::Axes;
using namespace Arction::WinForms::Charting::SeriesXY;
```

Declare a class to store global .NET managed variables:

```
//"Global managed variables"
ref class GlobalObjects
{
public:
   static Arction::WinForms::Charting::LightningChartUltimate^ chart;
};
```

Create an object of this class and LightningChartUltimate object inside it, for example in *InitInstance* function:

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   HWND hWnd;

   hInst = hInstance; // Store instance handle in our global variable

   hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
```

```
        CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

    if (!hWnd)
    {
       return FALSE;
    }

    CreateChart(hWnd); //Chart create function


    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    return TRUE;
}
```

Chart creation method can be like the following:

```
//Create a LightningChartUltimate object and place it inside the parent window
void CreateChart(HWND hwndParent)
{

    GlobalObjects^ go = gcnew GlobalObjects();

    go->chart = gcnew LightningChartUltimate("/* Create key in License manager
    program");

    LightningChartUltimate^ chart = go->chart;

    //Disable repaints for every property change
    chart->BeginUpdate();

    //Set parent window by window handle
    chart->SetParentWindow((System::IntPtr) hwndParent);

    //Remove existing series
    chart->ViewXY->PointLineSeries->Clear();

    //Create new series
    PointLineSeries^ series = gcnew PointLineSeries(chart->ViewXY,
           chart->ViewXY->XAxes[0],
           chart->ViewXY->YAxes[0]);

    const int PointCount = 10;

    //Create SeriesPoint array
    array<SeriesPoint> ^ data =
           gcnew array<SeriesPoint>(PointCount);

    //Fill the array
    double yValues[PointCount] = {0.0, 5.0, 4.0, 3.0, 8.0, 10.0, 9.0, 8.0, 3.0, 2.0};
    for(int i=0; i<PointCount; i++)
    {
           data[i].X = i+1;
           data[i].Y = yValues[i];
    }

    //Add the points to series
    series->AddPoints(data,false);

    //Add the series itself into chart's PointLineSeries collection
    chart->ViewXY->PointLineSeries->Add(series);

    //Fit the axis ranges to data assigned
```

```
      chart->ViewXY->FitView();

      //Allow repaints, update chart
      chart->EndUpdate();

    }
```

Add handling for WM_SIZE message (window is resized):

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;

    switch (message)
    {

          case WM_SIZE:
          {
               GlobalObjects^ go = gcnew GlobalObjects();
               if(go->chart != nullptr)
                    go->chart->SetBounds(0,0,0xFFFF & lParam, 0xFFFF &(lParam >> 16));

               break;
          }
    }
    return 0;
}
```

The resulting application of this example looks like the following, when compiled and executed:

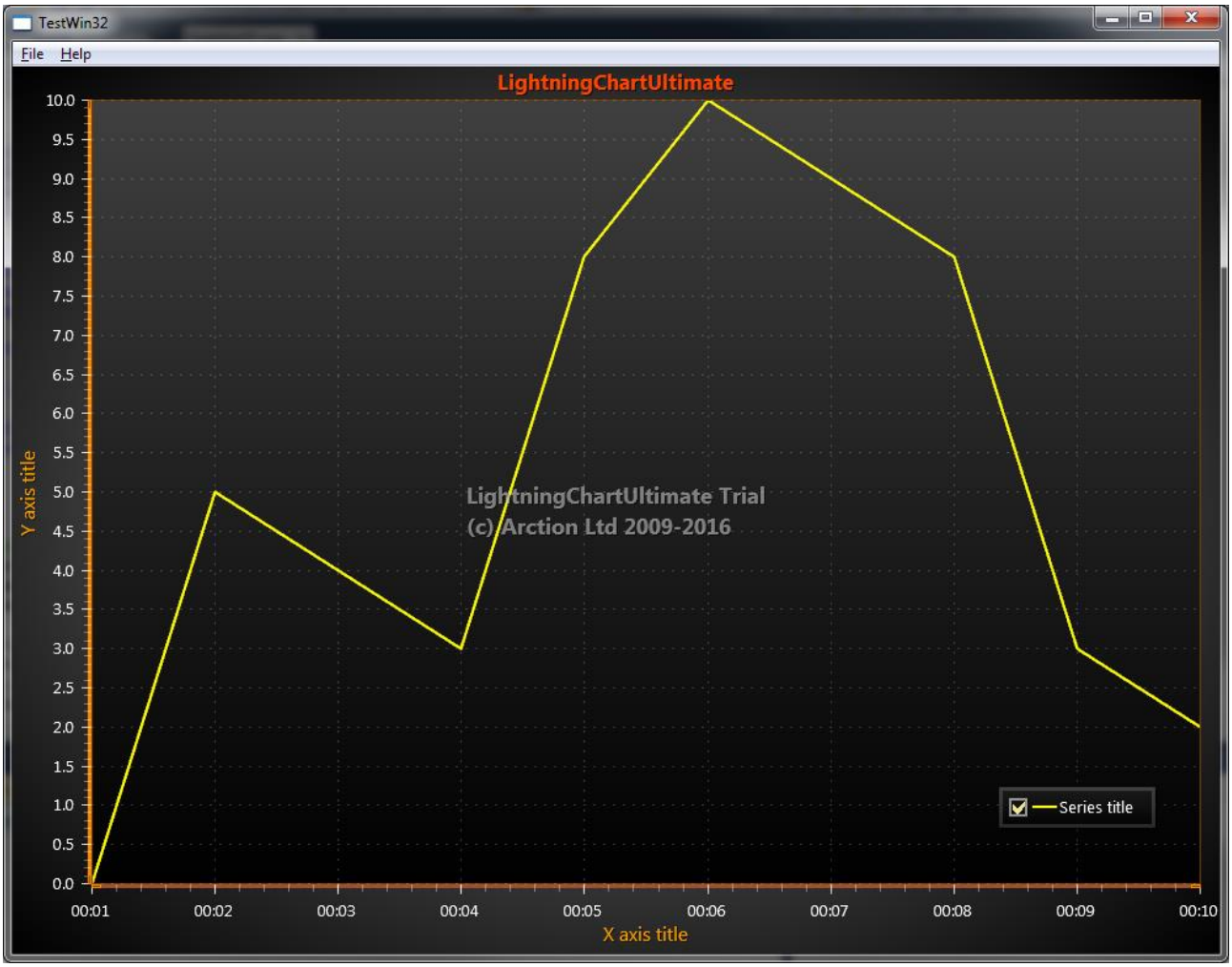


Figure 16-4 Example application executed.

23. Dispose pattern

23.1 Chart created in code

23.1.1 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...

23.1.2 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 LightingChart's objects are not needed anymore it's good practice to Dispose them to prevent memory leaking.

24. Object model notes

24.1 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.

Objects also have reference to their unmanaged counterparts, like GPU resources. Disposing of shared objects explicitly by user code, or internally by LightningChart, will very likely lead into crashing, black screen, or flickering problems.

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.

25. Deployment

25.1 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

25.2 License key

Remember to assign the ***LicenseKey*** property for all LightningChart instances. Otherwise the chart enters in trial mode and works only 30 days, with trial nag on it. Also assign the ***LicenseKey*** property for other Arction controls as well. For license keys management, see section 4.

26. Troubleshooting

26.1 Web support

See www.arction.com/support for frequently asked questions.

Discussion forums are available at http://www.arction.com/forum

26.2 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 5.6.

26.3 Credits

26.3.1 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.

26.3.2 Open-source projects

We present thanks to following open-source projects and material providers:

DirectX library for .NET
LightningChart uses SharpDX-derived DirectX .NET DLLs with Arction-made extensions, http://www.sharpdx.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.

Cryptography
Some cryptographic routines are derivatives of algorithms by RSA and Bruce Schneier.

Polynomial regression
Polynomial regression calculation code is partially based on Math.Net library, http://www.mathdotnet.com/