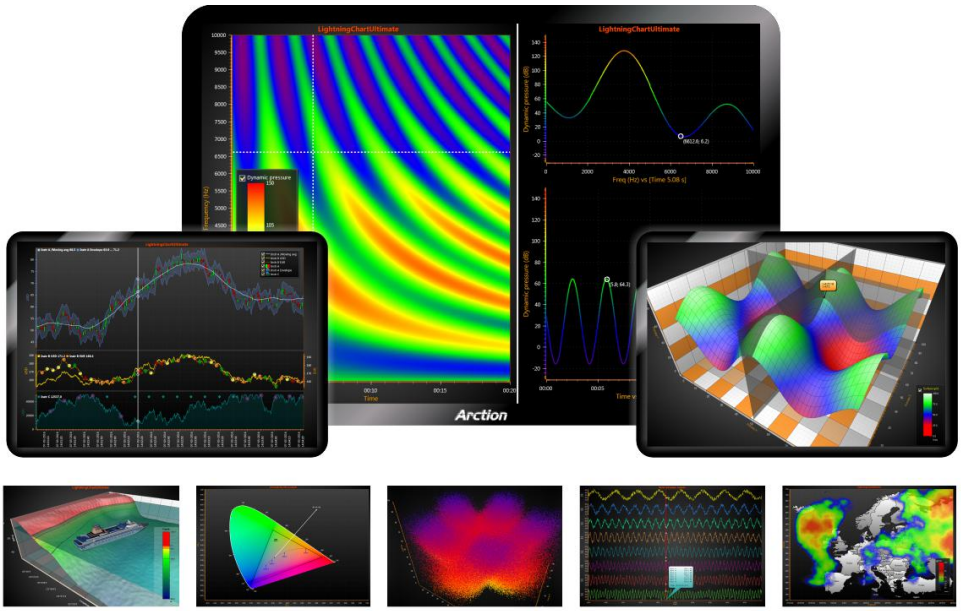


LightningChart®

사용 설명서

본 문서에 관하여

본 문서는 **Arction LightningChart®**에 관한 간략한 사용 설명서입니다. 필수 핵심 기능만에 대한 설명만 포함 되어 있습니다. 수백개의 클라스, 속성 또는 메소드가 본 문서에 기재되어 있지 않습니다. LightningChart 의 기능에 대한 빠른 미리보기를 위해서는 포함된 데모 어플리케이션을 실행하세요. 포함된 데모 어플리케이션의 소스 코드는 LightningChart 구성 요소를 코드로 사용하는 것의 이해를 돕습니다.

본 문서에 기재된 모든 코드 예시는 C# 언어로 쓰여져 있습니다. 대부분의 데모 어플리케이션도 코드 미리보기를 C# 및 Visual Basic .NET 으로 제공됩니다.

질문이 있으시다면 언제든 지원팀께 연락을 주세요 (support@arction.com)!

LightningChart .NET 에 적용됨, v.8.5.1

LightningChart 은 (주) Arction 의 등록 상표입니다

www.arction.com

www.lightningchart.com

目录

1. 개요 9
1.1 차트 판 9
1.2 구성 요소 10
1.3 네임 스페이스 11
2. 설치 13
2.1 설치 전 13
2.2 설정 마법사 실행 13
2.3 비주얼 스튜디오 툴박스에 수동으로 아크션 구성 요소를 추가하기 13
2.4 비주얼 스튜디오 2010-2017 도움말 수동으로 구성하기 14
2.5 코드 매개 변수 및 비주얼 스튜디오 인텔리센스 팁 16
2.6 타겟 프레임워크 선택하기 17
3. 뎁센터 19
3.1 데모 어플리케이션 열기 20
3.2 시드 프로젝트 생성하기 20
3.3 라이센스 추가 22
3.4 라이센스 제거 24
3.5 배포 키 추출 25
3.6 다른 응용 프로그램에 배포 키 적용 26
3.7 개발 컴퓨터에 배포 키 없이 실행 27
3.8 디버거와 실행하기 28
3.9 트라이얼 기간 28
3.10 유동 라이센스 28
4. 라이트닝차트 얼티밋 구성 요소 30
4.1 라이트닝차트® .NET 라이브러리 사용 30
4.2 코드로 차트 생성 31
4.3 툴박스에서 윈도우 폼 프로젝트에 추가 32
4.4 툴박스에서 WPF 프로젝트에 추가 33
4.5 블렌드 WPF 프로젝트에 추가 34
4.6 객체 모델 36
4.7 라이트닝차트 뷰 37
4.8 뷰 및 줌 영역 정의 38
4.9 배경 채우기 설정 39
4.10 외관 구성 / 성능 설정 43
4.11 DPI 핸들링 46
4.12 안티 앨리어싱 47
5. ViewXY 50
5.1 축 레이아웃 옵션 53
5.2 Y 축 63

5.3 X 축 ... 74
5.4 마진 ... 82
5.5 ViewXY 시리즈, 일반 ... 84
5.6 PointLineSeries ... 84
5.7 SampleDataSeries ... 87
5.8 FreeformPointLineSeries ... 89
5.9 라인 시리즈 고급 라인 색 칠하기 ... 91
5.10 하이 로우 시리즈 ... 94
5.11 AreaSeries ... 98
5.12 BarSeries ... 100
5.13 StockSeries ... 103
5.14 PolygonSeries ... 106
5.15 LineCollections ... 108
5.16 IntensityGridSeries ... 110
5.17 IntensityMeshSeries ... 118
5.18 밴드 ... 121
5.19 일정 라인 ... 123
5.20 주석 ... 123
5.21 레전드 상자 ... 130
5.22 줌 및 패닝 ... 137
5.23 NaN 또는 기타 값으로 DataBreaking ... 145
5.24 ClipAreas ... 148
5.25 지도 ... 149
5.26 벡터 지도 ... 150
5.27 타일 지도 ... 167
5.28 StencilAreas ... 169
5.29 LineSeriesCursors ... 173
5.30 EventMarkers ... 177
5.31 지속 시리즈 렌더링 레이어 ... 180
5.32 강도 레이어 지속 시리즈 렌더링 ... 184
6. View3D ... 187
6.1 3D 모델 및 차원 ... 188
6.2 벽 ... 189
6.3 FrameBox ... 190
6.4 카메라 ... 190
6.5 빛 ... 194
6.6 축 ... 196
6.7 마진 ... 199
6.8 3D 시리즈, 기본 ... 199
6.9 PointLineSeries3D ... 200
6.10 SurfaceGridSeries3D ... 208
6.11 SurfaceMeshSeries3D ... 220
6.12 WaterfallSeries3D ... 223

6.13 BarSeries3D....224
6.14 MeshModels....231
6.15 VolumeModels....240
6.16 Rectangle3D objects....252
6.17 Polygon3D objects....253
6.18 줌, 패닝, 회전....256
6.19 레전드 상자....261
6.20 축 범위 내 객체 클리핑....264
6.21 Annotation3D....265
7. 좌표 시스템 컨버터....266
7.1 SphericalCartesian3D....266
7.2 CylindricalCartesian3D....268
8. ViewPie3D....270
8.1 속성....272
8.2 파이 조각....272
8.3 코드로 데이터 설정....273
8.4 2D 로 파이 차크 보기....275
9. ViewPolar....276
9.1 축....277
9.2 마진....280
9.3 레전드 상자....282
9.4 PointLineSeries....284
9.5 AreaSeries....286
9.6 섹터....287
9.7 주석....288
9.8 마커....288
9.9 줌 및 패닝....289
9.10 ViewPolar 에서 데이터 클리핑....291
10. ViewSmith....293
10.1 축....293
10.2 마진....297
10.3 레전드 상자....298
10.4 PointLineSeries....298
10.5 데이터 설정....299
10.6 주석....299
10.7 마커....300
10.8 줌 및 패닝....301
11. 색 테마 설정....301
12. 스크롤바....302
12.1 스크롤바 속성....302
12.2 소수점 또는 음수 값의 스크롤바....303
13. 수출 및 프린트....305

13.1.1 비트맵 이미지 수출 305
13.1.2 벡터 이미지 수출 305
13.1.3 클립보드에 복사 305
13.1.4 바이트 배열로 캡쳐 306
13.1.5 지속적 프레임 쓰기 위한 산출 스트림 설정 306
13.1.6 프린트 307
14. 라이트닝차트 성능 308
14.1 제대로 된 API 판 선택하기 308
5.10 장을 보라 308
15. LightningChart 알림, 에러 및 예외 처리 312
16. ChartManager component 313
16.1 차트 상호 운용, 드래그 드롭 313
16.2 메모리 관리 향상 314
17. SignalGenerator 구성 요소 315
17.1 샘플링 회수, 출력 간격 및 팩터 315
17.2 사인 웨이브형 316
17.3 네모 웨이브형 317
17.4 세모 웨이브형 317
17.5 노이즈 웨이브형 318
17.6 주파수 스위프 319
17.7 앰프 스위프 319
17.8 시작 및 멈추기 320
17.9 주인-노예 구성 멀티 채널 제너레이터 320
17.10 출력 데이터 스트림 320
18. SignalReader 구성 요소 321
18.1 주요 속성 322
18.2 재생을 위해 빠르게 파일 열기 322
19. AudioInput 구성 요소 324
19.1 속성 324
19.2 메소드 324
19.3 이벤트 325
19.4 사용 (윈폼) 325
19.5 사용 (WPF) 327
20. AudioOutput 구성 요소 329
20.1 속성 329
21. SpectrumCalculator 구성 요소 330
22. 헤드리스 모드 333
22.1.1 헤드리스 렌더링 333
22.1.2 제한 및 요구 사항 335

22.1.3 해결 예시 337
23. WPF 어플리케이션에서 윈도우 폼 차트 사용 339
24. C++ 어플리케이션에서 라이트닝차트 사용 342
24.1 필요 C++/CLR 패키지 설치 342
24.2 비주얼 스튜디오 프로젝트 설정 343
24.3 C++ 프로젝트에서 라이트닝차트 어플리케이션 생성 345
25. 제거 패턴 348
25.1 차트 제거 348
25.2 객체 제거 348
26. 객체 모델 노트 349
26.1 다른 객체와 객체 공유 349
27. 라이트닝차트 어셈블리의 배포 351
27.1 참조된 어셈블리 351
27.2 라이센스 키 352
27.3 어플리케이션 코드 난독화 352
27.4 라이트닝차트 코드 난독화 352
27.5 아크션 어셈블리의 XML 파일 352
28. 문제 해결 353
28.1 구 버전에서 업데이트 353
28.2 웹 지원 355
28.3 가상 기기 플랫폼에서 실행 355
29. 크레딧 355
29.1 인텔 수학 커널 라이브러리 355
29.2 오픈 소스 프로젝트 356

1. 개요

LightningChart® .NET SDK 는 마이크로소프트 비주얼 스튜디오의 애드온이며 WPF (윈도우 프리젠테이션 파운데이션) 및 윈도우 폼 .NET 플랫폼을 위한 데이터 시각화 관련 소프트웨어 구성 요소 및 툴 클라스로 만들어 졌습니다.

아크션 구성 요소는 과학적, 공학적, 측정 및 트레이딩 솔루션과 특별 포커스 내 실행 성능 및 고급 기능을 위하여 포함 되었습니다.

라이트닝차트 구성 요소는 GDI/GDI+ 또는 WPF 그래픽 API 들 보다 빠른 저레벨 DirectX11 및 DirectX9 GPU 가속을 사용합니다. 라이트닝차트는 GPU 에 접근 불가능한 몇몇 가상 기기 플랫폼 등에서는 DirectX11/DirectX10 WARP 소프트웨어로 대체합니다.

1.1 차트 판

WPF 를 위해서는 다른 성능 및 MVVM (모델 – 뷰 – 뷰 – 모델) 바인딩 요구 사이의 밸런스를 위해 라이트닝차트 구성 요소가 여러 바인딩 레벨 판으로 사용 가능합니다.

차트 판	프로퍼티 바인딩	시리즈 데이터 바인딩	퍼 데이터 포인트 바인딩	성능
WPF (바인딩 불가)	아니오	아니오	아니오	우수함
WPF (반 바인딩)	예	예	아니오	매우 좋음
WPF (바인딩 가능)	예	예	예	좋음
WinForms	아니오	아니오	아니오	최고

Table 1-1. 바인딩 및 성능 매트릭스.

아크션은 반 바인딩 API 를 시작점으로 추천 합니다.

- WPF 및 멀티 스레딩 이익 내 최고 성능을 위해서는 바인딩 되지 않는 차트를 선택하세요
- WPF 바인딩과 성능의 협정을 위해서는 반 바인딩 차트를 선택 하세요
- 풀 WPF MVVM 디자인 패턴 지원을 위해서는 전체 바인딩 차트를 선택 하세요

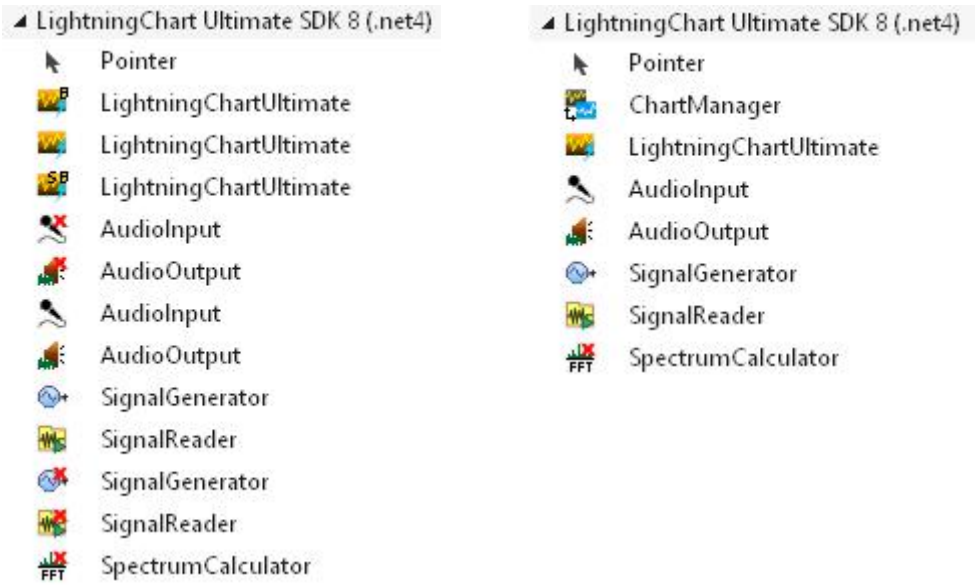
반 바인딩 차트 API 는 라이트닝차트 v.6 의 WPF 와 아주 비슷하지만 컬렉션으로 생성된 객체까지 포함하는 확장된 속성이 있습니다.

같은 어플리케이션 내에서 다른 차트 판을 사용할 수 있습니다. 전체 바인딩 되는 기본 차트를 만들어 데이터 바인딩을 하는 도중에 논 바인딩 차트를 성능위주 업무를 위해 사용하는 것도 가능합니다.

(뷰 XY 축, 3D Lights 등) 반 바인딩 및 바인딩 차트의 컬렉션 속성은 기본적으로 XAML 에디터를 완전히 지원하게 비어 있습니다. 비 바인딩 및 윈폼 컬렉션들은 기본 아이템으로 채워져 있습니다.

주의! 비 바인딩 WPF 차트는 전혀 XAML 에 구성되지 않게 되었습니다. 코드-비하인드로 사용하세요.

1.2 구성 요소



보기 1-1. 좌측에는 WPF 툴박스 구성 요소. 우측에는 윈폼 툴박스 구성 요소

차팅 집합

라이트닝 차트 얼티밋 차트 구성 요소. 데이터를 여러 방식으로 시각화함

아이콘 윗 구석에, **SB = 반 바인딩 WPF 차트** d 와 **B = 바인딩 WPF 차트**

차트 관리자 여러개의 차트 구성 요소의 상호 운용을 제어하고 실시간 계산 메모리 관리. 제 17 장을 보세요.

시그널 툴즈 어셈블리

UI 가 없는 구성 요소는 ✘ 자로 표시 되었습니다.

 **오디오 입력** 음향 기계에서 나오는 오디오 스트림의 웨이브형을 읽습니다. 라인-인 또는 마이크-인 커넥터는 음향 기기에 흔히 찾을 수 있습니다. 이 실시간 스트림은 다른 제어에 전달 가능합니다. 제 20 장을 보세요.

 **오디오 출력** 음향 기기의 스피커 또는 라인-출력을 통해 실시간 데이터 스르림을 재생합니다. 오디오 스트림이 아니어도 되며 아무 샘플된 실시간 시그널을 사용 가능합니다. 제 21 장을 보세요.

 **시그널 발생기** 여러 구성 가능한 웨이브형 요소에서 시그널을 발생 시킵니다. 제 18 장을 보세요.

 **시그널 리더** 시그널 파일의 PCM 포맷의 WAV 등 웨이브형 데이터를 읽습니다. 제 19 장을 보세요.

 **스펙트럼 계산기** FFT (고속 포리어 변형)을 사용하여 시그널 데이터 (시간 영역)를 스펙트럼 (주파수 영역)으로 변환 시킵니다. 반대로 주파수 영역에서 시간 영역으로 변환 가능한 메소드도 포함됩니다. 제 22 장을 보세요.

1.3 네임 스페이스

차트 판	어셈블리 명	네임 스페이스 루트	XML 네임스페이스
WPF (비 바인딩)	Arction.**Wpf.Charting**.LightningChartUltimate.dll	Arction.Wpf.Charting	xmlns:lcunb="http://schemas.arction.com/**charting**/ultimate/"
WPF (반 바인딩)	Arction.**Wpf.SemibindableCharting**.LightningChartUltimate.dll	Arction.Wpf.SemibindableCharting	xmlns:lcusb="http://schemas.arction.com/**semibindablecharting**/ultimate/"
WPF (바인딩)	Arction.**Wpf.BindableCharting**.LightningChartUltimate.dll	Arction.Wpf.BindableCharting	xmlns:lcufb="http://schemas.arction.com/**bindablecharting**/ultimate/"
윈폼	Arction.**WinForms.Charting**.LightningChartUltimate.dll	Arction.WinForms.Charting	N/A

표 1-2. 모든 라이트닝차트 .NET 판의 어셈블리 명 및 네임 스페이스.

2. 설치

2.1 설치 전

컴퓨터가 요구 사양에 충족하는지 확인하세요.

- DirectX 9.0c (쉐더 레벨 3) 레벨 그래픽 어댑터 이상, 또는 DirectX11 과 그래픽 하드웨어 없이 렌더링 호환 가능한 운영 체제. DirectX11 과 호환 가능한 그래픽 하드웨어 사용하는 것을 추천 드립니다.
- 윈도우 버전 Vista, 7, 8 또는 10, 32 bit 또는 64 bit, 윈도우 서버 2008 R2 이상
- 개발 위한 비주얼 스튜디오 2010-2017, 배포 버전 필요 없음.
- .NET 프레임워크 v. 4.0 이상 설치.

2.2 설정 마법사 실행

LightningChart .NET SDK v8.exe 를 우클릭을 하세요. 이것은 비주얼 스튜디오 툴박스에 구성 요소를 설치 합니다. 또 툴박스 제어 관련 파일들도 설치 합니다. 구성 요소 또는 설치가 실패할 시 다음 부분에 설명된 대로 수동으로 설치하세요.

라이트닝 차트 트라이얼판에는 **SetupDownloader.exe** 가 사용 됐을 확률이 큽니다. 이것은 SDK 를 다운로드 및 설치를 하여 LightningChart .NET SDK v8.exe 를 실행할 필요를 없앱니다.

2.3 비주얼 스튜디오 툴박스에 수동으로 아크션 구성 요소를 추가하기

윈폼

1. 비주얼 스튜디오를 실행하세요. 새로운 **윈폼** 프로젝트를 생성하세요. 툴박스에 우클릭 후 **탭 추가** 선택해 “아크션”이란 이름을 주세요.
2. 아크션 탭에 우클릭 후 **아이템 선택…**을 누르세요.
3. **툴박스 아이템 선택** 창에서 **.NET 프레임워크 구성 요소** 페이지를 선택하세요. **검색…**을 클릭 하세요.

   Arction.WinForms.Charting.LightningChartUltimate.dll,

Arction.WinForms.SignalProcessing.SignalTools.dll 을 구성 요소가 설치된 폴더에서 검색하세요. 이는 보통 **C:\Program Files (x86)\Arction\LightningChart .NET SDK v.8\LibNet4** 에 설치됩니다. 실행 후 구성 요소를 툴박스에서 찾을 수 있을겁니다.

WPF

1. 비주얼 스튜디오를 실행하세요. 새로운 **WPF** 프로젝트를 생성하세요. 툴박스에 우클릭 후 **탭 추가**를 선택하고 "아크션"이란 이름을 주세요.
2. 아크션 탭에 우클릭 후 **아이템 선택…**을 클릭하세요
3. **툴박스 아이템 선택** 창에서 **WPF 구성 요소** 페이지를 선택하세요. **검색…**을 클릭하세요

Arction.Wpf.Charting.LightningChartUltimate.dll,
Arction.Wpf.SemibindableCharting.LightningChartUltimate.dll,
Arction.Wpf.BindableCharting.LightningChartUltimate.dll 및 **Arction.Wpf.SignalProcessing.SignalTools.dll**, 구성 요소가 설치된 폴더에서 검색하세요. 이는 대부분 **C:\Program Files (x86)\Arction\LightningChart .NET SDK v.8\LibNet4** 에 설치 되었습니다. 실행하세요. 구성 요소가 툴박스에 생겼을 겁니다.

2.4 비주얼 스튜디오 2010-2017 도움말 수동으로 구성하기

이 장에는 LightningChart® .NET 도움말 내용 수동 설치 관련 정보를 찾을 수 있습니다. 비주얼 스튜디오 2010-2017 에 로컬 도움말 페이지가 설치 안되었을 시 이 정보를 사용하세요. LightningChart® .NET 설치 시 로컬 도움말 페이지가 설치 안되었다면 LightningChart® .NET 의 도움말도 설치가 안됩니다

다음과 같이 한다면 비주얼 스튜디오 2010-2017 에서 LightningChart® .NET 의 도움말을 볼 수 있습니다. 라이트닝 차트의 클라스에 F1 을 누르거나 또는 마이크로소프트 도움말 뷰어로 도움말 내용을 검색하세요.

2.4.1 비주얼 스튜디오 2010

다음과 같이 비주얼 스튜디오 2010 에 LightningChart® .NET 도움말 페이지를 수동 설치하세요.

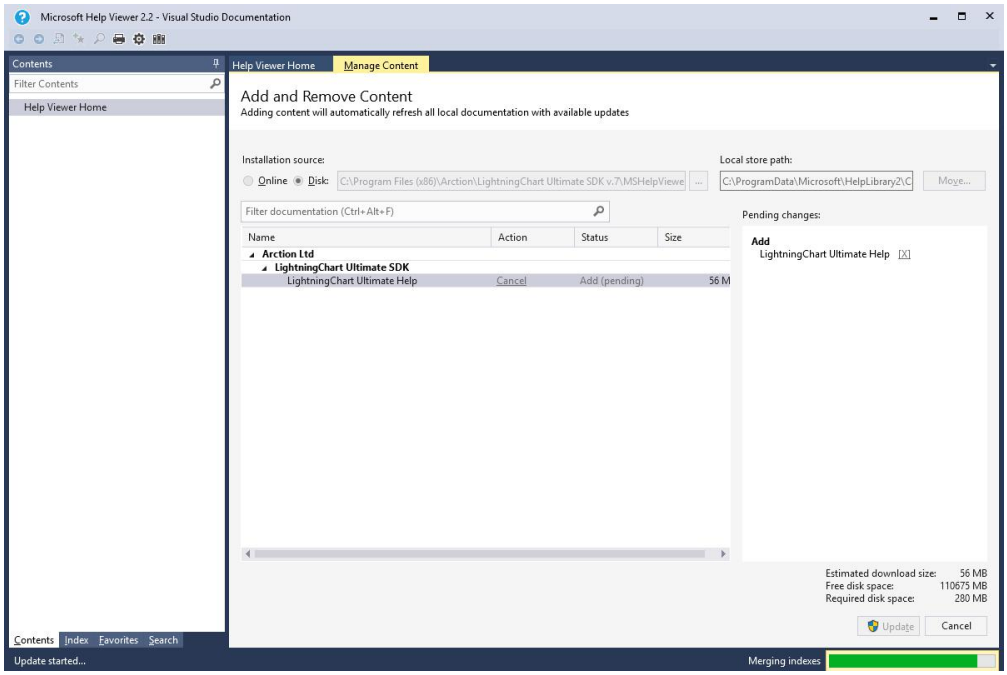
1. 비주얼 스튜디오 2010 을 실행하세요.
2. **도움말 -> 도움말 설정 관리** 선택하세요.

3. 도움말 라이브러리 관리자에서 **설정** 링크를 클릭하세요
4. **로컬 도움말을 사용하고 싶습니다**가 선택 되었는지 확인하세요.
5. ***로컬 도움말을 사용하고 싶습니다***가 선택 되었다면 **취소**를 눌러 도움말 라이브러리 관리자로 돌아 가세요. 선택이 안되었다면 **OK** 를 누르세요.
6. **디스크 내용물 설치** 링크를 누르세요.
7. 검색 버튼을 클릭하여 LightningChart® .NET 설치 폴더로 가세요. 기본 설치 경로는 **C:\Program Files (x86)\Arction\LightningChart .NET SDK v.8 \MSHelpViewer** 입니다.
8. **HelpContentSetup.msha** 를 선택하고 **열기** 버튼을 누르세요.
9. 다음을 클릭 하세요.
10. LightningChart® .NET 도움말 옆에 추가 링크가 있습니다. 클릭하고 **상태** 열 값이 **업데이트 펜딩** 으로 바뀐 것을 확인 하세요.
11. **업데이트** 버튼을 클릭하세요. 만약 도움말 라이브러리 관리자가 계속하고 싶냐고 물으면 **예** 버튼을 클릭 하세요. 도움말 라이브러리 업데이트가 시작 됩니다.
12. 도움말 라이브러리 업데이트가 완료된 후 **끝내기** 버튼을 눌러 도움말 라이브러리 관리자를 닫으세요.

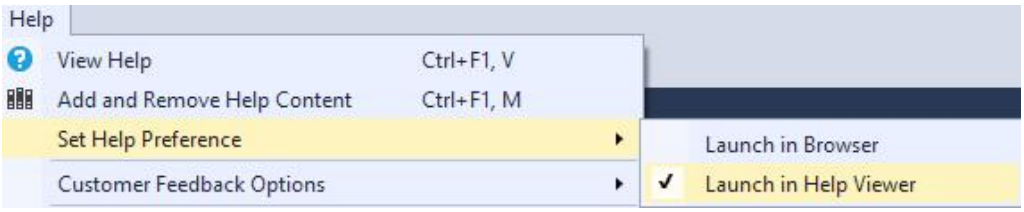
2.4.2 비주얼 스튜디오 2012-2017

다음과 같이 라이트닝차트 .NET 도움말을 비주얼 스튜디오 2012-2017 수동 설치를 하세요:

1. 비주얼 스튜디오 2012, 2013, 2015 또는 2017 를 실행 하세요.
2. **도움말 -> 도움말 추가 및 제거** 선택 하세요.
3. 마이크로소프트 도움말 뷰어가 실행된 후 **컨텐츠 관리** 선택하세요.
4. **설치 소스** 아래 **디스크** 선택하세요.
5. 파일을 검색 하기 위해 점 세개 있는 버튼을 클릭 하세요.
6. 라이트닝차트 .NET 가 설치 되어 있는 폴더로 가세요. 기본적으로 **C:\Program Files (x86)\Arction\LightningChart .NET SDK v.8\MSHelpViewer** 에 설치 되어 있습니다.
7. **HelpContentSetup.msha** 를 선택하고 **열기** 버튼을 누르세요.
8. 라이트닝차트 .NET 도움말 옆에 **추가** 링크가 있습니다. 클릭하고 상태 열 값이 **추가 대기**로 바뀐 것을 확인하세요.



9. **업데이트** 버튼을 누르세요. 만약 도움말 라이브러리 관리자가 계속 하고 싶냐고 물으면 **예** 버튼을 누르세요. 도움말 라이브러리 업데이트가 시작될 겁니다.
10. 도움말 라이브러리가 업데이트된 후 마이크로소프트 도움말 뷰어를 닫아도 됩니다.
11. 비주얼 스튜디오 메뉴 / 도움말에서 **도움말 설정 : 도움말 뷰어로 열기**를 선택하세요.



보기 2-2 도움말 설정하기

보기 2-1. 라이트닝차트 얼티밋 도움말 추가하기

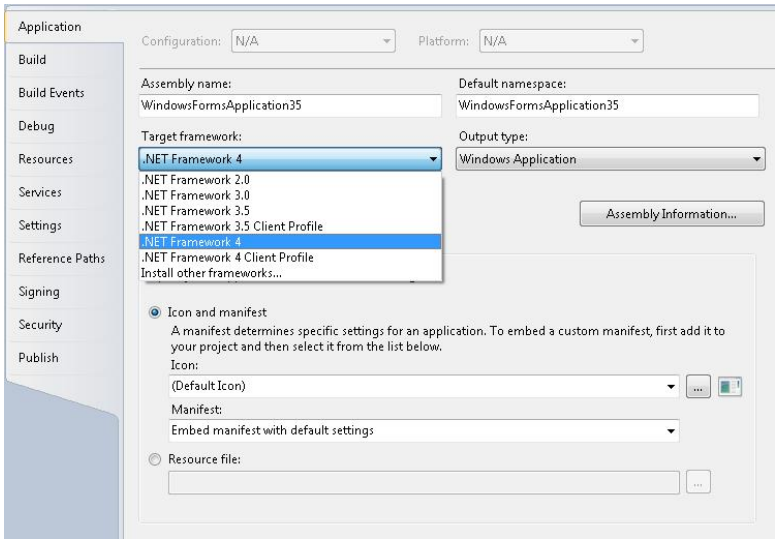
2.5 코드 매개 변수 및 비주얼 스튜디오 인텔리센스 팁

. LightningChartUltimate.dll 파일이 글로벌 어셈블리 캐쉬에서 참조 되었고 제어들이 자동 툴박스 인스톨레서 설치가 안되었을 시 인텔리센스는 라이트닝차트 관련 코드를 타이핑 할 때에 코드 힌트를 보이지 않을

수 있습니다. LightningChartUltimate.dll 파일을 프로젝트 레퍼런스 리스트에서 제거 하세요. 설치 디렉토리(원래 **C:\Program files (x86)\Arction\LightningChart .NET SDK v.8\LibNet4**)에서 검색해 다시 추가하세요.

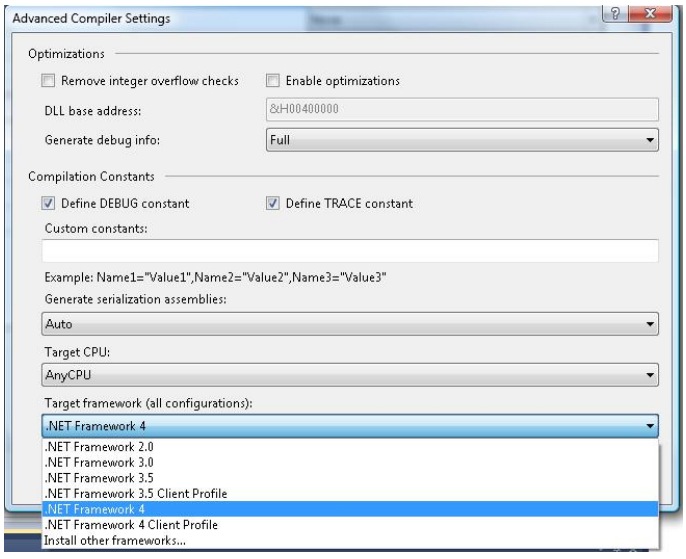
2.6 타겟 프레임워크 선택하기

C# 프로젝트 내 프레임워크 선택을 **프로젝트 -> 속성 -> 어플리케이션 -> 타겟 프레임워크**에서 할 수 있습니다.



보기 2-3. C# 프로젝트 타겟 프레임워크 선택하기

비주얼 베이직 프로젝트 내 프레임워크를 **프로젝트 -> 설정 -> 컴파일 -> 고급 컴파일 옵션 -> 타겟 프레임워크** 에서 선택 가능합니다.



보기 2-4. 비주얼 베이직 프로젝트 타겟 프레임워크 선택하기

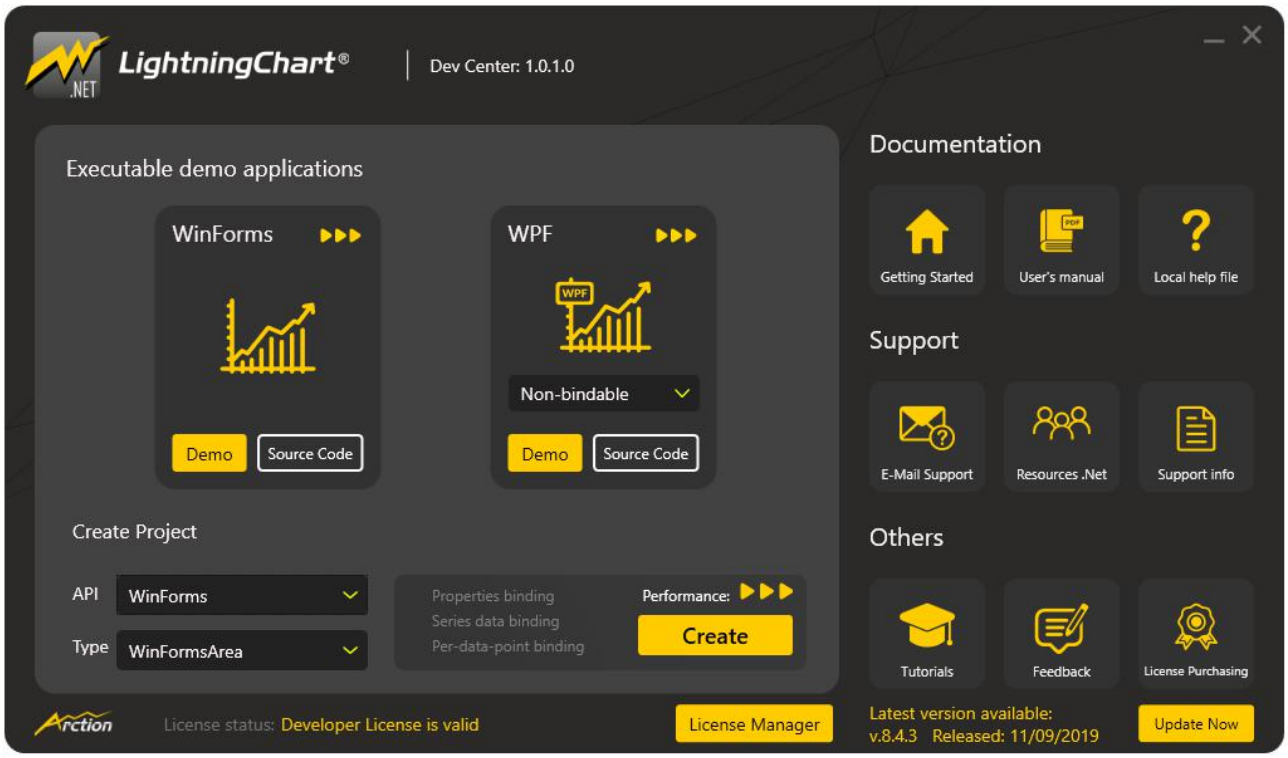
.NET Framework 4, .NET Framework 4 Client Profile 또는 **.NET Framework 4.5, or 4.6** 을 선택 하세요).

올맞은 .NET 프레임워크가 선택 되어 있어야만 라이트닝차트 .NET SDK 제어가 비주얼 스튜디오 툴박스에 나타납니다.

3. 뎁센터

라이트닝차트 .NET 버전 8.5 이상으로 **LightningChart .NET SDK v8.exe** 설치를 실행할 때 뎁센터도 자동 설치가 됩니다. 뎁센터는 라이트닝차트 .NET 기능 및 리소스에 빠른 접근을 허용하는 신 응용 프로그램 입니다. 몇가지의 마우스 클릭만으로 다음과 같은 작업 수행이 가능합니다.

- 데모 어플리케이션 열기
- 시드 프로젝트 생성
- 튜토리얼 및 사용자 메뉴얼 등 문서 리소스 열기
- 이메일로 지원 연락하기
- 지원팀에게 보낼 수 있는 어플리케이션 정보 자동으로 모으기. 이는 지원팀이 문제를 더욱 빠르게 해결 하는 데에 도와 줍니다.
- (주) 아크션에게 피드백을 보낼 수 있는 퀵링크
- 라이센스 상태 확인 및 라이센스 업데이트 또는 활성화 위한 라이센스 관리자 열기
- 신규 라이센스 구매

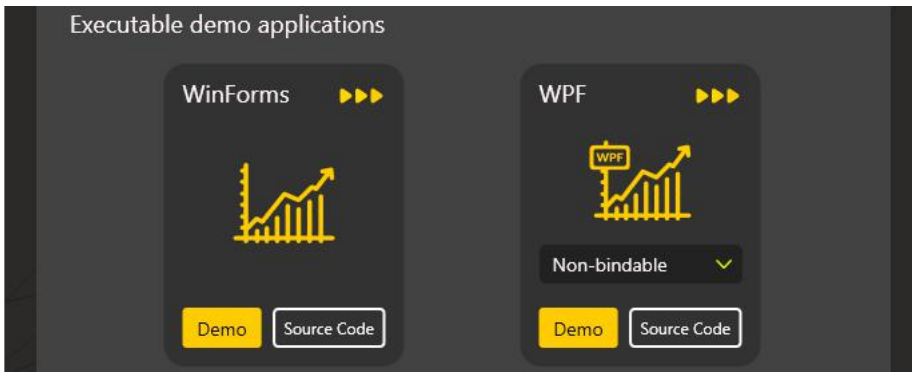


보기 3-1. 모든 기능을 실행할 수 있는 뎁센터 메인 창.

3.1 데모 어플리케이션 열기

라이트닝차트 구성 요소 및 기능 사용을 배울 수 있는 주요 정보의 출처 중 하나는 데모 어플리케이션 입니다. 차트판 당 하나, 총 4 가지의 데모 어플리케이션이 있습니다 (윈폼, WPF 비바인딩, WPF 반바인딩, WPF 바인딩, 차트판에 대한 정보를 위해 제 1.1 장을 보세요). 뎁센터는 데모 -버튼을 누름으로써 모든 데모 어플리케이션 실행을 가능케 합니다. WPF -데모 실행 시 드롭 다운 메뉴에서 WPF 차트판을 선택할 수 있습니다.

라이트닝차트 .NET SDK 를 설치하면 모든 데모 예시의 소스 코드를 자동으로 컴퓨터에 추가합니다. 뎁센터에서 소스 코드 -버튼을 누르면 비주얼 스튜디오 또는 선택 텍스트 이디터로 열 수 있는 코드 파일들이 저장



보기 3-2. 뎁센터로 데모 어플리케이션 실행하기. 우측에 드롭다운 메뉴로 WPF 차트판이 선택 가능합니다.

된 폴더를 엽니다.

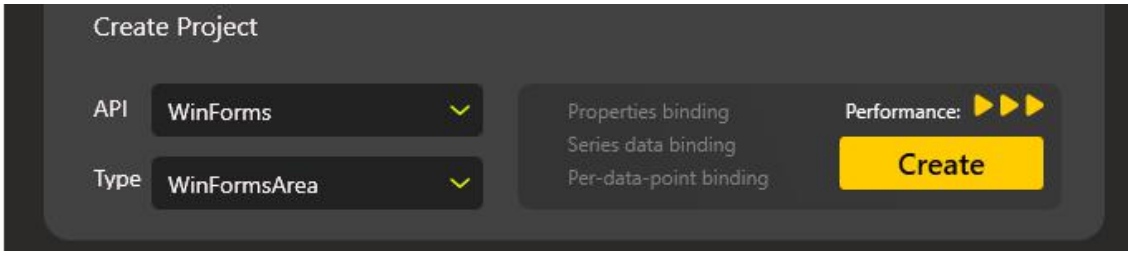
3.2 시드 프로젝트 생성하기

뎁센터로 시드 프로젝트 생성이 가능합니다. 이는 몇가지의 데이터 포인트와 속성 설정이 있는 단일 시리즈가 있는 간단한 .NET 프로젝트 입니다. 이 프로젝트들은 새로운 라이트닝차트 기반 어플리케이션 생성할 때 기반으로 사용 가능합니다.

시드 프로젝트를 생성하기 위해 먼저 API 버전 및 어플레이션 종류를 선택하세요. 종류 선택은 반바인딩 시드 프로젝트 생성할 때에만 필요 합니다. 이 경우에는 MVVM 기반 어플리케이션 생성 가능성이 있기 때문입니다. 생성 -버튼을 누르면 어느 폴더에 프로젝트를 생성할건지 선택할 수 있습니다. 폴더에 이미 비슷한 시드 프로젝트가 존재한다면 신규 프로젝트가 생성되지 않는 점을 주의하길 바랍니다. 다시 말해 구 프로젝트

를 덮어 쓰지 않습니다.

보기 3-3 시드 프로젝트 생성하기



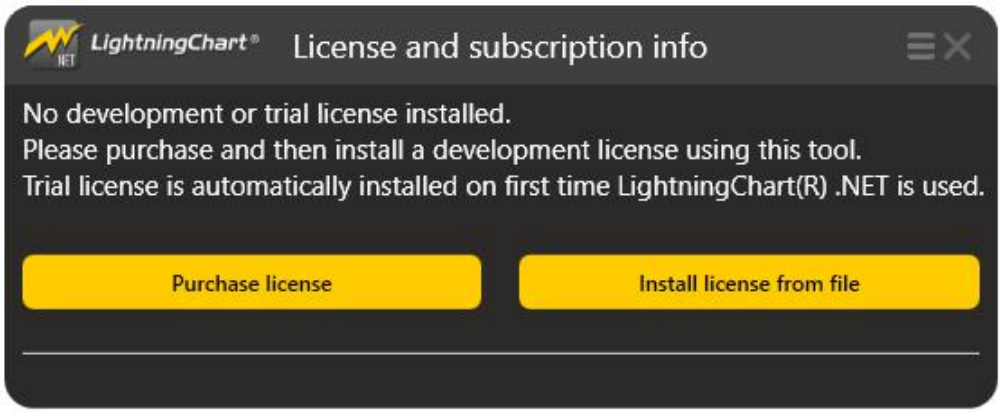
3.3 라이센스 추가

덴센터에서 또는 윈도우 시작 메뉴: 프로그램 / 아크션 / 라이트닝차트 .NET SDK / **라이센스 관리자**에서 라이센스 관리자 응용 프로그램을 실행해 라이센스를 관리하세요.

아크션 구성 요소들은 라이센스 키 보호 시스템을 사용합니다. 구성 요소들은 유효한 라이센스를 소유해야만 사용 가능합니다. 라이센스는 다음과 같은 정보를 갖고 있습니다:

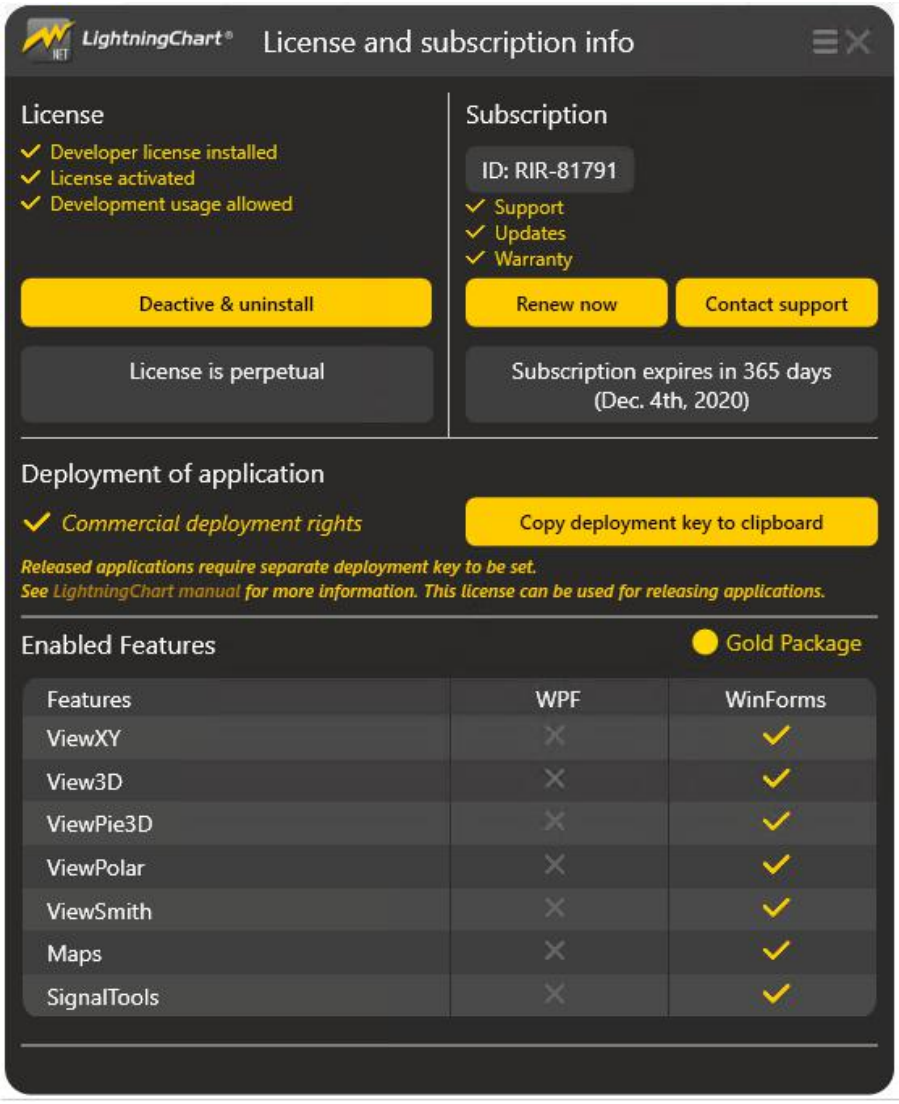
- XY, ViewXY, View3D, ViewPie3D, Maps, ViewPolar, ViewSmith, Volume Rendering, Signal Tools 등 활성화 기능들
- WPF / 윈폼 / 양 기술들
- 몇명의 사용자에게 라이센스 활성화 허용 설정 (기본 1 명).
- 구독 만료일 (업데이트 및 지원 만료 날짜)
- 기술 지원 포함
- 개발자 라이센스 또는 플로팅 라이센스
- 학생 라이센스

툭박스에서 어플리케이션으로 처음으로 아크션 구성 요소를 드래그할 때 라이센스 관리자 창에서 라이센스 키를 요구합니다. 받은 라이센스 파일에서 모든 라이센스 키를 동시 추가 하세요. **파일에서 라이센스 설치하기...** 를 눌러 **.alf** 파일을 검색하세요.



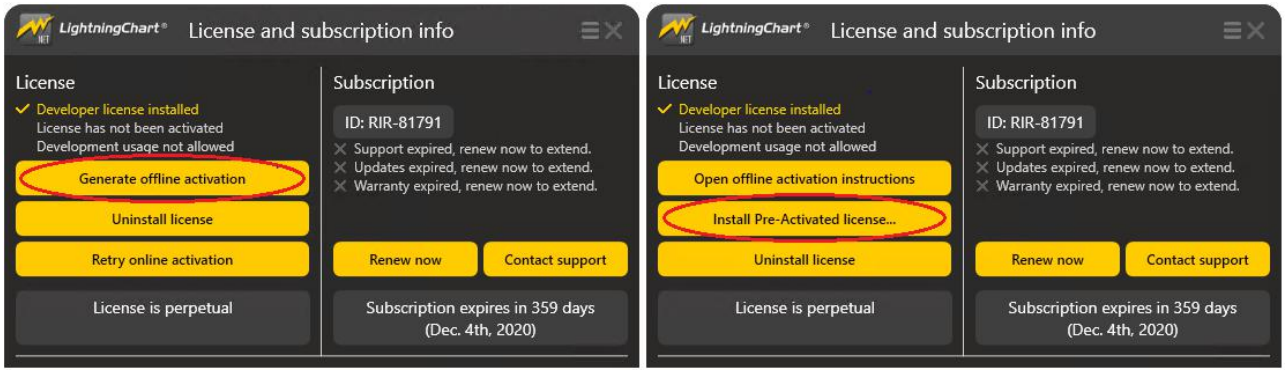
4-1. 아무 라이센스가 설치 안되었을 때 라이센스 관리자. 라이센스 파일에서 라이센스 설치를 통해 라이센스 파일 추가 가능합니다.

라이센스 파일에서 추가 후 개발자 라이센스들은 인터넷으로 자동으로 아크션 라이센스 서버로에 활성화 됩니다.



보기 4-2. 라이센스 파일이 성공적으로 추가된 라이센스 관리자 창.

온라인 활성화가 인터넷 연결 등의 문제로 가능하지 않으면 이메일을 통해 라이센스들을 활성화 할수 있습니다. 온라인 활성화가 두어번 실패 후 **오프라인 활성화 요청** 버튼을 누를 수 있습니다. 오프라인으로 라이센스를 비활성화 하는 것은 온라인 과정과 비슷합니다.



보기 4-3. 온라인 활성화 실패 후 오프라인 활성화 옵션이 라이센스 관리자에 나온다.

오프라인 버튼을 누르면 화면에 설명이 나옵니다. 설명을 따라 아크션 라이센스 팀에 이메일을 보내세요 (licensing@arction.com).

아크션이 오프라인으로 설치하는 설명서를 보내줄 겁니다. 보통 2 일 소요됩니다.

주의! 전화로 활성화/비활성화는 가능하지 않습니다. 이는 키 코드에 수 천개의 문자가 있기 때문입니다.

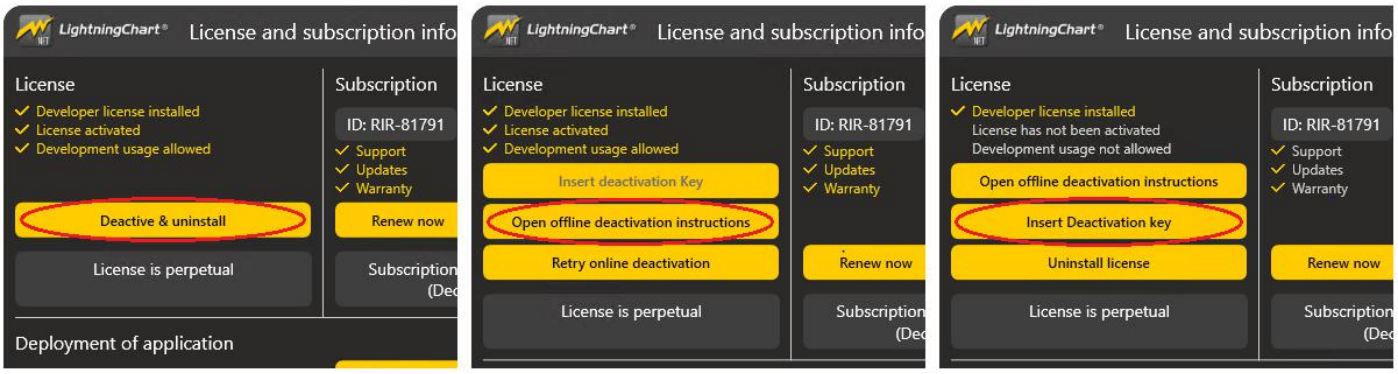
주의! 라이트닝차트 7.1 이상 버전부터는 차트 관리자 구성 요소를 사용하기 위해 라이센스 키가 필요 없습니다.

주의! 라이트닝차트 8.0 이상 버전부터는 LIC 유형의 키가 지원되지 않습니다. ALF 라이센스가 필요합니다. ALF 라이센스를 받지 못했을 경우 아크션에게 연락 바랍니다.

3.4 라이센스 제거

비활성화 & 제거 버튼을 이용해 라이센스를 시스템에서 제거 할 수 있습니다. 자동 비활성화를 위해서는 온라인 연결이 필요합니다. 인터넷 연결이 불가능한 경우 이전 장에 지시된것 같이 이메일을 통해 비활성화를 할 수 있습니다.

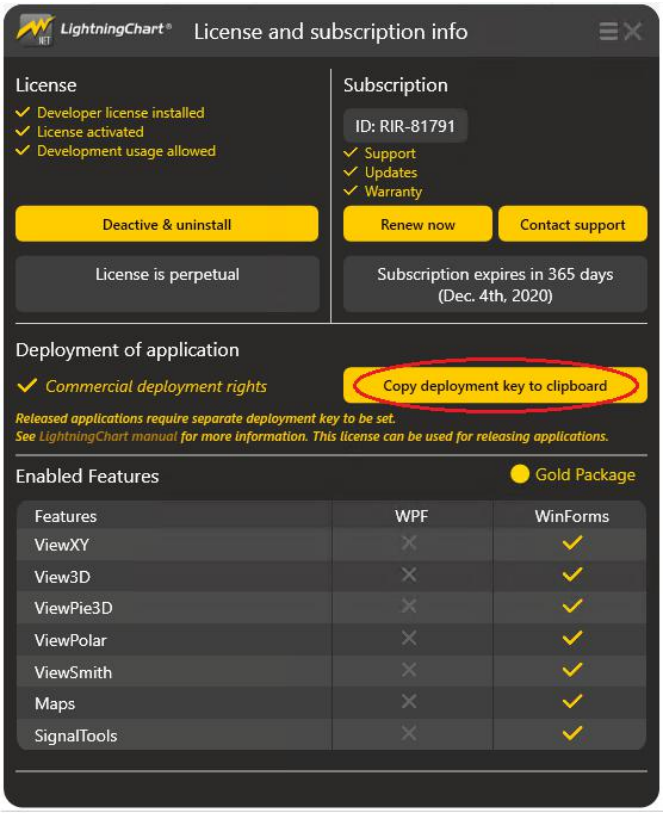
라이센스가 비활성화 된 후 다른 컴퓨터에 설치 가능합니다.



보기 4-4. 라이센스 비활성화 및 제거. 온라인 비활성화 (좌)가 실패 시, 오프라인 옵션 (우)를 사용 할 수 있습니다.

3.5 배포 키 추출

소프트웨어가 배된 컴퓨터에서 라이트닝 차트 응용 프로그램을 실행 시키기 위해서는 코드로 배포키를 적용해야 합니다. **클립보드에 배포 키 복사** 버튼을 이용해 라이센스 키에서 배포키 추출 가능합니다.



보기 4-5. 라이센스 관리자에서 클립보드에 배포키 복사.

3.6 다른 응용 프로그램에 배포 키 적용

코드로 원하는 구성 요소들을 위해서 스태틱 **SetDeploymentKey** 메소드를 이용하세요. 쓰지 않을 구성 요소들을 위해 키 설정할 필요 없습니다 (예. 비바인딩 어플리케이션에서 바인딩 차트 키 설정 등). 구성 요소들이 사용되기 전 어딘가에 **SetDeploymentKey** 메소드를 불러 사용하세요. 차트 사용하는 클라스의 스태틱 컨스트럭터 또는 어플리케이션의 메인 클라스에 넣는 것을 추천 드립니다.

배포 관련 더욱 세밀한 정보를 위해서 제 28 장을 보세요.

윈폼

모든 윈폼 어플리케이션에 기본으로 생성되는 프로그램 클라스에 스태틱 컨스트럭터 메소드를 적용하는 예시를 위해 다음을 보세요.

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

namespace WindowsFormsApplication1
{
  static class Program
  {

      static Program()
      {
         //Set Deployment Key for Arction components
         string deploymentKey = "VMalgCAAO6kO1RgiNIBJABVcG.R..Kikfd...";

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

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

WPF

App.xaml.cs 앞에 앱클래스 스태틱 컨스트럭터에 키를 적용 시키는 예를 위해 다음을 보세요.

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
```

```
using System.Linq;
using System.Windows;
using Arction.Wpf.SignalProcessing;

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

            // Setting Deployment Key for fully bindable chart
            Arction.Wpf.BindableCharting.LightningChartUltimate
            .SetDeploymentKey(deploymentKey);

            // Setting Deployment Key for semi-bindable chart
            Arction.Wpf.SemibindableCharting.LightningChartUltimate
            .SetDeploymentKey(deploymentKey);

            // Setting Deployment Key for non-bindable chart
            Arction.Wpf.Charting.LightningChartUltimate
            .SetDeploymentKey(deploymentKey);

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

        }

    }
}
```

주의! 어플리케이션에 배포 키를 설정 하지 않고서는 라이트닝차트 어플리케이션은 해당 기계에 30 일 데모 모드로 설정 됩니다 (개발 라이센스 키가 설치 되지 않은 컴퓨터에 해당).

3.7 개발 컴퓨터에 배포 키 없이 실행

SetDeploymentKey 로 배포키가 적용되고 개발 라이센스가 설치된 컴퓨터에서 어플리케이션을 실행할 때 라이브러리는 **개발 라이센스 키를 우선**으로 여깁니다. 배포 키에 로컬 설치 라이센스 (예, 실버 팩) 보다 더

많은 특성이 포함 되어 있을 때 (예, 골드 팩) 사용자 또는 디버깅에 헷갈림이 생길 수 있습니다. 해당 제한에 대해서 개발자는 알고 있어야 합니다.

아크션은 같은 팀 내 같은 종류의 라이센스를 사용 하는 것을 추천 드립니다.

3.8 디버거와 실행하기

배포 키가 제대로 설정 되고 디버거가 첨부된 비주얼 스튜디오에서 프로젝트를 실행할 때 개발 라이센스 키가 시스템에서 찾아지지 않으면 차트는 슬로우 렌더링 모드가 됩니다. 최대 FPS 는 ~1 정도이며 차트 위에 메시지를 표시합니다.

개발 라이센스 키 없이 라이트닝차트로 직접 개발 및 디버깅은 라이트닝차트 EULA 규정에 의해 금해져 있습니다.

3.9 트라이얼 기간

트라이얼 기간은 30 일 사용 가능합니다. 그 이후 제품을 계속 사용 하기 위해 라이센스를 구매해야 합니다. 트라이얼 라이센스 기간 중 만든 모든 프로젝트는 제대로된 라이센스로 업데이트 후 사용 가능 합니다. 트라이얼 라이센스로 차트 어플리케이션을 실행 시 업그레이드 메시지가 계속 표시 됩니다.

3.10 유동 라이센스

무제한 수의 컴퓨터에 유동 라이센스 설치 가능 합니다. 동시 개발자 수는 아크션으로 의해 구성 되었습니다. 오직 구매된 수의 동시 사용자들만 동시에 라이트닝차트로 개발 할 수 있습니다. 개발자가 라이트닝차트 개발을 완료 후 다른 개발자가 사용 할 수 있기 전 10-15 분 기다려야 합니다.

개발자당 라이센스와 비슷하게 배포 키도 설정이 되어야 합니다.

유동 라이센스들은 기본으로 아크션 라이센싱 서버에 제어 됩니다. 개발 중 지속된 인터넷 연결 상태가 필요합니다.

고객 쪽 유동 라이센스 컨트롤러도 사용 가능합니다. 개발 컴퓨터들은 근거리 통신망을 통해 고객의 조직에서 실행되는 서비스에 연결합니다. 아크션 또는 다른 당사자들과 온라인 커뮤니케이션은 이루어 지지 않습니다. 라이센스 관련 컨트롤러 서비스 및 유동 라이센스에 대해서 아크션은 따로 설명서를 제공합니다.

4. 라이트닝차트 얼티밋 구성 요소

4.1 라이트닝차트® .NET 라이브러리 사용

라이트닝차트® .NET 구성 요소를 사용하기 위해 아크션 .dll 파일들이 참조에 추가 되어야 합니다. 이는 설치 폴더에서 찾을 수 있습니다. 어플리케이션 개발 때 다음과 같은 어셈블리들이 필요합니다:

윈폼:	Arction.WinForms.Charting.LightningChartUltimate.dll.
WPF 비바인딩:	Arction.Wpf.Charting.LightningChartUltimate.dll Arction.DirectX.dll Arction.RenderingDefinitions.dll
WPF 반바인딩:	Arction.Wpf.SemibindableCharting.LightningChartUltimate.dll Arction.DirectX.dll Arction.RenderingDefinitions.dll
WPF 바인딩:	Arction.Wpf.BindableCharting.LightningChartUltimate.dll Arction.DirectX.dll Arction.RenderingDefinitions.dll
시그널툴즈 사용 시:	Arction.WinForms.SignalProcessing.SignalTools.dll or Arction.Wpf.SignalProcessing.SignalTools.dll

위에 레퍼런스들이 추가되고 프로젝트 빌딩을 시작하면 자동으로 필요한 모든 어셈블리들이 출력 폴더에 복사 됩니다. 제 28 장에 라이트닝차트 어플리케이션을 배포하는 데에 어떤 어셈블리들이 필요한지 보여 줍니다.

Arction.DirectXFiles.dll 은 파일이 너무 크기에 초기화 시간을 증가 할 수 있기에 자동으로 레퍼런스로 추가되지 않습니다. 알맞은 DirectX 어셈블리들이 이미 시스템에 없을 때에만 필요 합니다. Arction.DirectXInit.dll 루틴은 존재하는 dll 을 확인하고 필요할때 로딩 합니다. 한번 로딩하면 라이트닝차트가 미래에 접근할 수 있게 DirectX-dll 들을 윈도우 임시 폴더에 써서 초기화 시간을 단속 시킵니다.

Arction.DirectXFiles.dll 을 레퍼런스에 포함 시키지 않고 대신 exe 옆에 복사하는 것을 추천 드립니다.

4.2 코드로 차트 생성

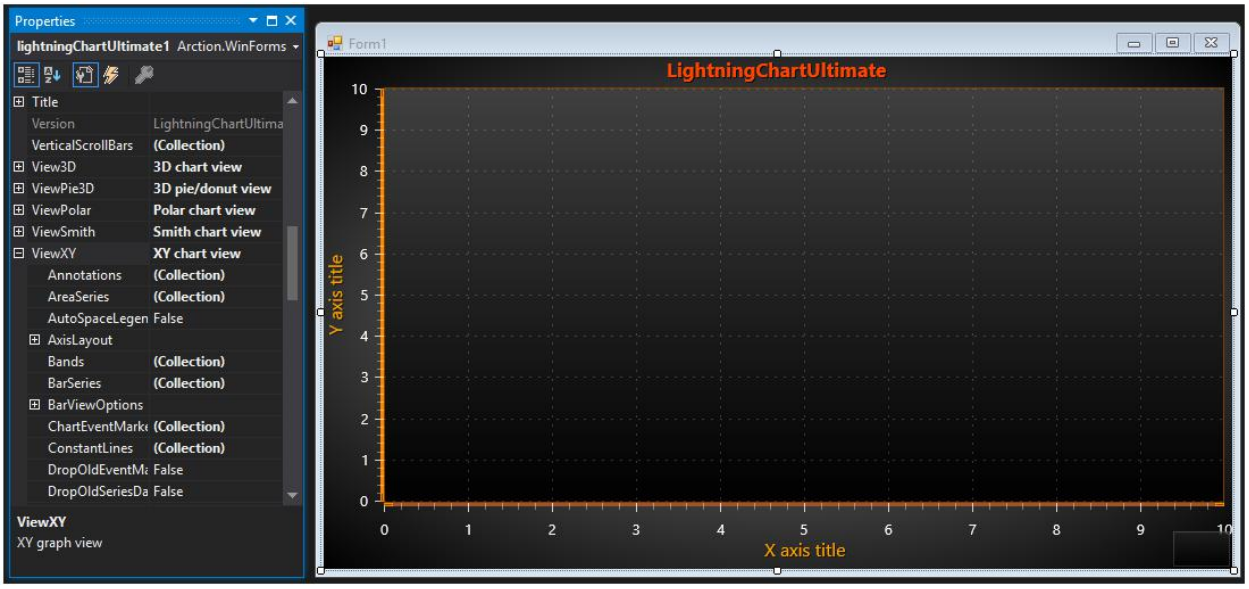
라이트닝차트 얼티밋 구성 요소는 툴박스에서 드래그 하거나 뒤에 온전히 코드로 생성해서 추가할 수 있습니다. 코드로 차트 개체를 생성하는 것을 더욱 편한 버전 업데이트란 장점이 있습니다. 또한 (역)직렬화 관련 이슈들을 피해 갈 수 있습니다.

다음은 뒤에 코드로 WPF 비바인딩 차트 생성하는 것을 보여 줍니다 (.xaml.cs -파일).

```
using Arction.Wpf.Charting;

namespace ExampleProject
{
    public partial class ExampleApp : Page
    {
        private LightningChartUltimate _chart = null;

        public ExampleApp()
        {
            InitializeComponent();

            CreateChart();
        }

        private void CreateChart()
        {
            _chart = new LightningChartUltimate();

            // Chart control into the parent container.
            (Content as Grid).Children.Add(_chart);

            // Disable rendering until the whole chart is set up correctly.
            _chart.BeginUpdate();

            // Configure chart here.

            // Allow rendering the chart.
            _chart.EndUpdate();
        }
    }
}
```

4.3 툴박스에서 윈도우 폼 프로젝트에 추가

라이트닝차트 얼티밋 제어판을 툴박스에서 폼으로 추가 하세요. 차트가 폼에 나타나고 이의 속성들이 **속성 창**에 나옵니다.



보기 5-1. 라이트닝차트 얼티밋 제어판이 윈도우 폼 디자이너에 추가됨.

4.3.1 속성

속성들은 자유롭게 변경 가능합니다. 또 신규 시리즈 및 기타 개체들도 컬렉션에 삽입 가능 합니다. 시리즈 데이터 포인트도 코드로 제시 되어야 합니다.

4.3.2 이벤트 핸들러

차트 메인 레벨의 이벤트 핸들러들은 속성 그리드로 할당 가능 합니다. 컬렉션들에 추가된 객체들 관련 이벤트 핸들러들도 코드로 할당 되어야 합니다.

4.3.3 버전 업데이트 관련 모범 사례

차트 속성 데이터는 비주얼 스튜디오 프로젝트 내 **.resx** 파일에 직렬화 되었습니다. 라이트닝차트 얼티밋 API 는 버전 업데이트 마다 살짝 변하는 성향이 있어 최신 버전의 **.resx** 파일 내 호환되지 않는 직렬화로 변질 될 수 있습니다.

보다 편리한 버전 업데이트를 위해 차트 개체 생성, 시리즈 및 이벤트 핸들러 추가를 코드로 하는 것을 강하게 추천 드립니다. 프로젝트는 이를 제대로 로딩하고 잠재 오류들은 컴파일 시간에 나와 .resx 파일보다 고치는 것이 쉬웁니다. .resx 파일에는 몇몇 속성 정의들은 손실 될 수 있으나 코드로는 항상 제시되어 있습니다.

4.4 툴박스에서 WPF 프로젝트에 추가

툴박스에서 윈도우 또는 다른 컨테이너에 **라이트닝차트 얼티밋** 반바인딩 또는 비바인딩 WPF 제어판을 추가 하세요. 디자이너에 차트가 보이고 속성이 **속성** 창에 나옵니다. XAML 에디터는 차트 기본 속성의 내용 및 수정을 보여 줍니다.

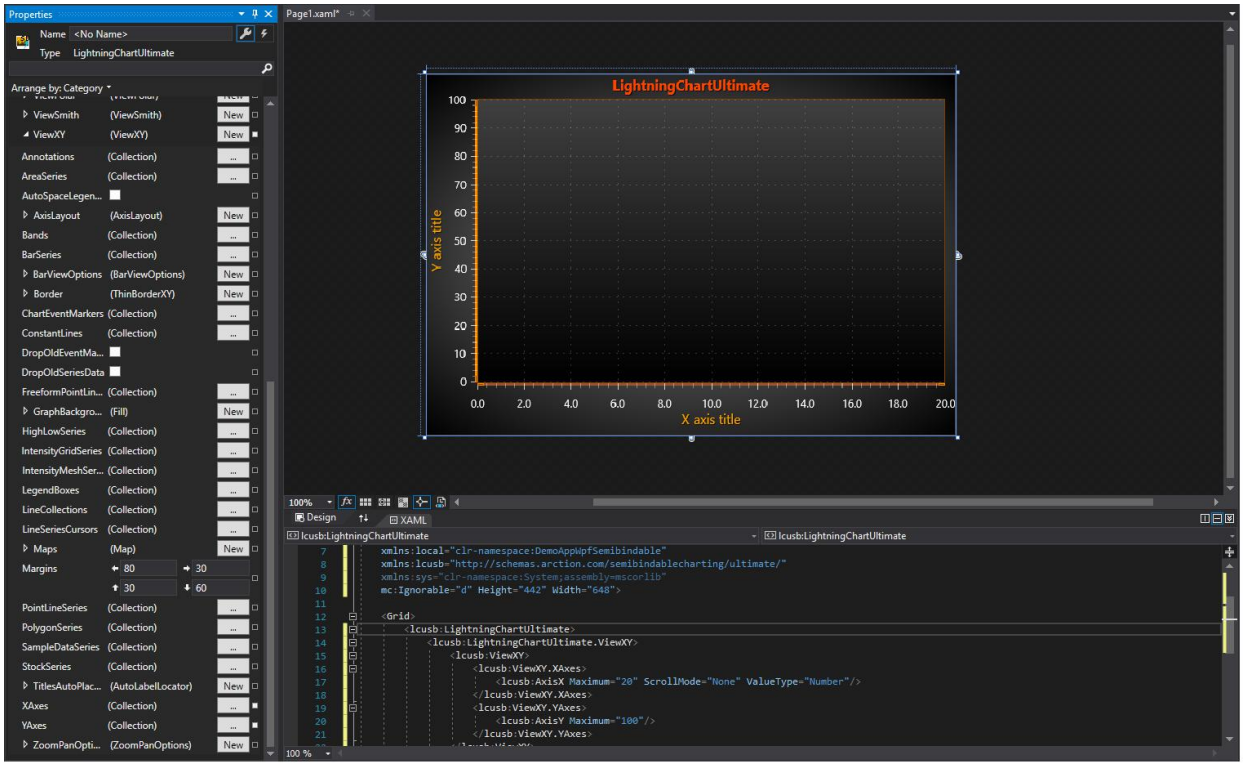
4.4.1 속성

속성들은 자유롭게 변경 가능하며 신규 시리즈 및 기타 객체들이 컬렉션에 삽입 가능 합니다.

- 반바인딩 차트에서는 시리즈 데이터 포인트는 코드로 쓰여져야 합니다
- 바인딩 차트에서는 시리즈 데이터 포인트들은 시리즈에 데이터를 바인딩 하거나 또는 뒤에 코드로 쓰여져도 됩니다.

4.4.2 이벤트 핸들러

차트 주요 레벨의 이벤트 핸들러들은 속성 그리드로 할당 가능 합니다. 컬렉션에 추가된 객체들 관련 이벤트 핸들러들은 코드로 할당이 되어야 합니다.

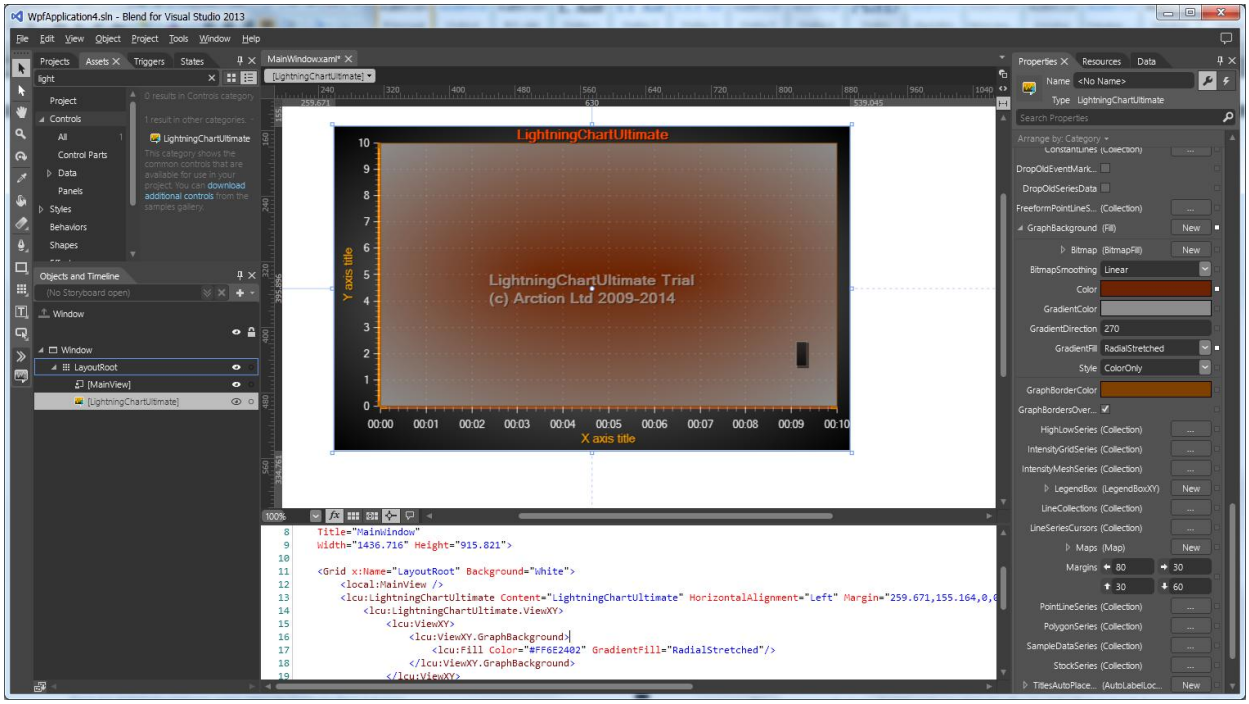


보기 5-2. WPF 디자이너에 추가된 라이트닝차트 얼티밋 제어판.

4.5 블렌드 WPF 프로젝트에 추가

프로젝트 탭에 **레퍼런스**로 가세요. 우클릭 해서 **레퍼런스 추가…**를 선택 하세요. **c:\program files (x86)\Arction\LightningChart .NET SDK v.8\LibNet4** 에서 Arction.WPF.Charting.LightningChartUltimate.dll 를 검색하세요.

자산 탭으로 가세요. 검색창에 “Lightning” 을 쓰세요. 검색 결과에 **라이트닝차트 얼티밋** 열을 찾을 수 있습니다. 객체를 WPF 창에 드래그 드랍 하세요.



보기 5-3. 비주얼 스튜디오 2013 디자이너를 위한 블렌드에 추가된 라이트닝차트 얼티밋 제어판.

4.5.1 버전 업데이트 관련 모범 사례

차트 속성 데이터는 XAML 로 저장 되었습니다.신규 버전들은 해당 라이트닝차트 얼티밋 객체가 디자이너에 보이지 않는 다른 속성 세트를 가질 수 있습니다. XAML 수정을 해야 합니다. XAML 태그 트리가 어마어마 할 수도 있어 수정이 어려울 수 있습니다.

보다 편한 업데이트를 위해 차트 객체를 생성하고 나열 및 정렬 속성 설정을 디자이너 내에서 하는 것을 추천 드립니다. 그 외 다른 것들은 코드로 설정하세요. 또는 차트 객체도 코드로 생성하세요.

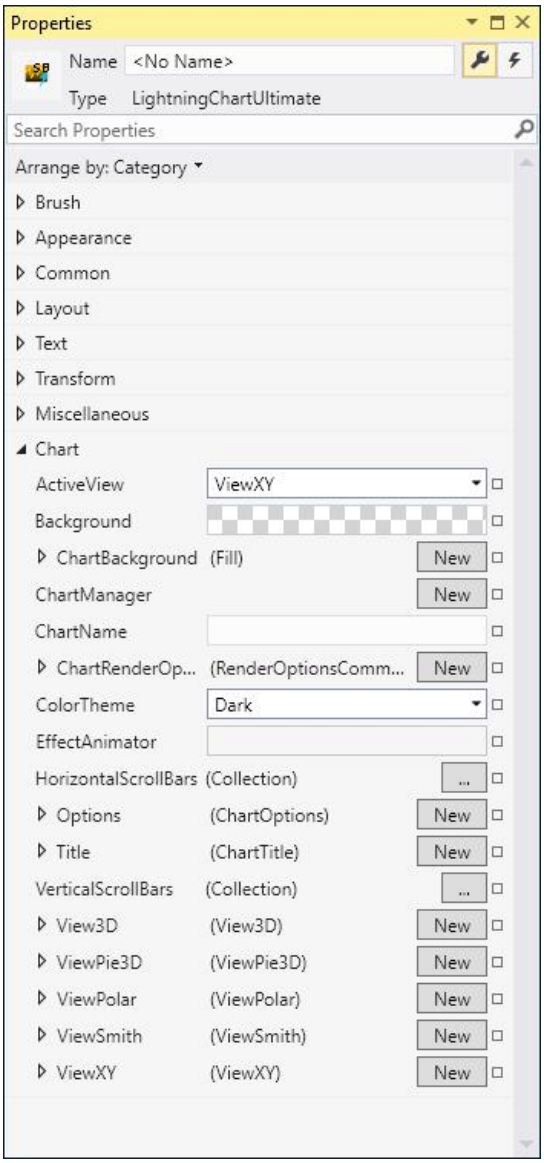
4.5.2 차트 흐려짐 방지

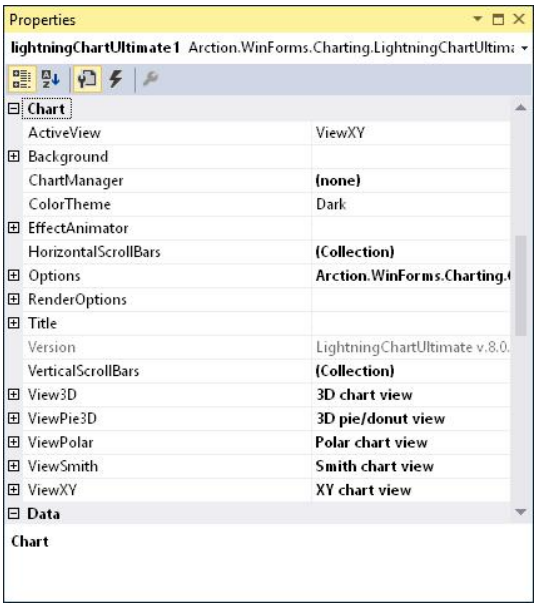
이는 WPF 에 흔한 특징이며 차트 자체와는 관계가 없습니다. 라이트닝차트에 정확한 렌더링을 하면 명확하게 보입니다.

차트가 흐리게 보이는 것을 방지하기 위해서 차트에 **패런트**가 되는 제어의 **레이아웃라운딩사용 = True** 로 세팅 하세요. 디자이너 내에서 차트가 아직 흐리게 보일 수도 있지만 응용 프로그램 실행 시 뚜렷하게 나올 겁니다. 패런트 제어는 예를 들어 **그리드, 캔버스, 도킹관리자** 등이 될 수 있습니다.

4.6 객체 모델

라이트닝차트의 객체 모델을 배우는 가장 좋은 방법은 비주얼 스튜디오의 속성 에디터를 사용해서 입니다.





보기 5-4. 라이트닝차트 관련 속성은 윈폼 및 WPF 속성 창의 차트 카테고리 아래 찾을 수 있습니다. 노드를 확장함으로 또는 WPF 내에서는 신규 객체를 생성함으로 거대한 세트의 속성을 볼 수 있습니다.

4.6.1 윈도우 폼과 WPF 의 차이점들

윈도우 폼과 WPF 의 차트 카테고리 관련 속성 트리 및 객체 모델은 거의 동일합니다. 가장 큰 차이점들은 다음과 같다:

	윈도우 폼	WPF
렌더링 옵션 속성	렌더 옵션	차트 렌더 옵션
배경 채우기 속성	배경	차트 배경
글씨체	System.Drawing.Font	Arction.WPF.LightningChartUltimate.WPFFont
색	System.Drawing.Color	System.Windows.Media.Color

표 4-1. 윈도우 폼 및 WPF 의 차이점들

이후 장들에 따로 표기가 없으면 윈도우 폼 속성 명이 사용 됩니다.

4.7 라이트닝차트 뷰

라이트닝차트는 다음과 같은 메인 뷰가 있습니다:

- 뷰 XY (제 6 장을 보세요)

- 뷰 3D (제 7 장을 보세요)

- 뷰파이 3D (제 9 장을 보세요)

- 뷰폴러 (제 10 장을 보세요)

- 뷰스미스 (제 11 장을 보세요)

액티브뷰 속성을 설정해서 뷰를 바꿀 수 있습니다. 기본 뷰 세팅은 뷰 XY 입니다.

```
// Set 3D as the visible view
chart.ActiveView = ActiveView.View3D;
```

4.8 뷰 및 줌 영역 정의

라이트닝차트 뷰는 담긴 정보에 따라 결정되는 여러 영역들이 있습니다. 이 영역들은 뷰의 내용 기반으로 2 차원 직사각형으로 볼 수 있다. 이 정의들은 뷰 종류에 상관 없이 균일하다. 특히 줌 조작에 차트의 어느 영역들을 보여줄 지 결정하는 데에 쓰인다.

- **차트영역/뷰영역:** 차트와 마진을 포함한 전체 영역.

-**마진직사각형:** 마진직사각형 (또는 MarginRect)는 마진 내의 영역도 포함한다.

-**그래프영역:** 축 범위에 따라 정의된 영역. 메이저 및 마이너 그리드를 포함한다. 데이터 값이 축 범위를 벗어나지 않는 이상 데이터는 이 영역 내에 그려져 있다.

-**배경 영역 / 원:** 그래프영역과 거의 일치하다. 그래프 축 범위 및 그리드 외 영역도 포함한다.

-**라벨영역:** 그래프 및 축 라벨로 구성된 영역. 데이터를 무시한다.

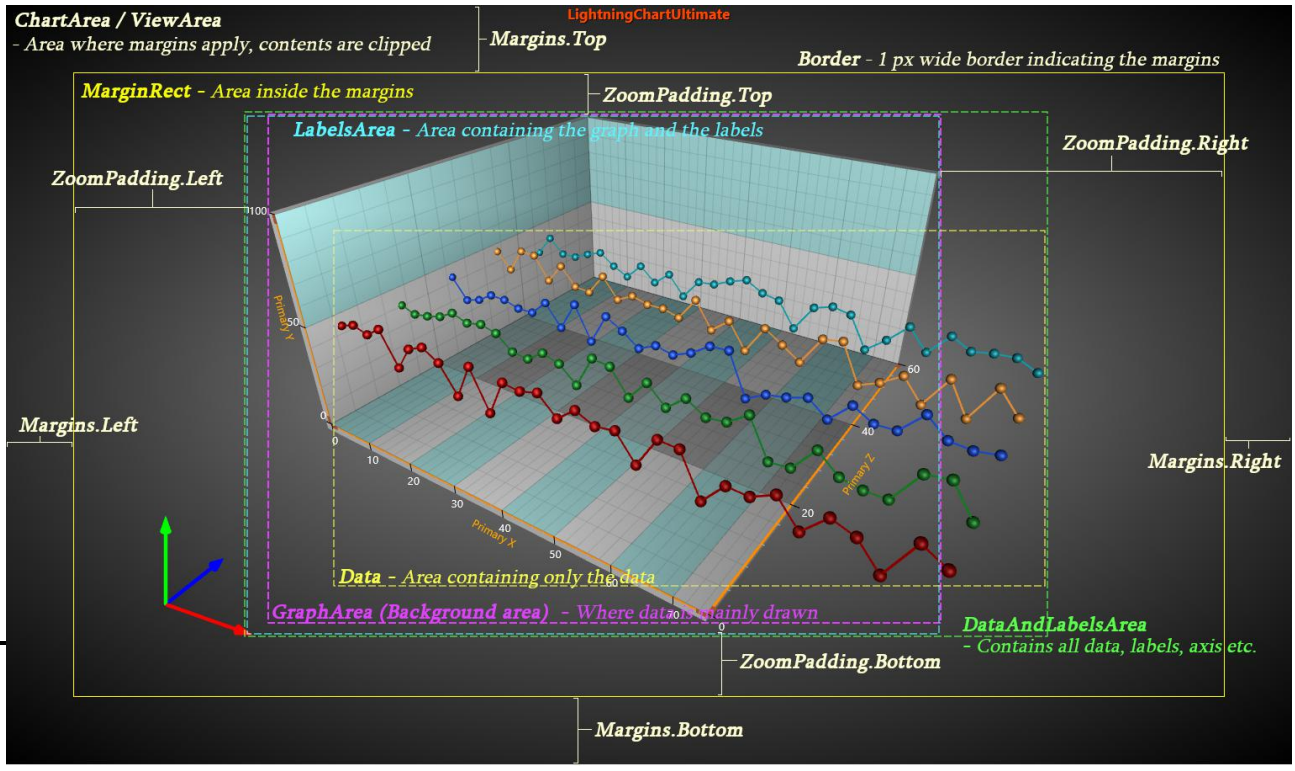
-**데이터:** 데이터로만 구성된 영역. 데이터의 최대 및 최소 값으로 정의 되었다.

-**데이터및라벨영역:** 데이터 및 라벨영역을 합침. 모든 데이터, 축, 라벨 및 마커가 포함 되었다.

-**태두리:** 마진의 위치를 표시하는 맞춤형 1 픽셀 넓이의 직사각형. 보이지 않게 설정 가능하다.

-**마진:** 마진은 그래프 영역 주변 공백 공간이다. 뷰 대부분의 내용은 마진 내에 위치하며 마진 외에는 잘렸다.

-**줌패딩:** 확대 조작 이후 마진과 사전에 정의된 영역 사이 남은 공간 (제 7.18.3 장을 보세요). 뷰 XY 에는 아무 효과 없습니다.



보기 5-5. 뷰 및 줌 영역 정의

4.9 배경 채우기 설정

모든 뷰는 공통된 배경 채움이 있습니다.

- 윈폼에서는 **chart.Background** 를 사용하세요.

```
chart.Background.Color = Color.DarkBlue;
```

- WPF 에서는 **chart.ChartBackground** 를 사용하세요.

```
chart.ChartBackground.Color = Colors.DarkBlue;
```

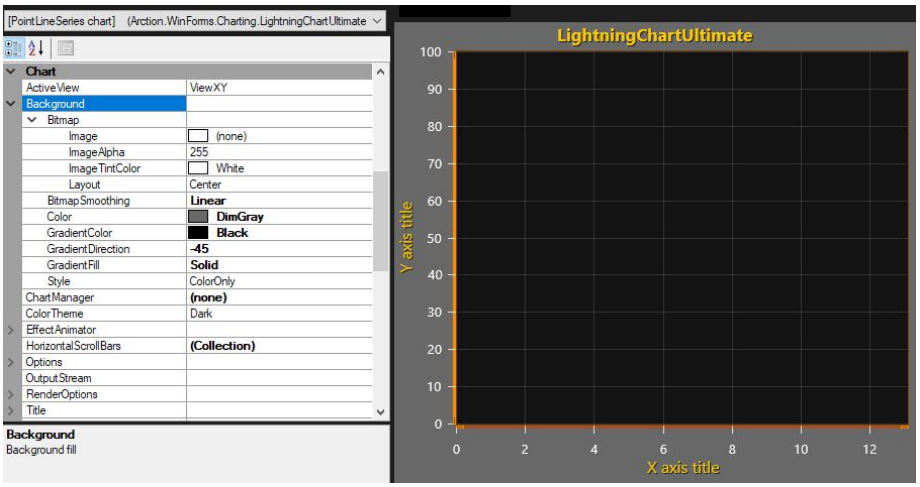
배경 채우기는 다음과 같이 지원합니다:

- 단색 채웃기. **GradientFill = Solid** 로 설정 후 **Color** 를 사용해 색을 정하세요.
- 구배 채우기, **Color** 에서 **GradientColor** 로 가는 것. GradientFill = Linear / Radial / RadialStretched / Cylindrical 로 설정하세요. **GradientDirection** 을 사용해 선형 및 원통형 구배 채우기 위치를 정하세요.

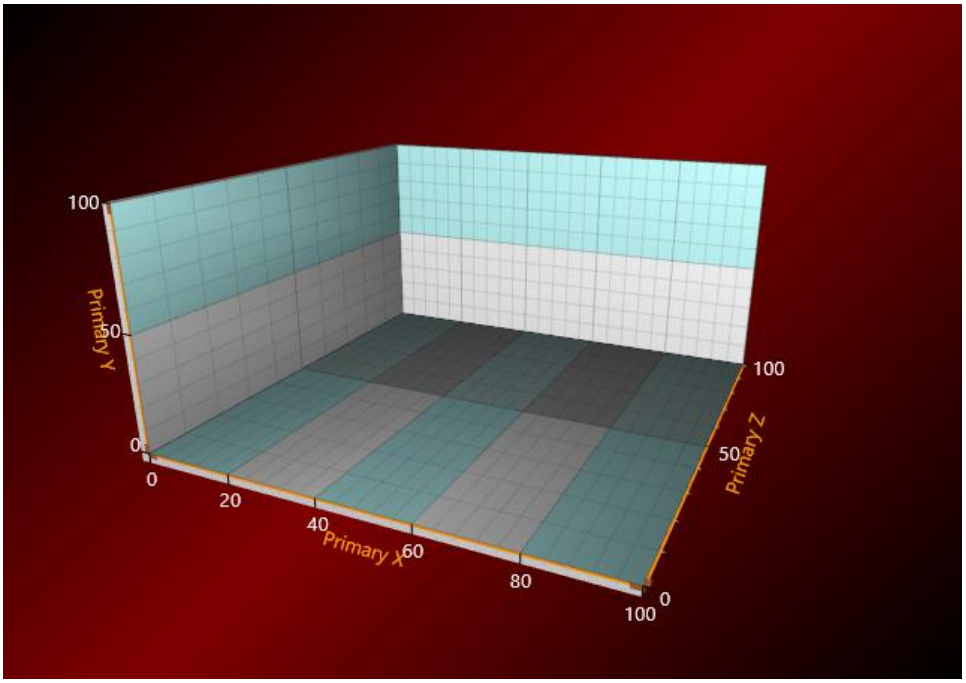
```
chart.ChartBackground.GradientFill = GradientFill.Cylindrical;
chart.ChartBackground.GradientColor = Colors.Black;
```

```
chart.ChartBackground.GradientDirection = -45;
```

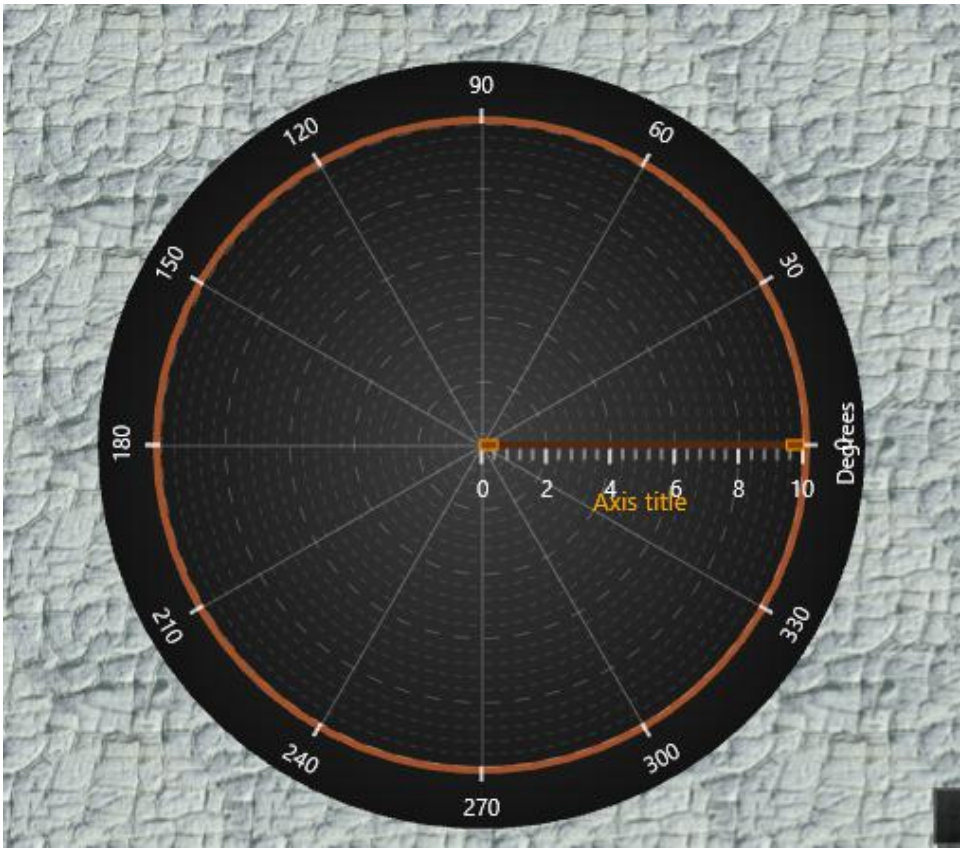
- 여러 기울기 및 스트레치 옵션으로 비트맵 채우기. 반투명 채우기가 가능한 비트맵 틴트 및 알파도 지원됩니다.



보기 5-6. 뷰 XY 아래 배경 설정. GradientFill = Solid, and Color = DimGray.



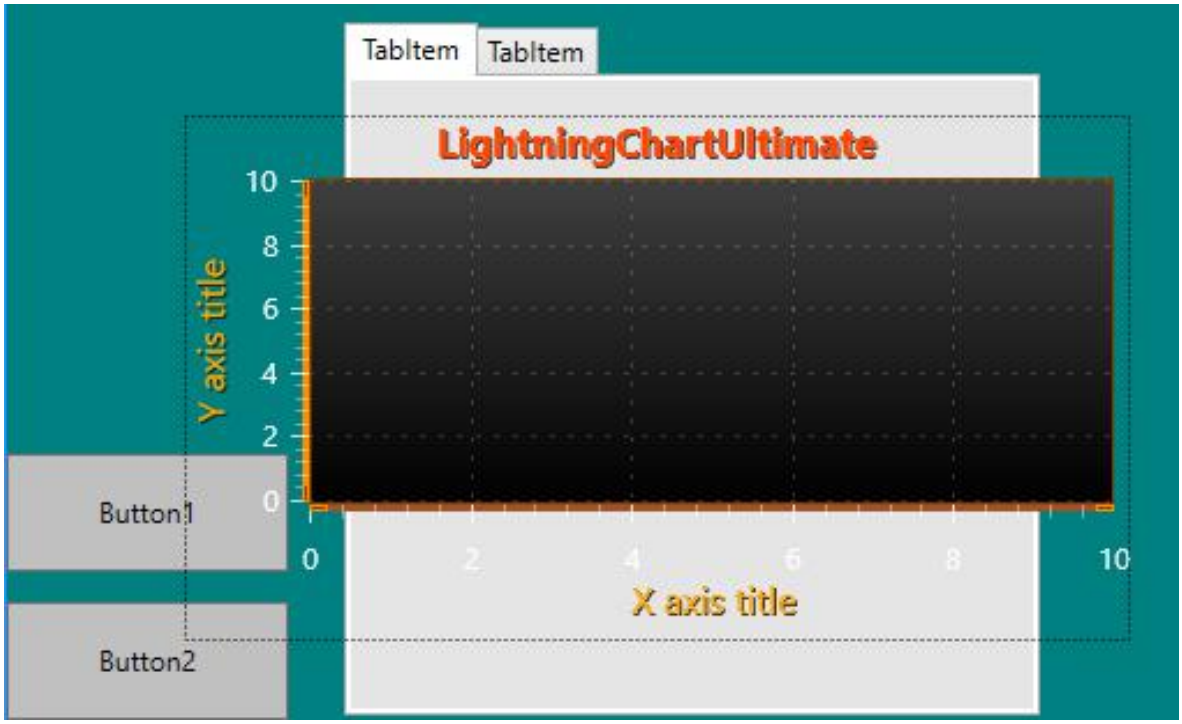
보기 5-7. 뷰 3D 아래 배경을 원통형 구배로 설정. GradientFill = Cylindrical, Color = Maroon, GradientColor = Black. GradientDirection = -45 degrees



보기 5-8. 기울인 비트맵 채우기로 배경 설정. Style = Bitmap, a picture set to Bitmap.Image 및 Bitmap.Layout = Tile, under ViewPolar

4.9.1 투명 배경 설정

WPF 내에서는 차트 아래에 놓여진 객체들도 비추어 질 수 있게 차트를 투명하게 만들 수 있다.



보기 5-9. WPF 차트 내 투명 배경.

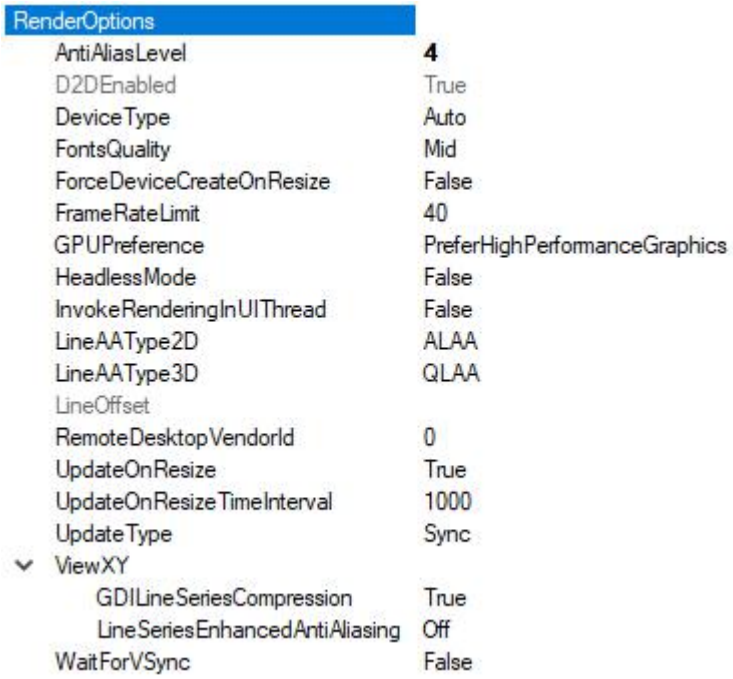
ChartBackground.Color = #00000000 로 설정하세요 (투명 검정).

주의! '투명'으로 설정하지 마세요 (#00FFFFFF). 뚫고 보여지지 않습니다.

윈폼은 제어들의 투명 배경을 지원하지 않습니다.

4.10 외관 구성 / 성능 설정

차트렌더 옵션 (윈폼에선 **렌더옵션**)에는 외관 및 성능 구성을 위한 속성이 있습니다.



보기 5-10. 렌더옵션 속성

DeviceType

```
// Changing the rendering device in code
chart.ChartRenderOptions.DeviceType = RendererDeviceType.Auto;
```

자동은 자동 D11 선호 옵션의 다른 이름이다. 이는 기본 설정이다.

자동 D9 선호 옵션은 DirectX9 하드웨어 렌더링을 선호하며 가용성에 따라 이 순서대로 기기를 선택한다: HW9 -> HW11 -> SW11 -> SW9. 사용 가능한 하드웨어가 없을 시 WARP (SW11) 소프트웨어 렌더링으로 폴백합니다.

자동 D11 선호 옵션은 DirectX11 하드웨어 렌더링을 선호하며 가용성에 따라 다음 순서대로 기기 선택을 한다: HW11 -> HW9 -> SW11 -> SW9. 사용 가능한 하드웨어가 없을 시 WARP (SW11) 소프트웨어 렌더링으로 폴백합니다. **고성능 및 최고 외관 설정으로 이 옵션을 사용하세요**. 비주얼 외관은 DirectX9 렌더 보다 낫습니다.

하드웨어온리 D9 은 오직 하드웨어 9 렌더링만 사용합니다.

하드웨어온리 D11 은 오직 하드웨어 11 렌더링만 사용합니다.

소프트웨어온리 D11 은 DirectX11 WARP 를 사용하고 DirectX9 레퍼런스 래스터라이저 비하면 매우 빠르지만 하드웨어 옵션 보다 느립니다.

소프트웨어온리 D9 는 DirectX9 레퍼런스 래스터라이저를 사용합니다 (매우 느림).

없음은 차트가 숨겨졌거나 배경에 비활성하면 **DeviceType** 을 **없음**으로 설정하면 그래픽 자료들을 다른 차트들을 위해 확보합니다.

GPUPreference

```
chart.ChartRenderOptions.GPUPreference = GPUPreference.SystemSetting;
```

듀얼 그래픽 어댑터 시스템에 해당되는 선택. 이는 대부분 저성능 그래픽 처리 장치 (GPU) 가 CPU/칩세트에 통합된 노트북들과 AMD 또는 Nvidia 등 고성능 그래픽 GPU 에 해당 됩니다.

시스템 설정은 윈도우 또는 AMD 또는 Nvidia 제어판 그래픽 설정에 선택된 옵션을 사용합니다.
고성능 그래픽 선호는 시스템에 존재하면 고성능 GPU 를 사용합니다. 주로 성능이 더욱 좋지만 에너지 소비량도 높을 수 있습니다.
저성능 그래픽 선호는 시스템에 고성능 GPU 가 설치 되었어도 더욱 느린 통합 GPU 를 사용합니다.
기본적으로 **고성능 그래픽 선호** 옵션 사용을 추천합니다. 최고 성능을 위해 이 옵션을 선택하세요.

FontsQuality

```
chart.ChartRenderOptions.FontsQuality  =  FontsRenderingQuality.High;
```

하 최고 성능. 글꼴이 안티 앨리어싱 안됨. 생김세가 흐트러지지 않게 글꼴 서체를 조심히 선택하세요.
중 은 **하**와 거의 비슷한 성능이다. 글꼴 주변 간단한 안티 앨리어싱 있다. 기본 설정이다.
고의 생김세가 최고지만 성능은 상당히 떨어진다.
주의: 고 퀄리티 설정으로 DirectX11 렌더링 할 때에 투명 배경은 해당이 안된다. DirectX9 에는 된다. 이는 렌더링 기술의 제한이다.

AntiAliasLevel

```
chart.ChartRenderOptions.AntiAliasLevel = 1;
```

전체적 씬 안티 앨리어싱 팩터. 가용성은 하드웨어에 따라 다르다. 더욱 높은 값은 외관을 좋게 나오게 하지만 성능이 떨어진다. 성능을 최대화하기 위해 0 또는 1 로 설정하세요. 유효한 안티 앨리어싱 설정에 대한 더 많은 정보를 위해서는 제 5.12 장을 보세요.

WaitForVSync

```
chart.ChartRenderOptions.WaitForVSync = true;
```

추천: 기본 값을 유지하세요. 활성화 되었을 때 디스플레이 다음 리프레시 중 렌더링을 홀드 한다 (예, 1/60 초의 배수). 이는 일시적으로만 사용하는 것을 추천합니다, 예를 들어 외부 화면 캡쳐 프로그램 사용 중 스트라이핑을 방지하기 위한 동기화, 또는 화면 내 이미지의 윗 부분이 아랫 부분과 동기화 되지 않았을 때 등. 끊어진 웨이브형 데이터를 보여줄 수 있습니다. WPF 에서 특히 활성화 되었을 때 성능이 상당히 저하 됩니다.

UpdateType

```
chart.ChartRenderOptions.UpdateType = ChartUpdateTypes.Sync;
```

Sync (기본): 차트가 동기적으로 업데이트 됩니다. 마지막 **EndUpdate()** 부르기 이후 또는 속성 설정 (메소드 부르거나)으로 인해 차트에 변화가 일어나면 업데이트가 됩니다. (**BeginUpdate…EndUpdate** 없이) 속성 변경은 바로 뉴 프레임 렌더링으로 이어집니다.

Async: 차트가 비동기 식으로 업데이트 됩니다. 속성 변경 이후 차트는 최대한 빠르게 업데이트를 하지만 나중에 차트가 신규 프레임을 렌더링 합니다. 이것은 몇몇 경우에 더욱 쉬운 차트 사용을 허용합니다.

LimitedFrameRate (제한된 프레임 레이트): FrameRateLimit 속성에 제시 되어있는 값으로 프레임 레이트가 제한 되어 있습니다. 0 = 무제한. 예) 최대 초당 10 번의 리프레시를 원하시면 10 의 값을 설정 하세요. 이는 비동기 옵션과 비슷하지만 첫 프레임 렌더링 직후 신규 프레임 렌더링을 방지함으로 프레임 레이트를 줄이지만 시스템 자원을 아낍니다.

주의! LimitedFrameRate 및 비동기 모드에서 알맞은 스레드 핸들링 설정을 확인하세요. 차트가 비동기 식으로 업데이트 되고 차트 속성이 동시에 업데이트 되면 충돌이 일어나 차트 또는 응용 프로그램에 에러가 발생 할 수 있습니다.

InvokeRenderingInUIThread

```
chart.ChartRenderOptions.InvokeRenderingInUIThread = true;
```

응용 프로그램에서 배경 스레드를 사용 시 스레드의 모든 UI 업데이트는 Invoke (윈폼에서는 **Control.Invoke()** 및 WPF 에서는 **Dispatcher.Invoke()**)를 통해서만 가야 합니다.

활성화 되었을 때 렌더링 부분은 내부 Invoke 를 통해 UI 스레드를 합니다.

기본 설정 값은 **False** 입니다. 이는 이 속성이 활성화 되어도 차트 내부에서 스레드 충돌 방지를 위해 속성 설정 및 메소드 부르기를 스레드 안전한 방법으로 해야하기 때문입니다.

HeadlessMode

```
chart.ChartRenderOptions.HeadlessMode = true;
```

이 세팅을 **True** 로 설정함은 콘솔 응용 프로그램 또는 유저 인터페이스가 없는 기타 응용 프로그램 등에서 차트를 배경 서비스에서 사용을 허용합니다. 제 22 장을 보세요.

4.11 DPI 핸들링

기본으로 WPF 응용 프로그램들은 윈폼 어플리케이션들과 다르게 DPI (인치당 도트 수) 인식을 합니다. 또한 크기를 제기 위해 픽셀 대신 DPI 를 사용합니다. 라이트닝차트는 모니터당 DPI 인식을 지원하지 않고 시스템 인식을 지원한다. 이는 WPF 어플들도 DPI 시스템 인식을 한다는 것이다. 윈폼에서 기본 DPI 설정은 72 지만 wpf .dll 파일들을 로딩 시 값이 96 으로 변경되는 점을 주의하는 것이 좋다.

하지만 라이트닝차트는 다른 DPI 설정이 있는 다른 화면으로 옮겼을 때 자동으로 크기 조정이 되지 않습니다. 이 기능을 활성화 시키려면 **ChartOptions** 아래 **AllowDPIChangeInduceWindowsResize** 속성 값을 **True** 로 설정해야 합니다. 또는 사용자는 **OnDPIChanged** 이벤트에 등록하여 이의 **allowWindowResize** 속성을 변경 할 수 있습니다. 위 방식들은 윈폼에 아무런 효과가 없습니다.

```
// Enabling automatic resizing
chart.Options.AllowDPIChangeInduceWindowResize = true;


// Via OnDPIChanged -event
chart.OnDPIChanged += chart_OnDPIChanged;
```

```
private void chart_OnDPIChanged(LightningChartUltimate chart, float dpix,
float dpiy, ref bool allowWindowResize)
{
        allowWindowResize = true;
}
```

4.11.1 DpiHelper 클라스

라이트닝차트는 DPI 관련 문제에 도움을 주는 **DpiHelper** 클라스가 있습니다.

DpiAware 는 시스템 프로세스가 DPI 인식 하는지 안 하는지를 말해 줍니다. 하지만 현재 이는 시스템 인식과 모니터당 인식 사이 구별은 못합니다.

```
bool isDPIAware = DpiHelper.DpiAware;
```

DpiXFactor/ DpiYFactor 는 화면 넓이/높이 시스템 DPI 의 효과적인 줌 팩터 입니다. X/Y 방향으로 1 DPI 에 몇 픽셀이 있는지 설명하는 팩터 입니다.

```
float dpiXFactor = DpiHelper.DpiXFactor;
```

DipToPx 및 **PxToDip** 메소드들은 시스템 DPI 설정을 사용해 DPI 와 픽셀을 바꿉니다. 싱글 포인트 또는 픽셀을 바꿀 수 있고 또는 직사각형의 크기 및 위치 값을 변경할 수 있습니다.

```
double pixelValue = DpiHelper.DipToPx(dipValue);
```

4.12 안티 앨리어싱

LightningChart® .NET 는 안티 앨리어싱 렌더링을 지원 합니다. **안티 앨리어싱** 속성이 있는 객체들에 적용이 됩니다. 안티 앨리어싱으로 라인 등의 가장자리를 부드럽게 렌데링 할 수 있으며 더욱 세련된 그래픽 표현 방식이 가능합니다. 이는 CPU/GPU 에 더욱 큰 부담을 가하여 기능적인 면에서 손실을 봅니다.

4.12.1 안티 앨리어싱 활성화

안티 앨리어싱은 **AntiAliasing** 속성을 통해 제어 가능 합니다. 이는 관련 구성 요소에 따라 부울 값 또는 **LineAntialias** 열거로 설정 되었습니다. 열거 설정은 현재 두가지 옵션이 있습니다:

```
LineAntialias.None;      안티 앨리어싱 없음
LineAntialias.Normal;    안티 앨리어싱
```

```
seriesEventMarker.Symbol.Antialiasing = true;

pointLineSeries.LineStyle.AntiAliasing = LineAntialias.Normal;
```

안티 앨리어싱은 차트의 **AntiAliasLevel** 의 영향을 받는다. 이는 선택 렌더링 엔진 (DirectX9 및 DirectX11) 에 따라 적용 안티 앨리어싱 모드를 정의하는 팩터이다. 안티 앨리어싱 레벨을 0 또는 1 로 설정 함은 개별 구성 요소의 **AntiAliasing** 속성이 참으로 또는 LineAntialias.Normal 으로 설정 되었어도 안티 앨리어싱이 렌더링에 적용 되지 않게 됩니다.

AntiAliasLevel 는 차트의 렌더링 옵션을 통해 설정 가능 합니다:

```
// Anti-aliasing factor. Values 0 and 1 result into no anti-aliasing applied.
chart.ChartRenderOptions.AntiAliasLevel = 2;
```

수동으로 값 설정을 안하면 **AntiAliasLevel** 은 기본 4 로 설정 됩니다.

4.12.2 DirectX11 안티 앨리어싱

DirectX11 에는 안티 앨리어싱으로 렌더링 돌릴 때 신경 써야할 몇가지의 일반적인 특징이 있습니다:

- **AntiAliasLevel** 을 1 이상의 값으로 설정함은 **AntiAliasing** 속성을 우세하게 됩니다. 이는 **AntiAliasing** 속성이 거짓 또는 `LineAntialias.None` 으로 설정 되었어도 렌더링이 안티 앨리어싱으로 된다는 것입니다. `LineOptimization.Hairline` 이 적용 되었을 때에만 위의 경우가 해당 되지 않습니다 (3D 렌더링에만 가능).
- **LineAntiAliasType** 는 알파블렌딩 (ALAA) 또는 사각형 안티 앨리어싱 (QLAA) 사이 선택하는 데에 사용할 수 있습니다:

```
LineAntiAliasingType.ALAA;      알파 블렌딩 안티 앨리어싱.
LineAntiAliasingType.QLAA;      사각형 안티 앨리어싱.
```

```
chart.ChartRenderOptions.LineAAType2D = LineAntiAliasingType.ALAA;
```

RasterizerStateDescription 의 **IsMultisampleEnabled** 및 **IsAntialiasedLineEnabled** 설정들도 아래와 같이 QLAA 와 ALAA 렌더링에 효과를 줍니다:

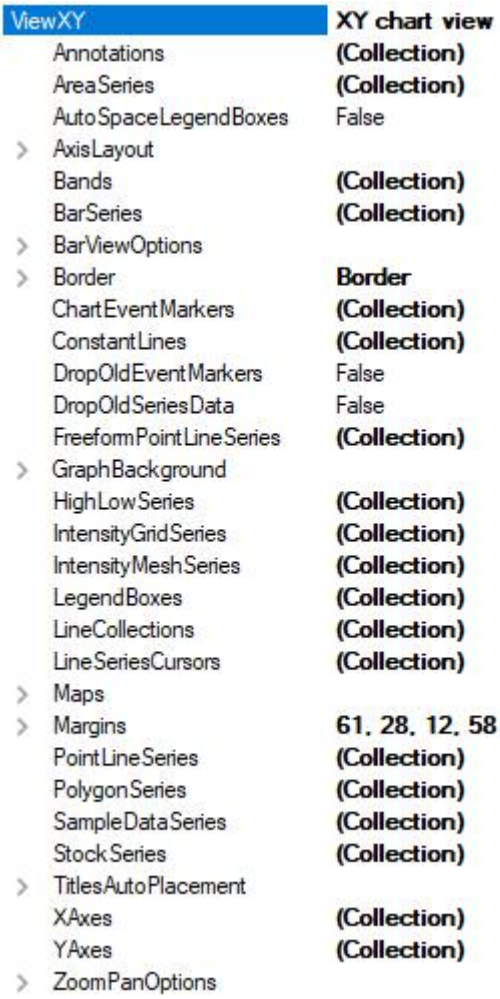
- **RasterizerStateDescription.IsMultisampleEnabled** == true 일 시, QLAA 사용.
- **RasterizerStateDescription.IsMultisampleEnabled** == false, ALAA 사용.
- **RasterizerStateDescription.IsAntialiasedLineEnabled** == true, ALAA 사용. 이는 **RasterizerStateDescription.IsMultisampleEnabled** == false 일 시에만 효과가 있습니다.

주의! DirectX11 으로 3D 렌더링에 대해서, 모든 세모 선들은 AntiAliasLevel 값이 0 또는 1 로 설정 되지 않을 시 안티 앨리어싱으로 렌더링 됩니다.

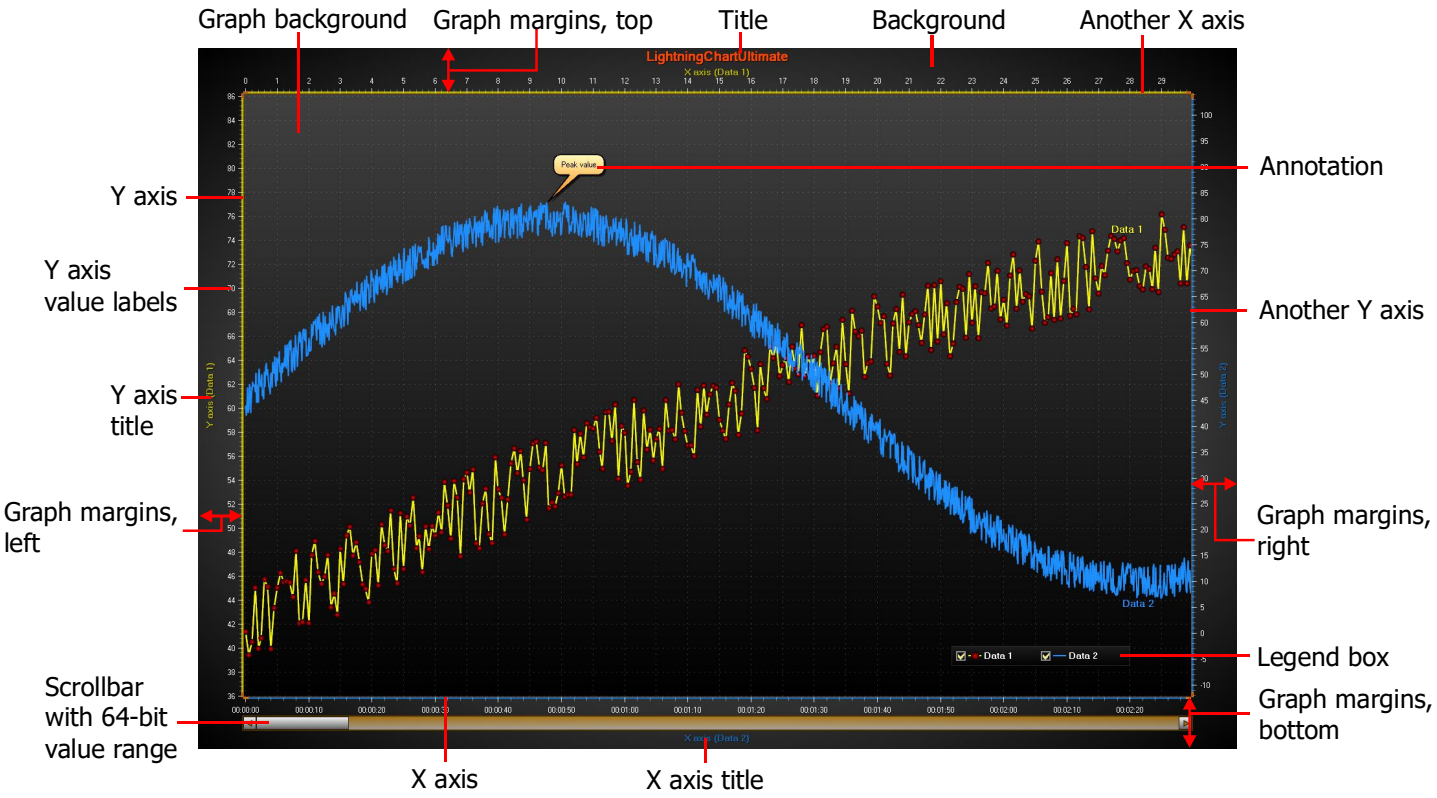
5. ViewXY

ViewXY 는 여러 포인트 라인 시리즈, 지역 시리즈, 하이 로우 시리즈, 강도 시리즈, 히트맵, 바 시리즈, 밴드, 라인 시리즈 커서 등을 직교 XY 그래프 형식으로 표시하는 것을 허용 합니다. 시리즈들은 X 와 Y 축들에 묶여 있고 지정된 축들의 값 범위를 사용합니다.

ViewXY 는 지리적인 지도 또한 보여줄 수 있습니다, 더 많은 정보를 위해 제 6.25 장을 보세요.



보기 6-1. ViewXY 객체 나무.



보기 6-2. ViewXY 의 개요.

그래프 마진

축 값과 설정으로 자동으로 마진은 조정이 됩니다. 값을 **ViewXY.AxisLayout.AutoAdjustMargins = False** 으로 설정하는 것은 **마진** 속성이 적용되어 마진 크기를 수동 설정을 허용 합니다. 그래프가 전체 뷰 구역을 채우기 위해 모든 마진을 0 으로 설정하세요. 더 많은 정보를 위해 제 6.4 장을 보세요.

그래프 경계

그래프 주변 마진들의 위치에 경계선이 그려져 있다. **Border** 속성을 이용해 경계선의 색과 투명도 및 시리즈 뒤에 렌더링 여부도 정할 수 있습니다. 제 6.4 장에서 경계선에 대해 더 배우세요.

배경

Background (WPF 에서는 ChartBackground) 속성을 통해 배경 채우기를 설정 하세요. 많은 채우기 옵션들이 있습니다. 제 5.9 장을 보세요.

그래프 배경

GraphBackground 속성을 통해 그래프 배경 채우기를 설정하세요. 그리드, 시리즈, 시리즈 커서, 이벤트 마커 등의 렌더링은 모두 그래프에서 됩니다.

```
chart.ViewXY.GraphBackground.Color = Colors.DarkBlue;
```

제목

차트의 메인 제목입니다. 텍스트, 색, 텍스트 태두리, 회전, 글씨체, 정렬 등을 **Title.Text, Title.Shadow…** 속성들로 설정하세요.

```
chart.Title.Text = "Title text";
```

Y-축

세로 축들은 Y 값을 표기합니다. 제 6.2 장을 보세요.

X-축

가로 축들은 X 값을 표기합니다. 제 6.3 장을 보세요.

주석

주석은 마우스 대화식 텍스트 라벨 또는 그래픽을 그래프 구역 어디에든 표시하는 것을 허용합니다. 제 6.20 장을 보세요.

표제 상자

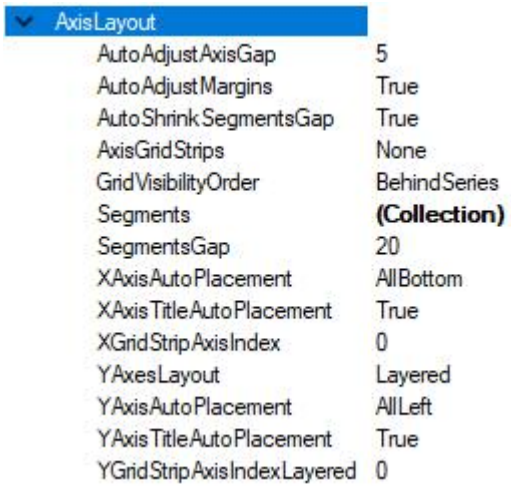
차트의 모든 시리즈의 목록. 제 6.21 장을 보세요.

스크롤바

어마어마한 수의 표본 지수를 직접 지원하기 위한 64-비트 값 범위 비지정 스크롤바. **HorizontalScrollBars** 및 **VerticalScrollBars** 들은 차트 루트 레벨에 있는 컬렉션 속성이지만 ViewXY 의 마진을 인식합니다. 제 13 장을 보세요.

5.1 축 레이아웃 옵션

축 위치 선정, 자동 마진 등을 조정하는 기본 속성들은 **ViewXY.AxisLayout** 속성 및 하위 속성에서 찾을 수 있습니다.



보기 6-3. AxisLayout 속성 트리.

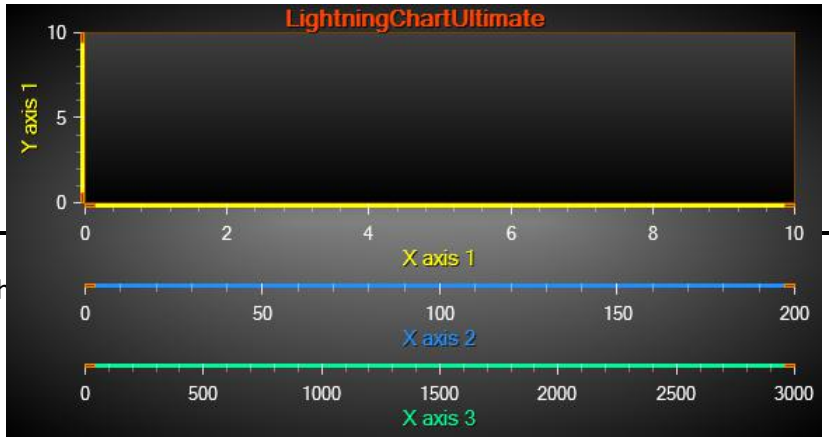
5.1.1 축 위치 선정 설정

5.1.1.1 X-축 자동 위치 선정

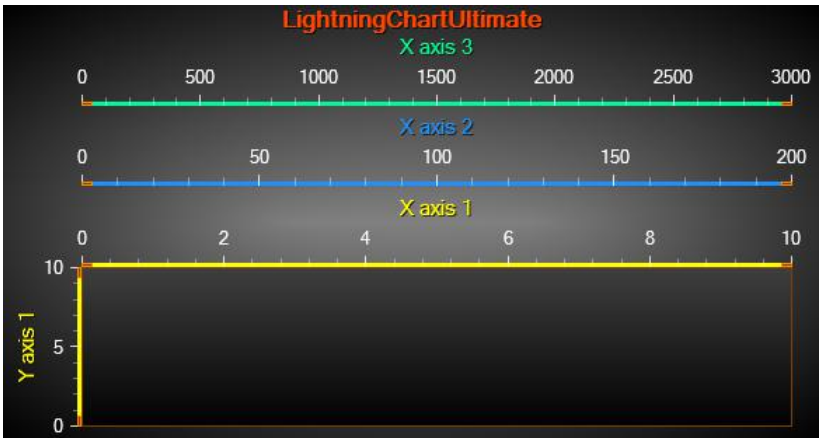
데모 예시: 자동 축 위치 선정; 여러 축

XAxisAutoPlacement 는 X 축의 수직 위치 선정을 제어한다.

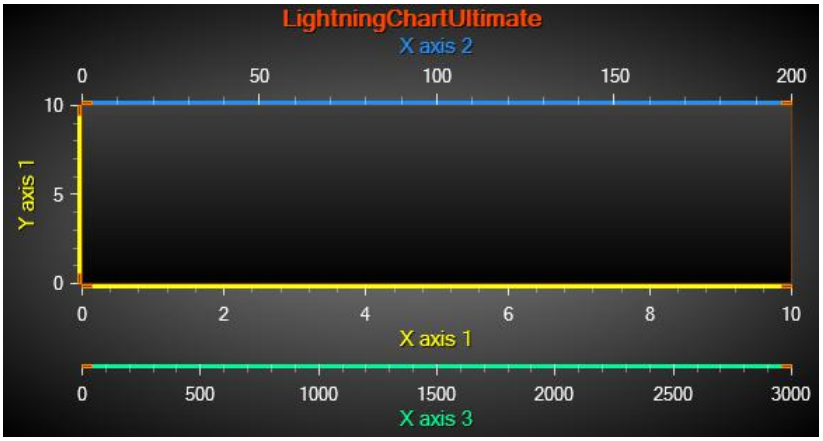
```
chart.ViewXY.AxisLayout.XAxisAutoPlacement = XAxisAutoPlacement.AllBottom;
```



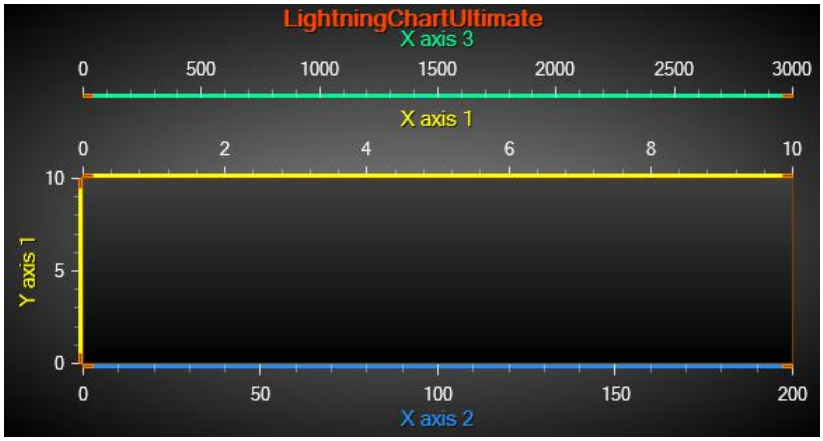
보기 6-4. XAxisAutoPlacement = AllBottom. 세 축이 추가 되었고 모두 그래프 아래에 위치 되었다.



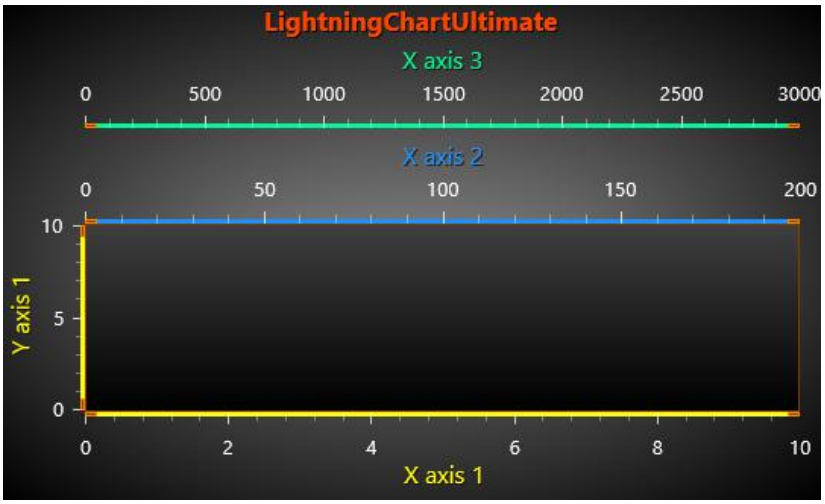
보기 6-5. XAxisAutoPlacement = AllTop. 모든 X 축들이 그래프 위에 위치 되었다.



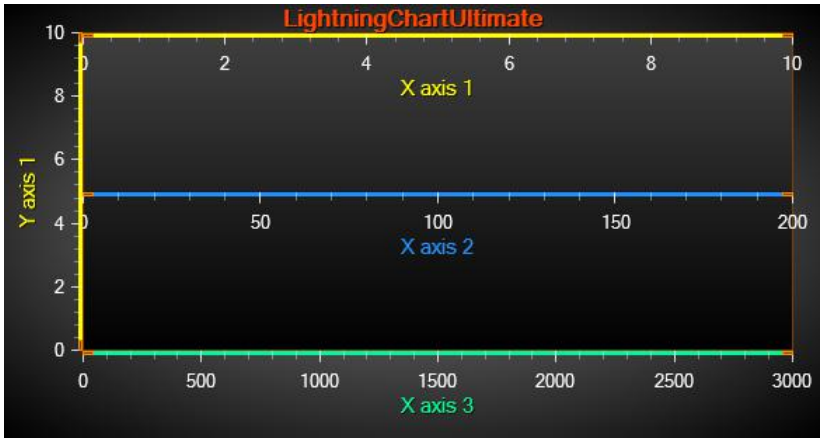
보기 6-6. XAxisAutoPlacement = BottomThenTop. 축들이 그래프 위 아래로 분배 되었다. 아래부터 시작되어 그래프 위 아래로 번갈아 배치 된다.



보기 6-7. XAxisAutoPlacement = TopThenBottolm. 축들이 그래프 위 아래로 분배 되었다. 그래프 위 부터 배치 되어 반대쪽으로 번갈아 가며 배치 된다.



보기 6-8. XAxisAutoPlacement = Explicit. 축이 선택되고 명시된 쪽에 생성된다. XAxis1 은 ExplicitAutoPlacementSide 속성이 Bottom 으로 설정 되었고 XAxis2 와 XAxis3 는 Top 으로 설정 되었다.



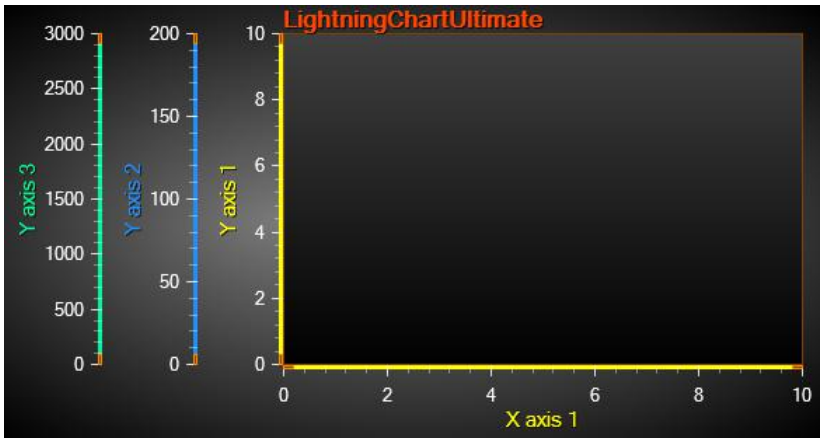
보기 6-9. XAxisAutoPlacement = Off. 자동 축 위치 선정이 비활성화 되었다. 축마다 위치 및 정렬 속성이 따로 적용 된다. 첫째 축은 Position = 0, 두번째는 Position = 50, 세번째는 position = 100 이다.

5.1.1.2 Y-축 자동 위치 선정

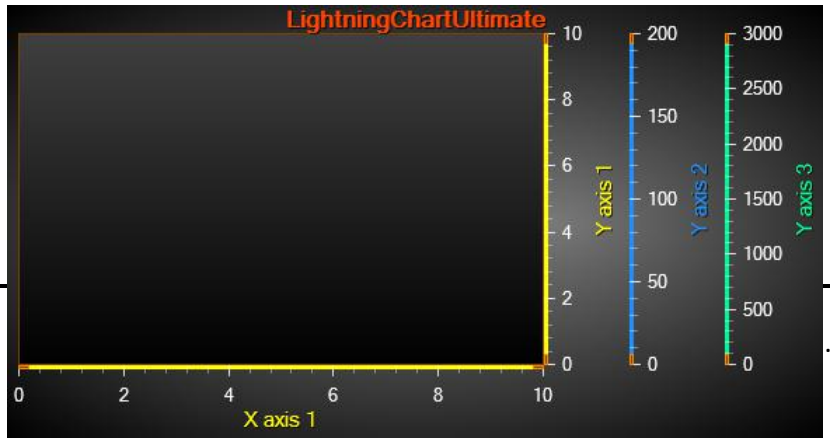
데모 예시: Y 축 레이아웃; 자동 축 위치 선정; 여러 축

YAxisAutoPlacement 은 Y-축의 가로 위치 선정을 제어합니다.

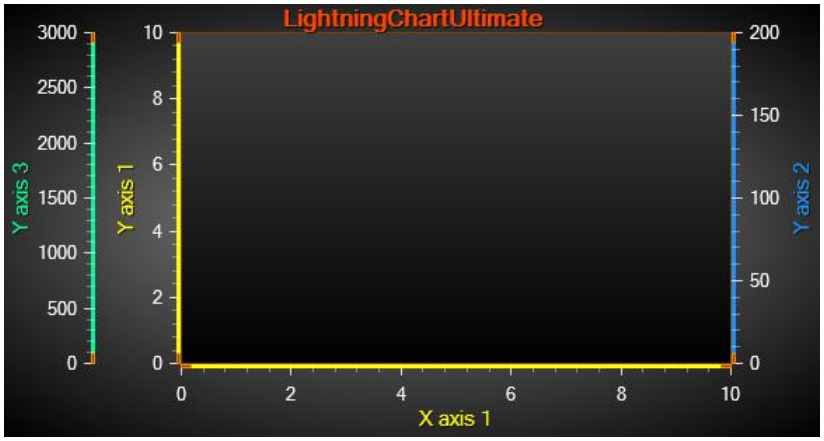
```
chart.ViewXY.AxisLayout.YAxisAutoPlacement = YAxisAutoPlacement.AllLeft;
```



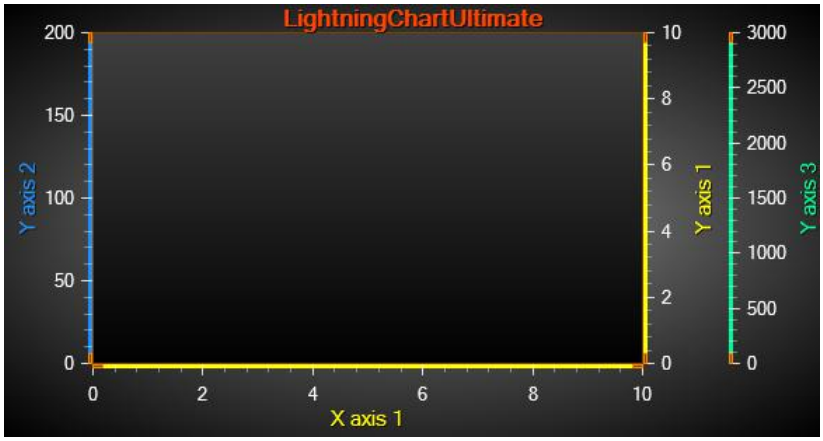
보기 6-10. YAxisAutoPlacement = AllLeft. 세개의 Y 축 추가. 모두 그래프 왼쪽에 위치.



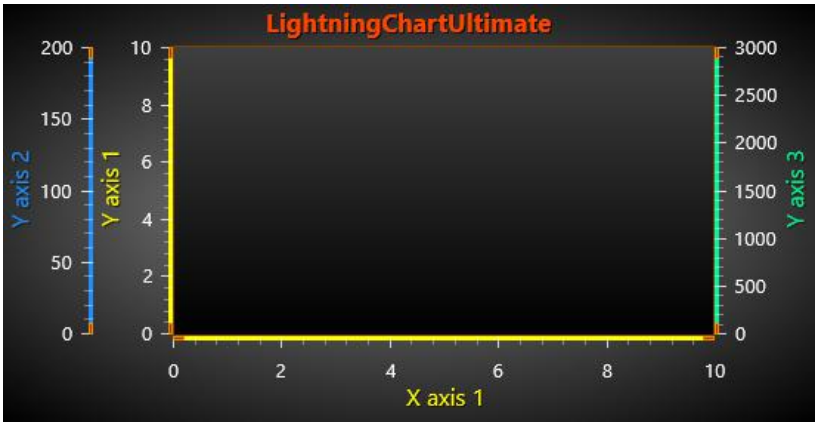
보기 6-11. YAxisAutoPlacement = AllRight. 모든 Y 축들이 그래프 우측에 위치 되었다.



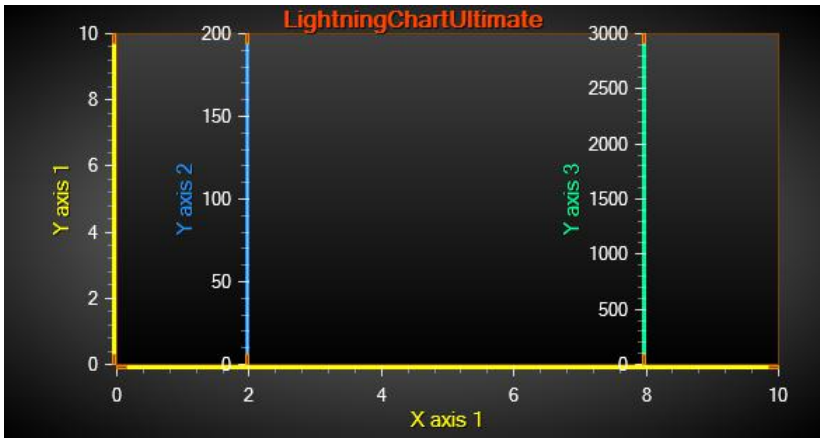
보기 6-12. YAxisAutoPlacement = LeftThenRight. 축들이 그래프 좌 우로 배정된다. 좌측부터 시작되어 번갈아 가며 배정 된다.



보기 6-13. YAxisAutoPlacement = RightThenLeft. 축들이 그래프 좌 우로 배정된다. 우측부터 시작되어 번갈아 가며 배정 된다.



보기 6-14. YAxisAutoPlacement = Explicit. 축들이 그래프 지정된 위치에 나타난다. YAxis1 와 YAxis2 는 ExplicitAutoPlacementSide 속성이 Left 로 설정 되었고 YAxis3 는 Right 로 설정 되었다.



보기 6-15. YAxisAutoPlacement = Off. 자동 축 위치 선정이 비활성화 되었으며 위치 및 정렬 속성이 축마다 따로 적용 된다. 첫번째 축은 Position = 0, 두번째는 Position = 20, 세번째는 position = 80 이다.

5.1.2 그래프 세그먼트 및 그 내 Y 축 위치 선정

여러 Y 축들이 정의 되었을 시 3 가지의 다른 방식으로 세로 정렬이 가능하다: **레이어, 스택**과 **세그먼트.**

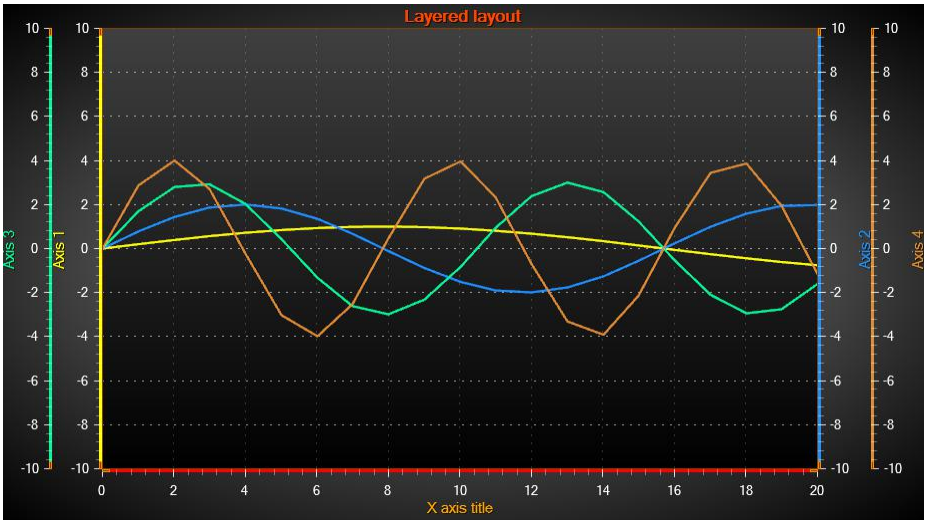
ViewXY.AxisLayout.YAxesLayout 속성으로 선택 가능하다.

5.1.2.1 레이어

데모 예시: Y 축 레이아웃; 자동 축 위치 선정

레이어 뷰에서는 모든 Y 축들이 그래프 위에서 부터 시작되어 그래프 하단까지 늘어진다. 축과 그에 묶인 시리즈는 같은 세로 스페이스를 공유한다.

```
chart.ViewXY.AxisLayout.YAxesLayout = YAxesLayout.Layered;
```



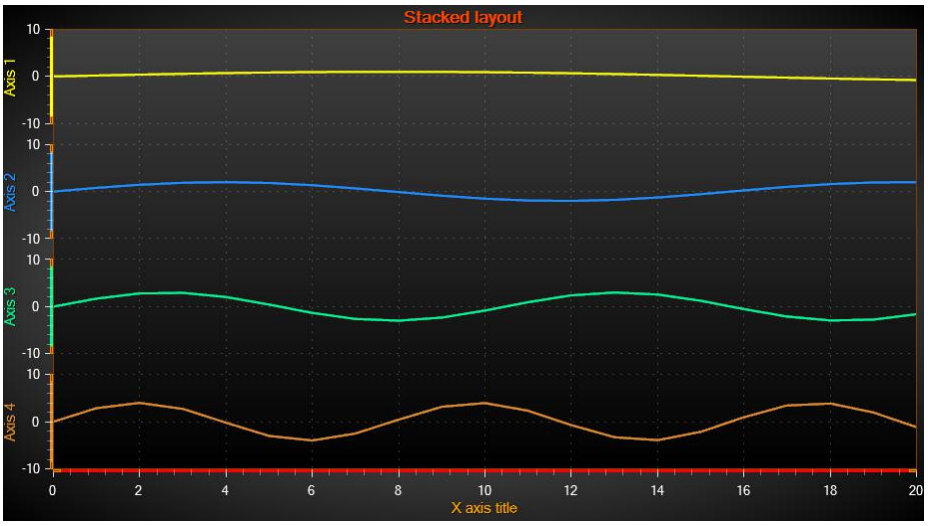
보기 6-16. YAxesLayout = Layered 에 있는 4 가지 Y 축의 예시.

5.1.2.2 스택

데몬 예시: Y 축 레이아웃; 멀티 채널 커서 트래킹; 시리즈로 데이터 브레이크

스택 뷰로는 각 Y 축이 자기만의 세로 스페이스가 지정된다. 모든 Y 축들은 같은 높이가 있다.

```
chart.ViewXY.AxisLayout.YAxesLayout = YAxesLayout.Stacked;
```



보기 6-17. YAxesLayout = Stacked 에 4 가지의 Y 축들의 예시.

5.1.2.3 세 그 먼 트

데모 예시: Y 축 레이아웃; 여러 레전드; 스플리터 있는 세그먼트

세그먼트 뷰에서는 세로 스페이스가 **세그먼트**로 나뉘어 졌습니다. 각 세그먼트에는 여러 Y 축들이 존재할 수 있다. 각 세그먼트의 관계적 높이는 설정 가능하며 세그먼트 내 각 Y 축은 세그먼트의 높이를 갖게 된다.

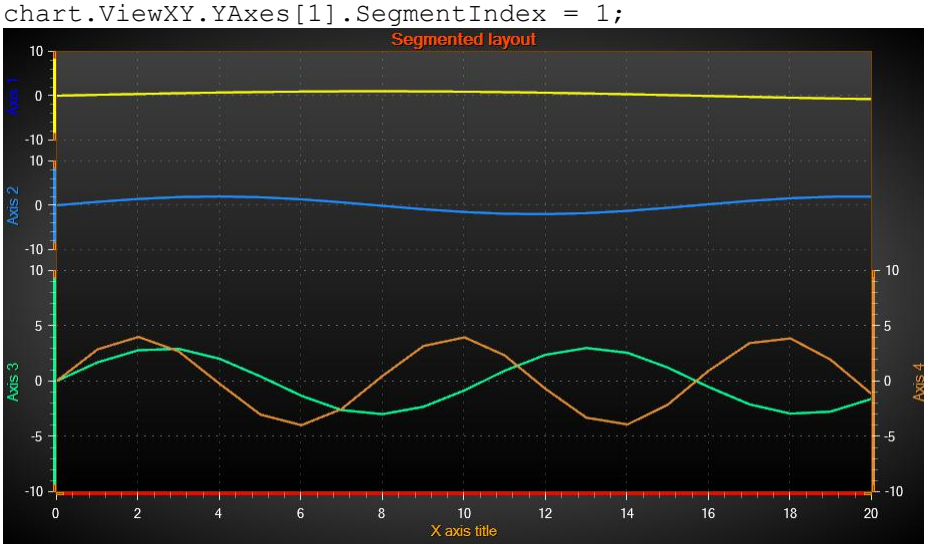
```
chart.ViewXY.AxisLayout.YAxesLayout = YAxesLayout.Segmented;
```

세그먼트는 **AxisLayout.Segments** 컬렉션에서 생성되어야 한다. 처음으로 추가된 세그먼트는 차트 아래에 놓여질 것이다. 세그먼트는 단 한 가지의 속성을 갖고 있다: **높이**. 이는 다른 세그먼트들의 비해 관계적인 크기이다. 세그먼트는 차트 크기로 재조정이 필요하기에 화면 픽셀로 정의 되어 있지 않다.

```
// 세그먼트 두개 추가, 두번째의 높이가 첫번째의 두배
chart.ViewXY.AxisLayout.Segments.Add(new YAxisSegment());
chart.ViewXY.AxisLayout.Segments.Add(new YAxisSegment());
chart.ViewXY.AxisLayout.Segments[0].Height = 1;
chart.ViewXY.AxisLayout.Segments[0].Height = 2;
```

Y 축들은 **yAxis.SegmentIndex** 속성 설정으로 세그먼트에 할당 가능하다. **SegmentIndex** 는 **AxisLayout.Segments** 컬렉션에 있는 지수다.

```
chart.ViewXY.YAxes[0].SegmentIndex = 0;
chart.ViewXY.YAxes[1].SegmentIndex = 1;
```



보기 6-18. YAxesLayout = Segmented 에 있는 Y 축들의 4 가지 예시. 첫 두 세그먼트는 Height = 1 로 설정되고 마지막 세그먼트는 Height = 2.5 로 설정 되었다. Axis1.SegmentIndex = 0, Axis2.SegmentIndex = 1, Axis3 와 Axis4.SegmentIndex = 3.

스택 또는 **세그먼트** 뷰가 선택 되었을 때 그래프 세그먼트 사이 세로 스페이스는 **ViewXY.AxisLayout.SegmentsGap** 속성을 통해 조정 가능합니다.

```
chart.ViewXY.AxisLayout.SegmentsGap = 10; // 각 세그먼트 사이 10 픽셀 거리를 둠
```

많은 양의 Y 축들이 정의 되었을 시 **AutoShrinkSegmentsGap** 속성을 활성화해 사이 거리를 자동으로 줄이세요. 함으로써 각 Y 축은 그려질 수 있는 세로 스페이스를 배정 받습니다.

```
chart.ViewXY.AxisLayout.AutoShrinkSegmentsGap = false;
```

ViewXY.GetGraphSegmentInfo() 메소드는 세그먼트 관련 유저 인터페이스 로직을 구현할 필요가 있을 때 그래프 세그먼트 태두리가 어디에 있는지 찾을 수 있다.

```
// 각 세그먼트의 위 아래 좌표 부르기
float[] topCoords = chart.ViewXY.GetGraphSegmentInfo().SegmentTops;
float[] bottomCoords = chart.ViewXY.GetGraphSegmentInfo().SegmentBottoms;
```

5.1.3 축 그리드 스트립

데모 예시: 역사적 데이터 리뷰; 줌 바 차트

축 그리드 (디비전) 간격들은 채우기로 그래프 배경 위에 표시 가능하다. **ViewXY.AxisLayout.AxisGridStrips** 를 **X** 로 설정 함으로써 스트립 설정에 X 축이 사용된다. 비슷하게 **AxisGridStrips** 을 **Y** 로 설정 함으로 써 스트립 설정에 Y 축이 사용된다. **Both** 옵션은 X 및 Y 축 둘다를 위해 스트립을 설정하고 **None** 옵션은 그리드 스트립이 사용되지 않음으로 표시된다.

```
chart.ViewXY.AxisLayout.AxisGridStrips = XYAxisGridStrips.X;
```

XGridStripAxisIndex 은 여러 축들이 사용 되었을 때 스트립을 위해 사용될 X 축을 설정한다. 동시에 하나의 X 축만 사용 가능하다.

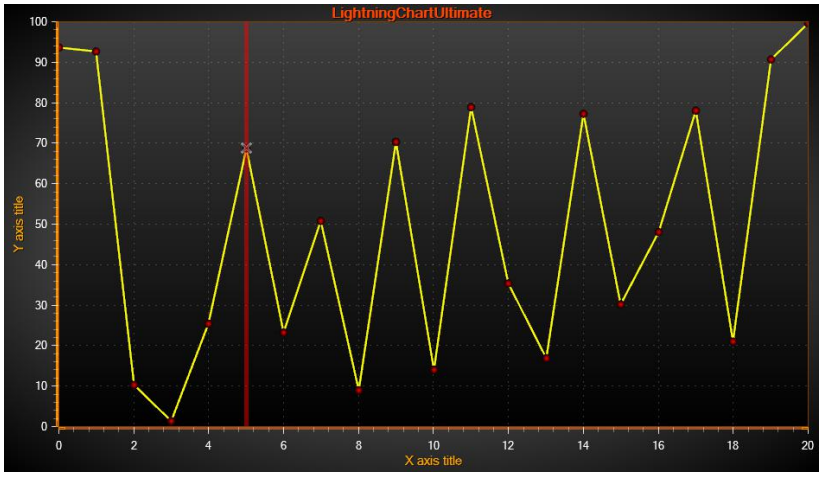
```
chart.ViewXY.AxisLayout.XGridStripAxisIndex = 0;
```

YGridStripAxisIndexLayered 는 **Layered YAxisLayout** 옵션이 사용 되었을 때 스트립을 위해 사용할 Y 축을 설정한다. **스택** 레이아웃 시 모든 Y 축들은 개인의 스트립이 있다.

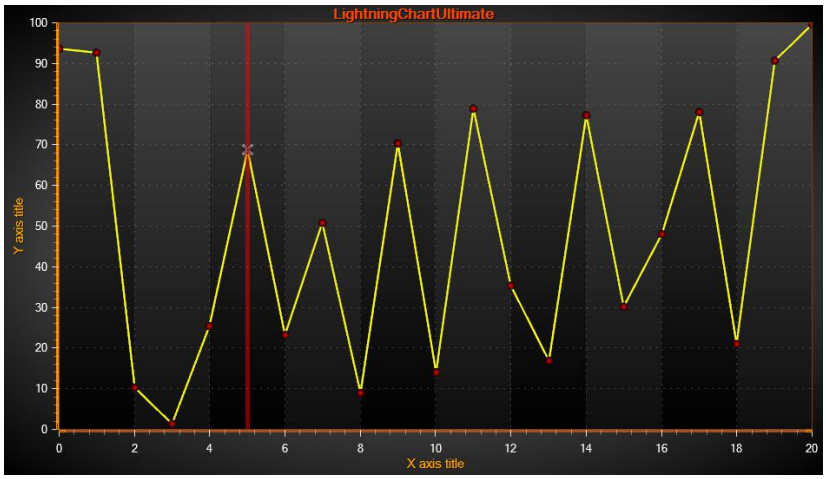
```
chart.ViewXY.AxisLayout.YGridStripAxisIndexLayered = 0;
```

스트립 색은 X 또는 Y 축 객체의 **GridStripColor** 속성에서 조정 가능하다.

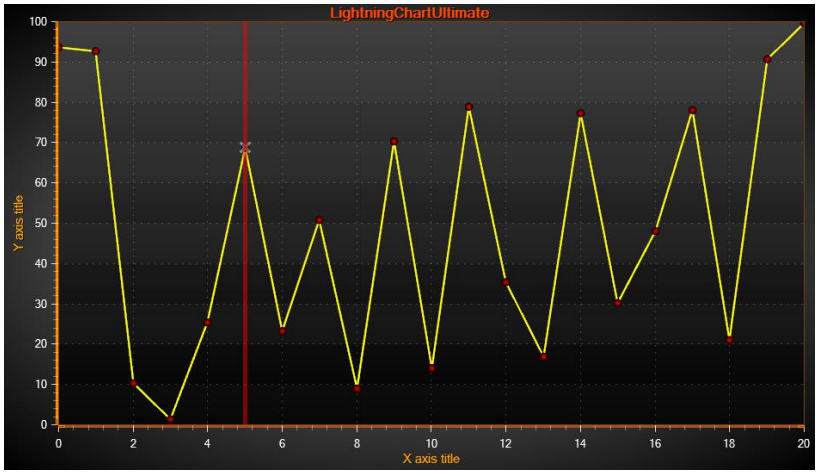
```
chart.ViewXY.YAxes[0].GridStripColor = Color.FromArgb(80, 0, 0, 100);
```



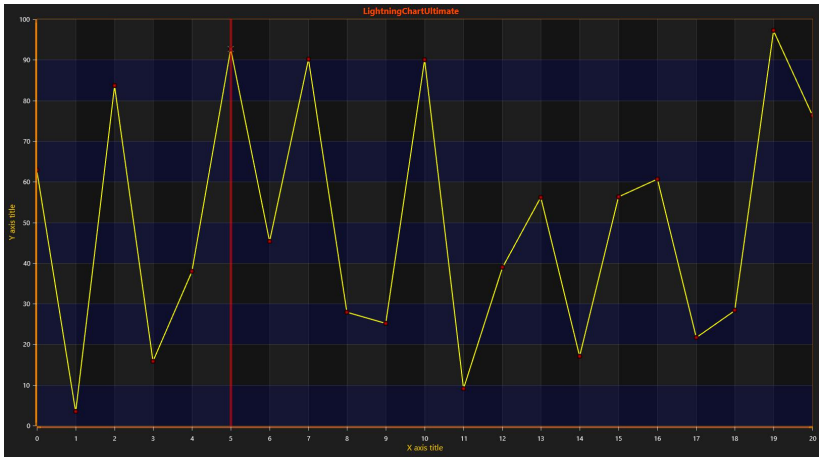
보기 6-19. AxisGridStrips = None.



보기 6- 20. AxisGridStrips =X.



보기 6- 21. AxisGridStrips = Y.



보기 6-22. AxisGridStrips = Both. Y 축에 대한 GridStripColor 은 변경 되었다.

5.1.4 기타 AxisLayout 옵션

XAxisAutoPlacement 또는 **YAxisAutoPlacement** 가 활성화 되었을 때 **AutoAdjustAxisGap** 은 두 인접한 축 지역 사이 공간을 픽셀 크기로 설정한다.

```
chart.ViewXY.AxisLayout.XAxisAutoPlacement = XAxisAutoPlacement.AllBottom;
chart.ViewXY.AxisLayout.AutoAdjustAxisGap = 10;
```

XAxisTitleAutoPlacement (또는 **YAxisTitleAutoPlacement**)을 활성화 함은 축 제목 거리는 값 라벨 길이, 정렬 옵션 및 티크 라인 기반으로 자동으로 계산된다. **XAxisTitleAutoPlacement** (또는 **YAxisTitleAutoPlacement**)가 비활성화 되었을 시 축 객체 속성의 **Title.DistanceToAxis** 가 축 선까지의 거리를 설정한다.

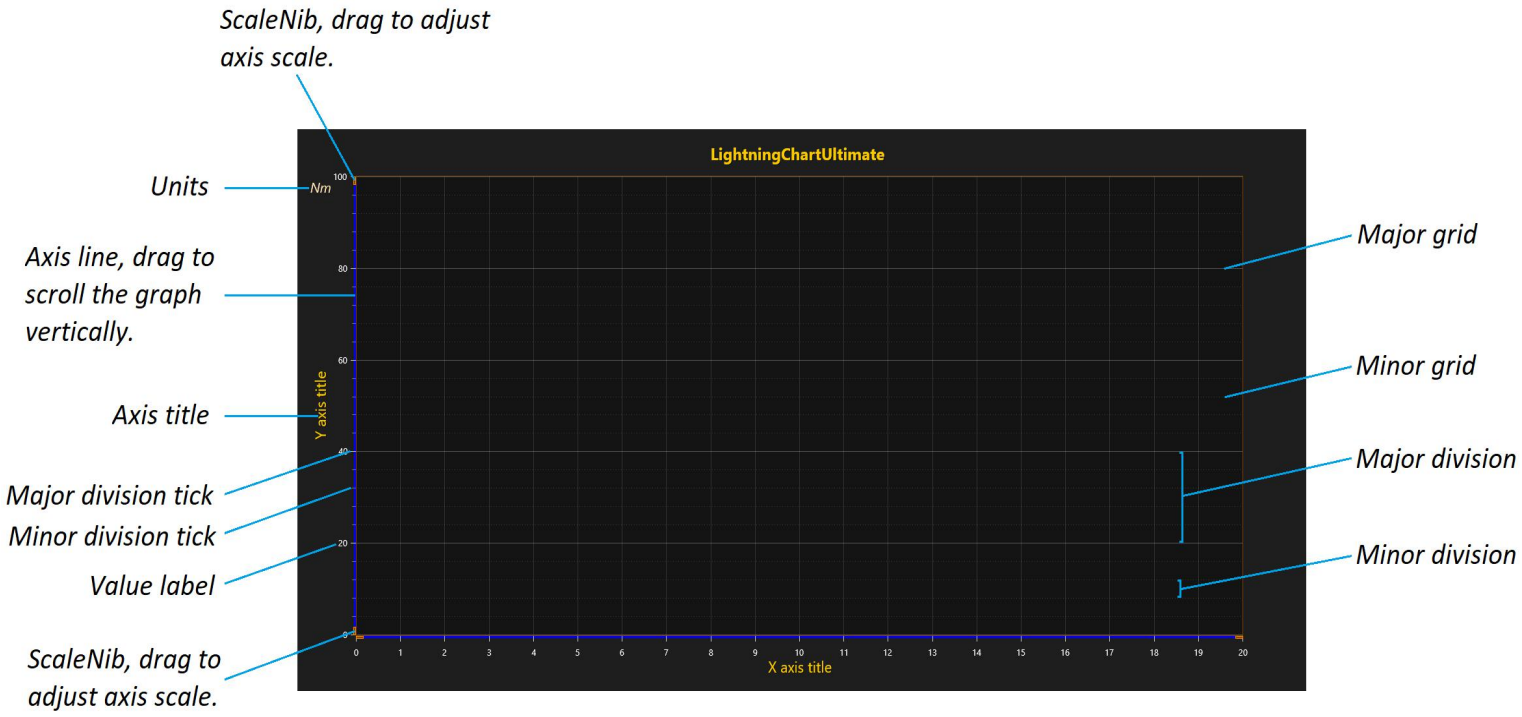
```
chart.ViewXY.AxisLayout.XAxisTitleAutoPlacement = false;
chart.ViewXY.XAxis[0].Title.DistanceToAxis = -20;
```

5.2 Y 축

무제한 수의 Y 축의 정의가 가능하다. **YAxes** 컬렉션 속성을 통해 Y 축을 추가하라.

```
// 차트에 Y축 추가
chart.ViewXY.YAxes.Add(new AxisY());

AxisY axisY = new AxisY(_chart.ViewXY);
axisY.Title.Text = "Y-axis";
chart.ViewXY.YAxes.Add(axisY);
```

5.2.1 AxisY 클라스 속성



보기 6-23. Y 축, 나눔 및 그리드

5.2.2 티크 값 라벨 서식

데모 예시: 하이-로우; 기온 그래프; 멀티 채널 커서 트래킹; 지도 루트

AutoFormatLabels 은 보이는 범위에 알맞게 자동 계산하기 위해 소수 세기, 시간 형식 표현, 또는 지수 표현을 허용한다. 수동으로 값 설정하기 위해서는 **AutoFormatLabels** 을 비활성화 해라.

```
chart.ViewXY.YAxes[0].AutoFormatLabels = false;
```

LabelsNumberFormat 은 숫자 값의 서식 설정에 사용할 수 있다.

```
// 항상 소수점 두개 사용
chart.ViewXY.YAxes[0].LabelsNumberFormat = "0.00";
```

```
// 하나의 소수점을 사용한 지수 표현
chart.ViewXY.YAxes[0].LabelsNumberFormat = "0.0E+00";
```

시간 형식을 수동으로 설정하기 위해 **LabelsTimeFormat** 속성을 사용하라. 모든 초 단위의 소수점을 지원하여 (예) ".ffffffff") 정확한 줌 뷰를 허용한다.

```
// 시간, 분, 초 및 소수 4 자리 수까지 보여주기
_chart.ViewXY.YAxes[0].LabelsTimeFormat = "HH:mm.ss.ffff";
```

5.2.3 값 종류

ValueType 속성은 축 라벨에 어느 값 종류가 사용 되는지 제어한다.

```
// 축 값 종류 변경
chart.ViewXY.YAxes[0].ValueType = AxisValueType.DateTime;
```

ValueType 에는 다음과 같은 옵션들이 있다:

Number

정수 및 소수 숫자 형식. **AutoFormatLabels** 가 비활성화 되었을 때 **LabelsNumberFormat** 가 적용된다. 기본 값.

Time

시간 표시. **AutoFormatLabels** 가 비활성화 되었을 때 **LabelsNumberFormat** 가 적용 된다.

DateTime

시간 옵션 기능이 있는 날짜 표시. **Time** 종류와 비슷하게 **AutoFormatLabels** 가 비활성화 되었을 때 **LabelsNumberFormat** 가 적용 된다

주의! 최고의 정확성을 위하여 **DateOriginYear**, **DateOriginMonth** 및 **DateOriginDay** 를 차트에 있는 날짜들 살짝 아래로 설정 하는 것을 추천합니다. **DateTimeToAxisValue** 메소드를 사용하여 .NET **DateTime** 객체에서 시리즈 데이터에 사용될 축 값을 얻으세요.

```
// Convert current time to Y value
data[0].Y = chart.ViewXY.YAxes[0].DateTimeToAxisValue(DateTime.Now);
```

MapCoordsDegrees

십진수로 지리적 지도 지표 표시.

예시: `40.446195° -79.948862°`

MapCoordsDegNESW

십진수로 지리적 지도 표시. 동서남북 표시도 있음.

예시: `40.446195N 79.948862W`

MapCoordsDegMinSecNESW

십진수, 아크 분, 아크 초, 동서남북 표시 있는 지리적 지도 표시.

예시: `40°2'13"N 9°58'2"W`

MapCoordsDegPadMinSecNESW

십진수, 아크 분, 아크 초, 동서남북 표시 있는 지리적 지도 표시. 10 미만의 아크 분 및 초들은 0 으로 패딩 됨. 이는 숫자가 정렬되어 Y 축에서 좌표 표시하기 좋은 방식이다.

예시: `40°02'13"N 9°58'02"W`

5.2.4 범위 설정

Minimum 및 **Maximum** 속성에 값을 주어 축 값의 범위를 설정하라. **Minimum** 은 **Maximum** 보다 낮아야 한다. **Minimum** > **Maximum** 또는 그 반대를 설정할 때에 정수 리미터가 다른 값 근처로 제한을 둘 것이다. 동시에 두 값을 설정하기 위해 **SetRange(…)** 메소드를 사용하라. **SetRange** 에서 **Minimum** > **Maximum** 을 넣는 것은 **Minimum** < **Maximum** 이 되게끔 자동으로 값들을 뒤집는다.

```
chart.ViewXY.YAxes[0].Minimum = 5;
chart.ViewXY.YAxes[0].SetRange(5, 10);
```

MouseScrolling 이 활성화 되었을 때 Y 축의 값 범위는 마우스롤 축을 드래그 함으로 변경이 가능하다. **MouseScaling** 속성이 활성화 되었을 때 **Minimum** 또는 **Maximum** 은 스케일 닙 부분 (축의 끝부분)을 위 아래로 드래그해서 변경 가능하다.

```
// Enabling / disabling dragging with mouse
chart.ViewXY.YAxes[0].MouseScrolling = true;
chart.ViewXY.YAxes[0].MouseScaling = true;
```

5.2.5 범위 복원

축은 **RangeRevertEnabled**, **RangeRevertMaximum** 및 **RangeRevertMinimum** 속성들을 갖고 있다. 이 속성들은 마우스 주밍이 우에서 좌로 적용 되었을 때 축 범위들을 특유 값으로 복원하는 데에 사용할 수 있다. 더 많은 정보를 위해 5.22.5 를 보세요.

5.2.6 나눗셈

MajorDiv 와 MinorDiv 속성으로 제어된 나눗셈은 차트 내 메이저 및 마이너 티크의 수를 센다. 예를 들어 5 개의 메이저 나눗셈을 설정하면 Y 축을 티크와 메이저 그리드 라인으로 같은 크기의 5 구간으로 나눈다. 기본적으로 메이저 티크가 활성화 되었고 마이너 티크가 비활성화 되었다.

```
// Enabling minor division ticks
chart.ViewXY.YAxes[0].MinorDivTickStyle.Visible = true;
```

AutoDivSpacing 속성은 자동으로 메이저 나눗셈을 계산하게 허용한다. 이는 기본 설정이다. 사용자의 편리함을 위해 스페이싱은 값 라벨의 글꼴 크기 및 **AutoDivSeparationPercent** 속성들 기반으로 계산된다. **AutoDivSeparationPercent** 는 값 라벨 사이에 공간을 남긴다. 라벨의 높이를 기반으로 적용되어 값을 높일 수록 메이저 나눗셈의 수를 줄인다. **AutoDivSpacing** 및 **AutoDivSeparationPercent** 은 마이너 나눗셈에 효과가 없다. 대신 **MinorDivCount** 속성 값 기반으로 메이저 나눗셈 사이 계산된다.

```
// 100 percent of value labels' height left between each label
chart.ViewXY.YAxes[0].AutoDivSeparationPercent = 100;
```

AutoDivSpacing 이 비활성화 되었을 시 나눗셈 스페이싱은 **MajorDiv 와 MajorDivCount** 속성들로 수동적으로 제어 가능하다. **MajorDiv** 는 크기로 스페이싱을 제어하는 반면 **MajorDivCount** 는 나눗셈 수로 제어를 한다. **KeepDivCountOnRangeChange** 속성을 이용해 축 범위가 변경될 때마다 **MajorDiv** 설정 별개로 나눗셈 수를 강제로 유지할 수 있다.

```
// Major ticks after each 20 units (0, 20, 40, 60…)
chart.ViewXY.YAxes[0].MajorDiv = 20;

// Show exactly five major divisions
chart.ViewXY.YAxes[0].MajorDivCount = 5;
// Keep the division count the same even if the axis range is changed
chart.ViewXY.YAxes[0].KeepDivCountOnRangeChange = true;
```

MajorDivTickStyle 속성을 통해 메이저 나눗셈 티크 스타일 설정 가능하다. **MajorDivTickStyle.Alignment** 속성을 이용해 티크 및 라벨 정위를 수정하라. **MinorDivTickStyle** 속성을 이용해 마이너 나눗셈 속성들 변경 가능하다.

```
// Changing alignment and tick length of the major division ticks
chart.ViewXY.YAxes[0].MajorDivTickStyle.Alignment = Alignment.Far;
chart.ViewXY.YAxes[0].MajorDivTickStyle.LineLength = 20;
```

5.2.7 그리드

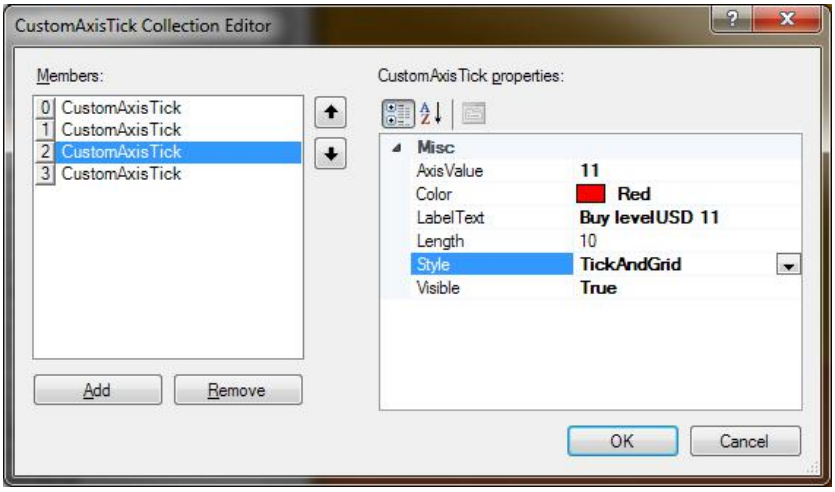
수평 그리드 라인들은 나눗셈 티크의 수직 위치에 그려져 있다. 메이저 나눗셈 티크는 메이저 그리드를 위해, 마이너 나눗셈 티크는 마이너 그리드를 위해. **MajorGrid** 및 **MinorGrid** 속성들은 그리드의 생김세를 수정하는데에 사용 가능하다.

```
// Modifying the grid styles
chart.ViewXY.YAxes[0].MajorGrid.Color = Color.FromArgb(100, 200, 200, 200);
chart.ViewXY.YAxes[0].MinorGrid.Pattern = LinePattern.Dash;
```

5.2.8 커스텀 틱

데모 예시: 커스텀 축 틱; 날짜 축 및 커스텀 틱

축 틱 위치 및 라벨 문구는 커스텀 틱을 이용해 수동적으로 설정 가능하다. **CustomTicksEnabled** 을 참으로 설정하고 **CustomTicks** 리스트 속성으로 틱들의 위치를 정의해 주세요.

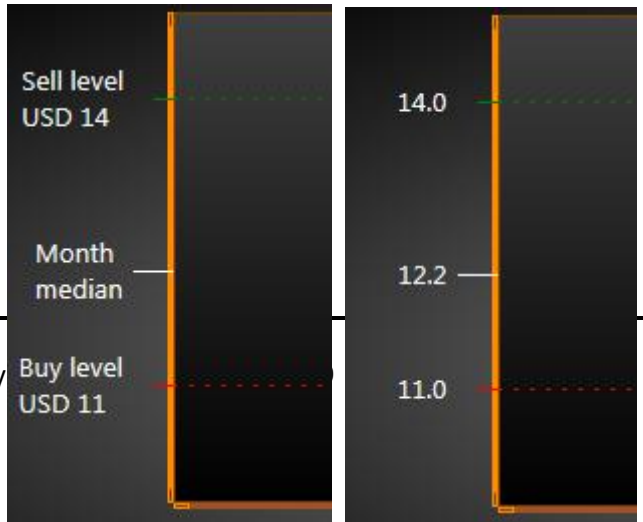


보기 6-24. 커스텀 틱 속성

커스텀 틱은 틱 또는 그리드 또는 둘다를 포함할 수 있다. **Style** 을 이용해 Tick, Grid, 또는 TickAndGrid 사이 선택하라. 틱 또는 그리드의 색은 **Color** 속성을 통해 변경 가능하다. 틱 길이를 **Length** 속성에서 설정하라. 그리드 라인 패턴은 **MajorGrid.Pattern** 및 **PatternScale** 축 속성 설정을 따른다.

CustomAxisTick 은 **AxisValue** 및 **LabelText** 속성들을 통해 위치 및 해당 라벨 문구가 정의된다. 커스텀 틱 사용 시 **AutoFormatLabels** 를 비활성화 해서 커스텀 라벨 텍스트를 보여라. 코드로 신규 커스텀 틱 설정 후 **InvalidateCustomTicks()** 를 불러라.

```
// Adding a custom tick with a green tick and grid line
_chart.ViewXY.YAxes[0].CustomTicks.Add(new CustomAxisTick(_chart.ViewXY.YAxes[0],
14, "Sell level\nUSD 14", 10, true, Colors.Green, CustomTickStyle.TickAndGrid));

// White tick with no grid line
_chart.ViewXY.YAxes[0].CustomTicks.Add(new CustomAxisTick(_chart.ViewXY.YAxes[0],
12.2, "Month\nmedian", 20, true, Colors.White, CustomTickStyle.Tick));

// Red tick and grid line
_chart.ViewXY.YAxes[0].CustomTicks.Add(new CustomAxisTick(_chart.ViewXY.YAxes[0],
11, "Buy level\nUSD 11", 10, true, Colors.Red, CustomTickStyle.TickAndGrid));

// Allow showing the custom tick strings
_chart.ViewXY.YAxes[0].CustomTicksEnabled = true;
_chart.ViewXY.YAxes[0].AutoFormatLabels = false;
_chart.ViewXY.YAxes[0].MajorGrid.Pattern = LinePattern.Dot;
_chart.ViewXY.YAxes[0].InvalidateCustomTicks();
```



보기 6-25. Y 축에 커스텀 틱. 좌측에는 axis.AutoFormatLabels = false. 우측에는 AutoFormatLabels = True.

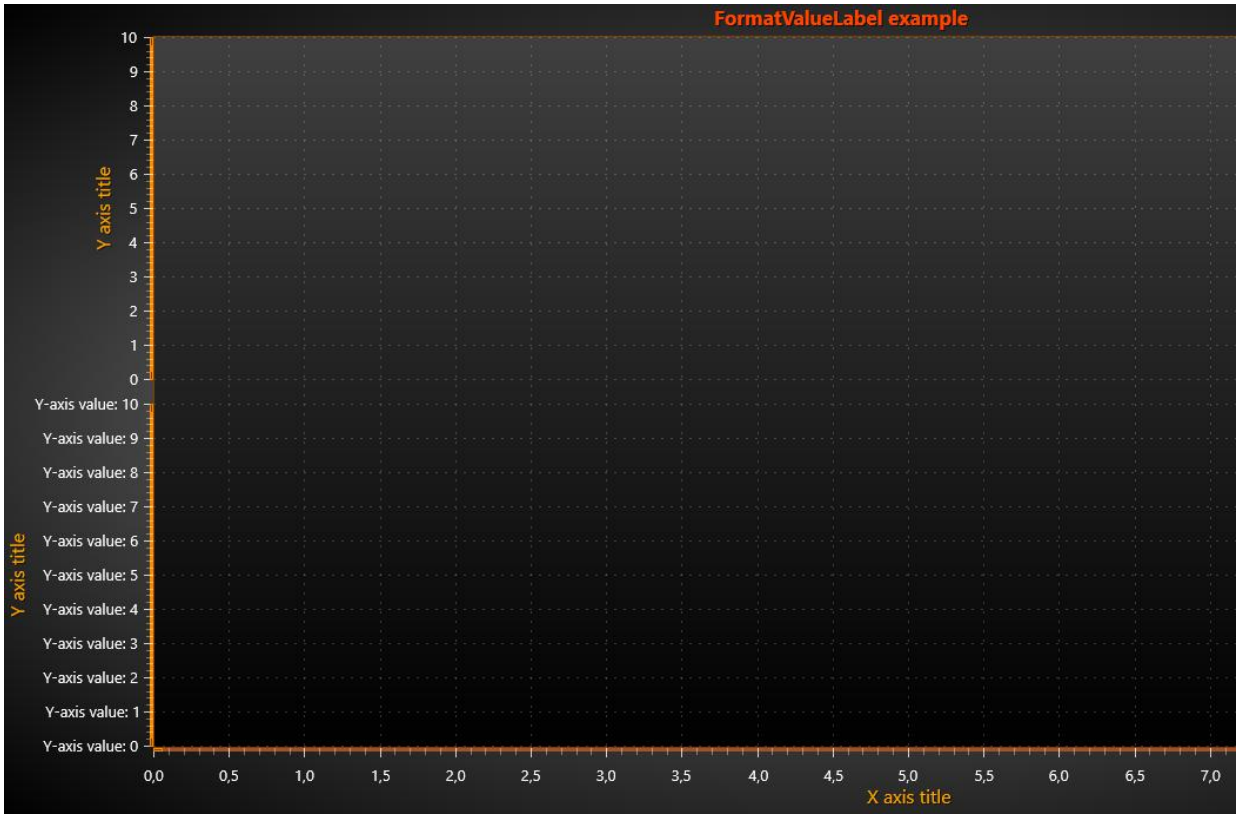
마이너 틱 또는 그리드는 **CustomAxisTicksEnabled** 이 참일 시 보여지지 않는다. 임의의 마이너 틱 또는 그리드 설정하기 위해서는 다른 색 또는 선 길이로 **CustomTicks** 컬렉션에 **CustomAxisTicks** 를 추가하라.

5.2.9 이벤트 기반 축 값 서식

데모 예시: 축 범위 수정, 값 라벨; 비즈니스 대시보드; 강도 지속 레이어, 시그널

CustomAxisTicks 이외 축 값 라벨은 **FormatValueLabel** 이벤트를 통해 서식 가능하다. 이는 해당 축의 각 값 라벨을 돌려진 스트링 값에 따라 변경한다. 이 이벤트는 **e.Axis** 및 **e.Value** 속성들을 갖고 있으며 이 속성들은 축 객체 및 라벨 값이 변경 되지 않고서는 접속이 불가하다. **CustomAxisTicks** 와 다르게 **FormatValueLabel** 는 축을 위해 나눗셈 설정을 따르기에 (제 6.2.6 장을 보세요) 라벨들의 위치 변경에 사용할 수 없다.

```
// Subscribing to FormatValueLabel -event for the Y-axis
_chart.ViewXY.YAxes[0].FormatValueLabel += Chart_FormatValueLabel;

// Modifying the value labels inside the event
private string Chart_FormatValueLabel(object sender, FormatValueLabelEventArgs e)
{
      return "Y-axis value: " + e.Value.ToString();
}
```



보기 6-26. 아래 Y 축으로 FormatAxisValues 사용. 값들은 "Y-axis value: " + 현 값으로 보여진다.

FormatValueLabel 은 Y 축과 X 축 둘다와 함께 사용 가능하며 View3D 에서는 모든 축과 함께 사용 가능하다.

5.2.10 반전된 X 와 Y 축

최소 갑이 최대 값보다 위/나중으로 보이게 X 와 Y 축들은 반전이 가능하다. 이는 예를 들어 Y 축에 할당된 시리즈 데이터를 시각적으로 부정 극성하는 데에 유용하다.

```
chart.ViewXY.YAxes[0].Reversed = true;
```

5.2.11 로그 축

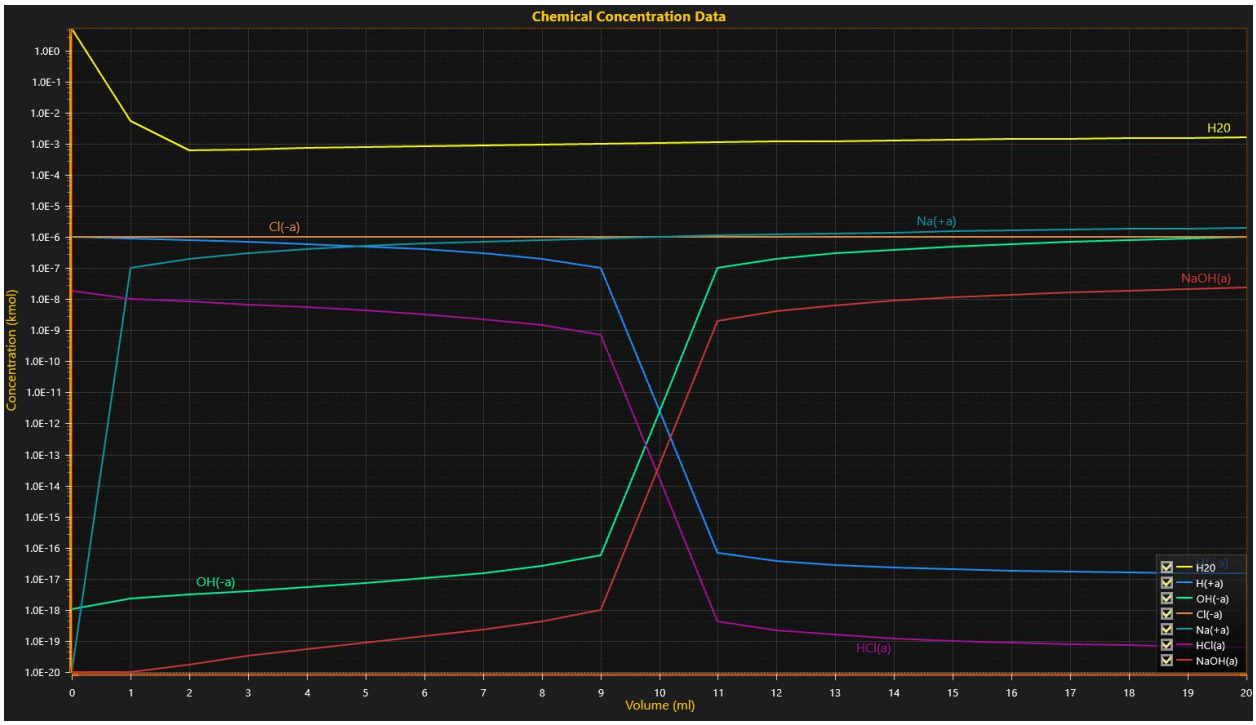
데모 예시: 로그 축: 최소 로그 값; 로그 축 핏, 0 무시

로그 형식 표기를 위해 **ScaleType** 를 **Logarithmic** 으로 설정하라. 로그 베이스 값을 **LogBase** 속성으로 설정하라. 차트는 0 과 1 사이의 로그 값을 보여줄 수 있다. **LogZeroClamp** 를 사용해 축의 최소 값을 설정 하라. 전형적인 로그 축의 최소 값을 사용하기 위해서는 1 로 설정하라. 0 미만 값을 사용하기 위해서는 1.0E-20 과 같은

사용 데이터에 적합한 작은 양수 값을 설정하라. 틱 라벨을 위한 특별 서식을 사용하기 위해서는 **LogLabelsType** 설정하라.

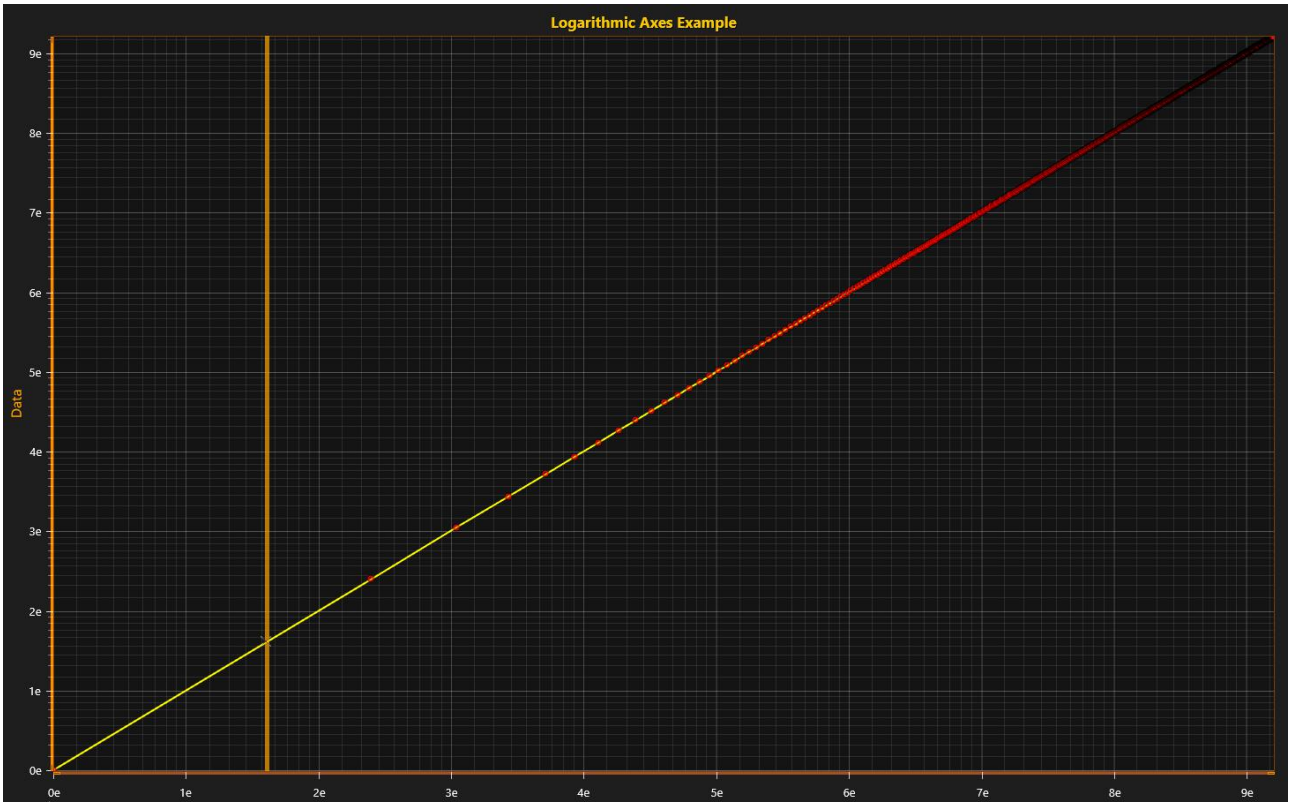
```
// Setting a logarithmic axis
chart.ViewXY.YAxes[0].ScaleType = ScaleType.Logarithmic;
chart.ViewXY.YAxes[0].LogBase = 10;
chart.ViewXY.YAxes[0].LogZeroClamp = 1;
chart.ViewXY.YAxes[0].LogLabelsType = LogLabelsType.Log10Exponential;
```

5.2.11.1 10 베이스를 위한 지수 표기



보기 6-27. 0 주위의 값이 있는 로그 Y 축. LogZeroClamp 는 1.0E-20 로 설정 되었다. LogBase 는 10 으로 설정 되었고 LogLabelsType 는 1.0E 값 표현을제대로 표시하기 위해 Log10Exponential 로 설정되었다.

5.2.11.2 자연 로그



보기 6-28. 자연 로그 뷰. LogBase 는 Math.E 로 설정 되었다. LogLabels Type 는 LogE_MultiplesOfNeper 로 설정 되었다.

5.2.12 축 값 및 화면 좌표 사이 변경

축에는 축 값 (데이터 포인트 값)을 화면 좌표로 변경과 그 반대로 변경할 수 있는 메소드가 있다. **ValueToCoord** 메소드를 사용해 축 값을 화면 좌표로, **CoordToValue** 메소드를 화면 좌표에서 축 값으로 변경하는 데 사용하라. 장치 독립적 필셀 (DIP)보다 픽셀을 선호하면 **UseDIP = False** 로 설정 하라.

```
float screenCoordinate = _chart.ViewXY.XAxes[0].ValueToCoord(axisValue);
```

ValueToCoord 및 **CoordToValue** 메소드들은 차트의 최종 크기 확정 이후 사용 가능하다. 예를 들어 **chart.AfterRendering** 이벤트에 구독하여 차트가 완전히 렌더링 된것을 확인하라.

한번에 여러 값 또는 좌표를 변경하기 위해서는 **ValuesToCoords** 및 **CoordsToValues** 메소드들을 사요하라. 이들은 축 값을 더블 배열로 돌려주며 (X 축 **CoordToValue** 는 정수 배열) 화면 좌표는 플로트 배열로 돌려준다.

```
chart.ViewXY.YAxes[0].CoordsToValues(coordArray, out doubleValueArray, false);
```

5.2.13 MiniScale

MiniScale 은 X 와 Y 축의 작은 대용품이다. 이런 스케일 표현 방식이 데이터 규모 개요를 위해 또는 실제 축들을 구현할 자리가 없을 때에 사용이 된다. **MiniScale** 은 **Visible** 속성으로 활성화 가능하다. **MiniScale** 은 Y 축 클라스의 하위 속성이다. X 치수는 항상 첫 X 축에 묶여있다 (**XAxes[0]**). X 와 Y 축들의 **Units.Text** 속성을 변경하여 보이는 단위를 설정하라. **MiniScale** 은 로그 축과 함께 사용할 수 없다.

```
// Configuring a MiniScale
chart.ViewXY.YAxes[0].MiniScale.Visible = true;
chart.ViewXY.YAxes[0].MiniScale.VerticalAlign = AlignmentVertical.Bottom;
chart.ViewXY.YAxes[0].MiniScale.Offset.SetValues(-10, -30);
chart.ViewXY.YAxes[0].MiniScale.PreferredSize = new Size(30, 30);
chart.ViewXY.XAxes[0].Units.Text = "s";
chart.ViewXY.YAxes[0].Units.Text = "µV";
```

보기 6-29. 그래프 우측 하단에 보이는 MiniScale

5.3 X 축

X 축 디비션 및 그리드 설정은 Y 축의 설정과 같다. 그럼으로 이전 장에 설명된 모든 속성 및 특징은 X 축에도 적용이 된다. 하지만 X 축에는 Y 축에 없는 여러 실시간 스크롤 관련 속성이 있다.

5.3.1 실시간 모니터링 스크롤링

데모 예시: 십억 포인트; 온도 그래프; 스레드 페드 멀티 채널 데이터

실시간 모니터링 솔루션을 만들 때에 X 축은 현재 모니터링 위치를 보여줄 수 있게 제대로 스크롤링이 되있어야 한다. 이는 대체적으로 최신 시그널 포인트의 타임 스탬프이다. 새로운 시그널 포인트가 시리즈로 설정된 이후 **ScrollPosition** 속성을 제일 최신 타임 스탬프로 설정하라.

```
// Set real-time monitoring scroll position to the latest X value
chart.ViewXY.XAxes[0].ScrollPosition = latestDataPoint.X;
```

라이트닝차트는 **ScrollMode** 속성을 통해 선택 가능한 여러 스크롤링 모드가 있다.

```
chart.ViewXY.XAxis[0].ScrollMode = XAxisScrollMode.Scrolling;
```

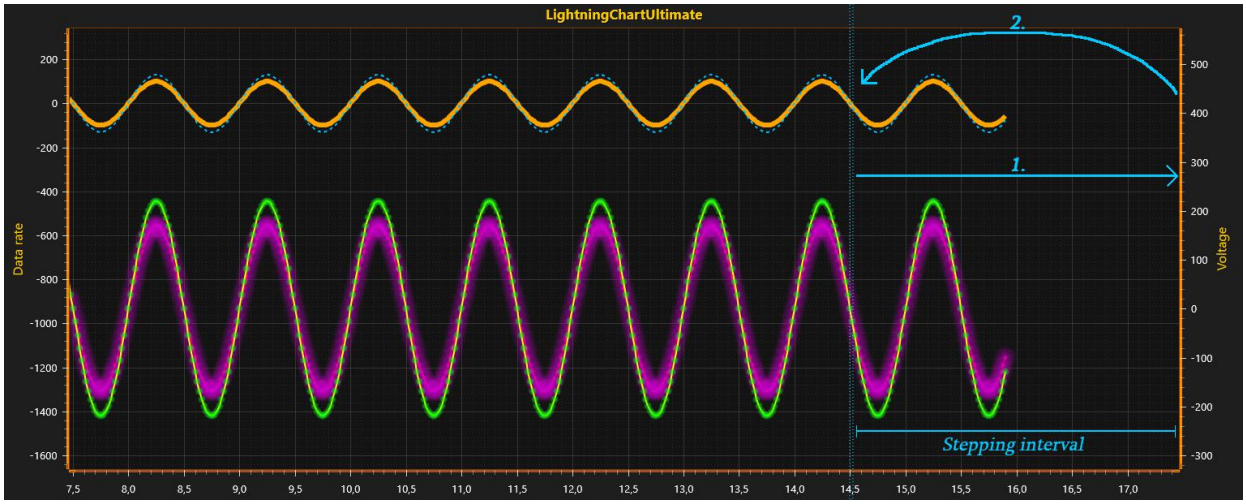
5.3.1.1 없 음

기본 설정. **ScrollPosition** 을 **None** 으로 설정하면 아무런 스크롤링이 적용되지 않는다. 실시간 모니터링을 사용하지 않을 때에 주로 선택되는 옵션이다.

5.3.1.2 스 테 핑

수집된 데이터가 x 축의 끝에 도달했을 때 축과 모든 시리즈 데이터는 왼쪽으로 스테핑 간격 만큼 움직인다. 이 움직임은 x 축의 끝을 도달했을 때마다 실행된다. **Steppinginterval** 속성은 값 범위로 정의된다.

```
chart.ViewXY.XAxes[0].SteppingInterval = 3;
```

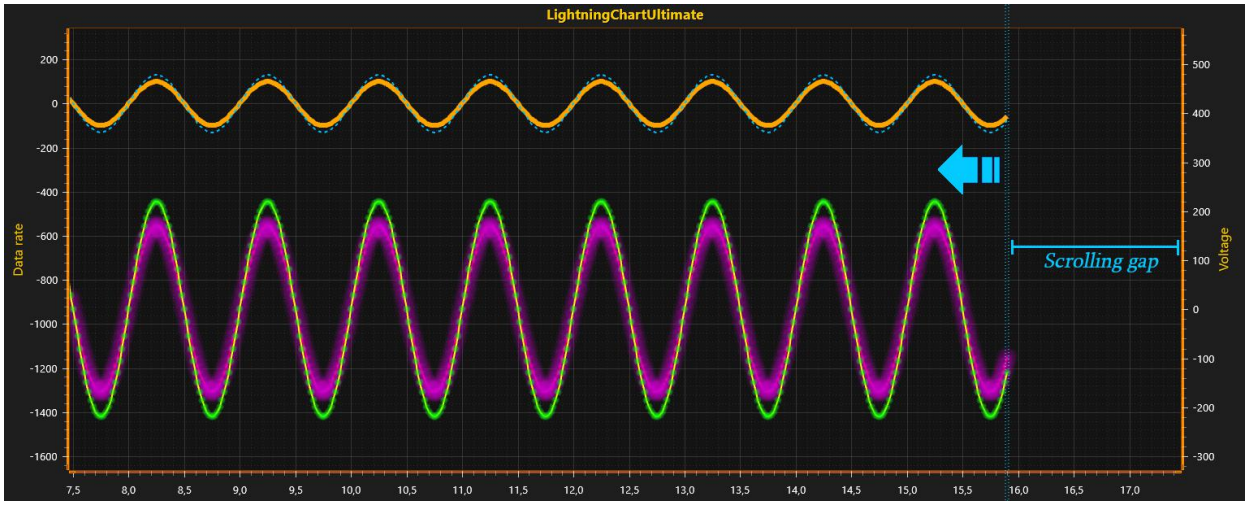


보기 6-30. X 축 스크롤 모드: 스테핑

5.3.1.3 스 크 롤 링

x 축은 스크롤링 간격에 도달했을 때까지 고정이 유지 되었다. 도달한 이후 x 축과 이의 모든 시리즈는 계속 되어 왼쪽으로 움직여 진다. 스크롤링이 스크롤 위치가 x 축의 끝에 도달한 이후 실행될 시에 **ScrollingGap** 을 0 으로 설정하라. **ScrollingGap** 속성은 그래프 넓이의 퍼센트로 정의 된다.

```
chart.ViewXY.XAxes[0].ScrollingGap = 15;
```



보기 6-31. X 축 스크롤 모드: 스크롤링

스크롤링 중 웨이브 형 안정성

라이트닝차트는 **series.AddPoints**(), **AddValues()** 또는 **AddSamples()** 메소드를 사용할 때에 실시간 시그널의 증분 렌더링 데이터 구성을 지원한다. 이는 렌더링 데이터는 신규 데이터와 이미 존재하는 렌더링 데이터로만 계산된다는 것이다.

PointLineSeries, **SampleDataSeries**, **AreaSeries** 및 **HighLowSeries** 는 웨이브 형 품질을 유지하며 스크롤 시리즈의 시각적 안정성에 효과 있는 **ScrollMode** = **Scrolling** 을 위한 특별 속성이 있다. 이 속성은 **ScrollingStabilizing** 이다.

```
chart.ViewXY.PointLineSeries[0].ScrollingStabilizing = true;
```

ScrollingStabilizing 이 활성화 되면 플로트 포인트 좌표가 가장 가까운 정수 좌표로 반올림 및 반내림이 된다. 이는 시각적으로 안정하고 변동하지 않는 웨이브 형을 만든다. 대부분의 경우에는 이것이 최선의 방법이다. 하지만 좌표를 반올림/반내림 함으로 페이즈 정보를 살작 왜곡할 수 있다.

ScrollingStabilizing 이 비활성화 되었을 때에 데이러 렌더링은 GPU 가 픽셀 좌표를 결정할 대에 살작 변동하는 웨이브 형으로 보일 수 있는 플로트 포인트 좌표를 사용한다. 이는 픽셀마다 위 아래로 변동하는 사인 데이터를 보여줄 때 특히 더욱 좋은 질의 그래프를 보여줄 수 있다.

증분 렌더링 데이터 구성을 사용하기 위해서는 다음과 같은 신규 포인트를 추가하라

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

시리즈의 **InvalidateData()** 콜을 함으로써 렌더링 데이터를 전체 새로고침 할 수 있다.

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

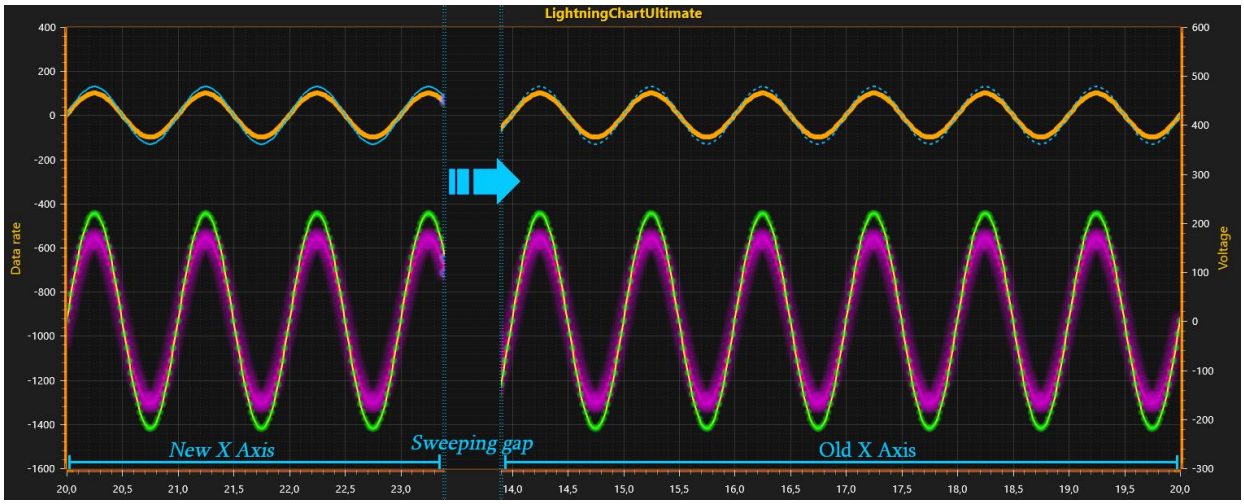
	성능	안정성	단계
series.AddPoints(), ScrollStabilizing 비 활성화	완벽	손상	좋음
series.AddPoints(), ScrollStabilizing 활 성화	완벽	최고	살짝 손상
series.AddPoints(), InvalidateData()	손상	손상	완벽

표 5-1. 스크롤링 도중 웨이브 형 안정성

5.3.1.4 스 위 핑

스위핑 모드는 아마 실시간 모니터링 보기 중 제일 사용자 친화적일 것이다. 스위핑은 두 x 축들을 사용한다. 첫 축은 스위핑 간격이 나타난 이후 풀로 수집된다. 둘째 축은 이후 첫째 축 위로 스위핑이 된다. 두 x 축은 자기만의 값 라벨을 표시한다. **SweepingGap** 속정은 그래프 넓이의 퍼센트로 정의된다.

```
chart.ViewXY.XAxes[0].SweepingGap = 5;
```

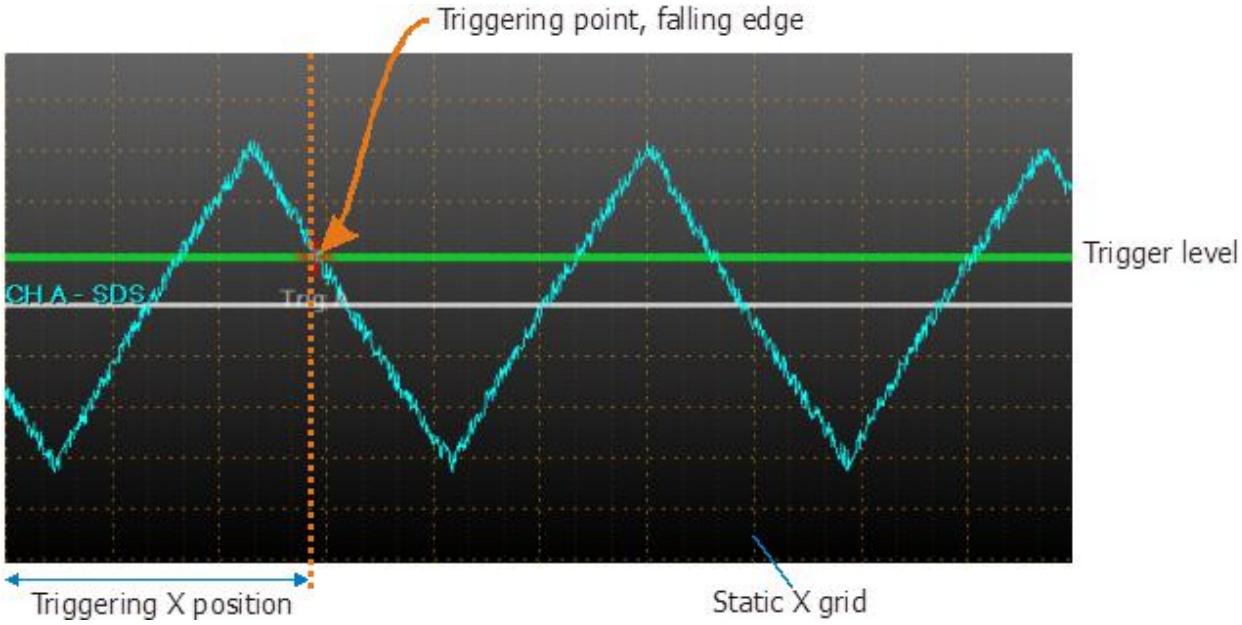


보기 6-32. X 축 스크롤 모드: 스위핑

5.3.1.5 트 리 거

X 축의 위치는 트리거 레벨을 초과 또는 미달하는 시리즈 값으로 결정된다. **Triggering** 속성을 이용해 트리거 옵션을 설정하라. 트리거는 **Triggering.TriggeringActive** 속성을 활성화 해서 설정 가능하다.

한 시리즈는 트리거 시리즈로 설정 되어야 한다. 수락된 트리거 시리즈 종류는 **PointLineSeries** 와 **SampleDataSeries** 이다. **Triggering.TriggerLevel** 로 Y 레벨 트리거를 설정하라. **Triggering.TriggeringXPosition** 을 사용해 트리거 포인트가 그래프 넓이의 퍼센트로 수평으로 그려질 레벨을 정하라.



보기 6-33. X 축 스크롤 모드: 정적 X 그리드로 트리거.

X 축 트리거 스크롤 위치를 사용 할 때에 들어오는 시리즈 데이터 기반으로 이리 저리 왔다 갔다 하기에 기본 X 축의 값 및 그리드로 보여주는 것이 적합하지 않다.

- 방법 1: 정적 X 그리드를 사용하라. 정규 X 축 객체를 **XAxis.Visible = false** (또는 **LabelsVisible** = false, **MajorGrid.Visible** = false 및 **MinorGrid.Visible** = false) 로 설정하여 숨겨라. 이후 **Triggering.StaticMajorXGridOptions** 및 **Triggering.StaticMinorXGridOptions** 설정으로 정적 X 그리드를 보여주라.
- 방법 2: 적합한 스케일의 다른 X 축을 생성하고 VeiwXY 컬렉션에 설정하라. 시리즈의 두번째 X 축을 할당하지 마라.

"200ms/div" 등의 범위의 스케일 표시를 위해 Y 축 **MiniScale** 사용 또는 **Annotation** 객체를 정의하세요 (제 6.20 장을 보세요).

5.3.2 스케일 브레이크

데모 예시: 스케일 브레이크; 이전 클로스로 스탁 코스

버전 8 부터 X 축들은 **ScaleBreaks** 를 지원한다. **ScaleBreaks** 는 비활성 거래 시간/날짜 또는 기계 생산 중단 시간 등의 특별 X 범위를 제외하는 데에 사용된다. 축 및 라벨 포함 특별 X 축에 지정된 모든 시리즈는 클립 된다.

ScaleBreaks 사용에 제한이 있다: **ScrollMode** 는 '**None**'으로 설정되어야 하고 **ScaleType** 는 '**Linear**'로 설정되어야 한다.

X 축의 **ScaleBreaks** 컬렉션에 **ScaleBreak** 객체를 삽입하라.

Figure 6- 34. ScaleBreak properties.

Begin 과 **End** 로 브레이크의 범위를 지정하라. DateTimes 가 아닌 축 값을 받는다. DateTimes 를 사용하고 있다면 axis.**DateTimeToAxisValue** 메소드를 사용해 변경하라.

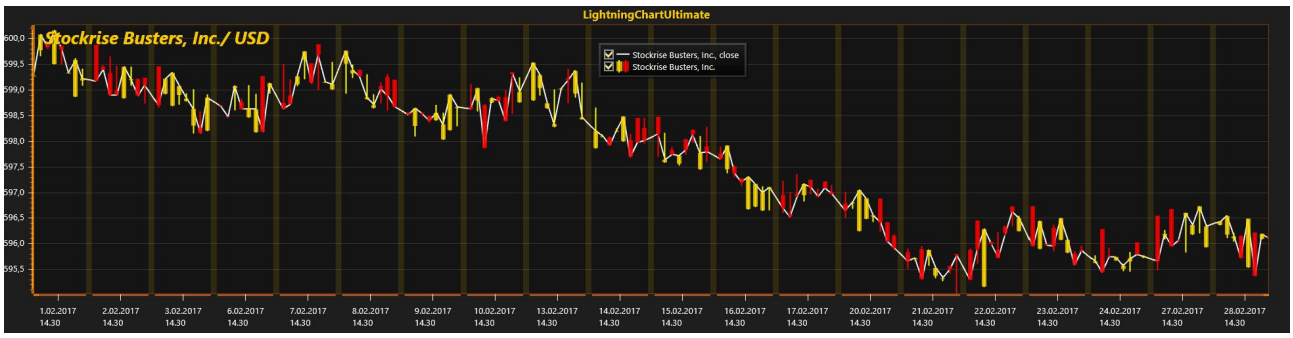
간격 넓이는 **Gap** 으로 수정 가능하다. 간격이 보이지 않아야 하면 0 의 값을 입력 가능하다. 간격 생김세는 **Style** 로 구성 가능하다.

- **Style** = '**Fill**'로 Fill 속성을 수정하라.
- **Style** = '**DiagonalLineUp**' 또는 '**DiagonalLineDown**'으로 **DiagonalLineSpacing** 및 **LineStyle** 속성들의 생김세를 구성하라.

Enabled = **False** 로 설정함으로 브레이크는 효과가 없다.

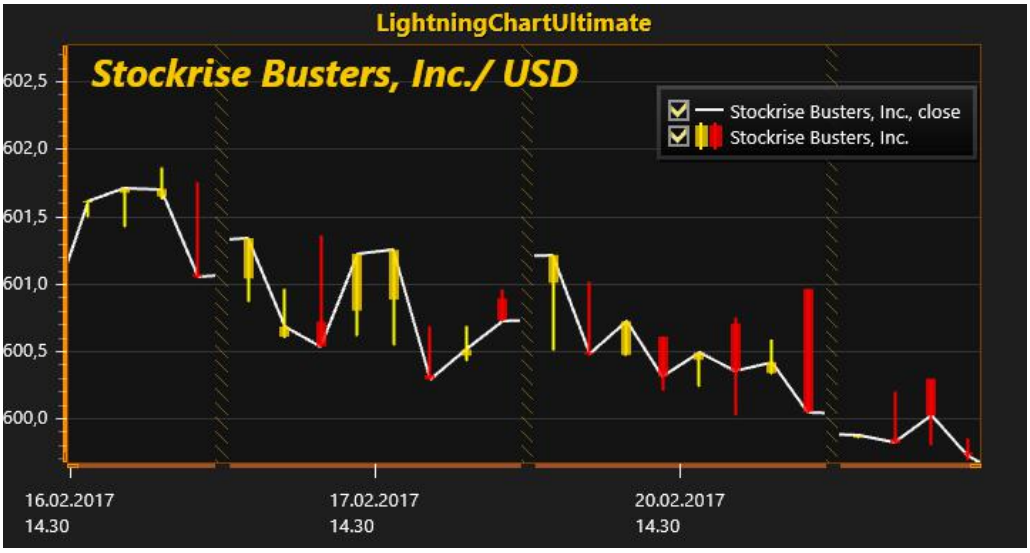
PointLineSeries, **AreaSeries** 및 **HighLowSeries** 는 **ContinuousOverScaleBreak** 속성을 갖고 있다. 활성화하는 것은 연결 선이 간격 위로 렌더링 된다.

보기 6-35. 원본 트레이딩 데이터, 월-금, 10AM – 6PM. ScaleBreaks 적용 되지 않음. 대부분의 시간 범위는 증권 거래소가 닫혀서 필수 정보를 보기 더 어렵게 되어 데이터가 없다. PointLineSeries 가 Close-to-Close 값들 사이 왔다 갔다 한다.



보기 6-36. ScaleBreaks 가 적용되어 비활성화 거래 시간을 제외하고 보여준다. 필수 정보를 보여주기 위해 화면에 더 많은 공간이 확보 되었다. Style = Fill, Gap = 10. PointLineSeries 가 Close-to-Close 값들 사이 왔다 갔다 한다. ContinuousOverScaleBreak = True.

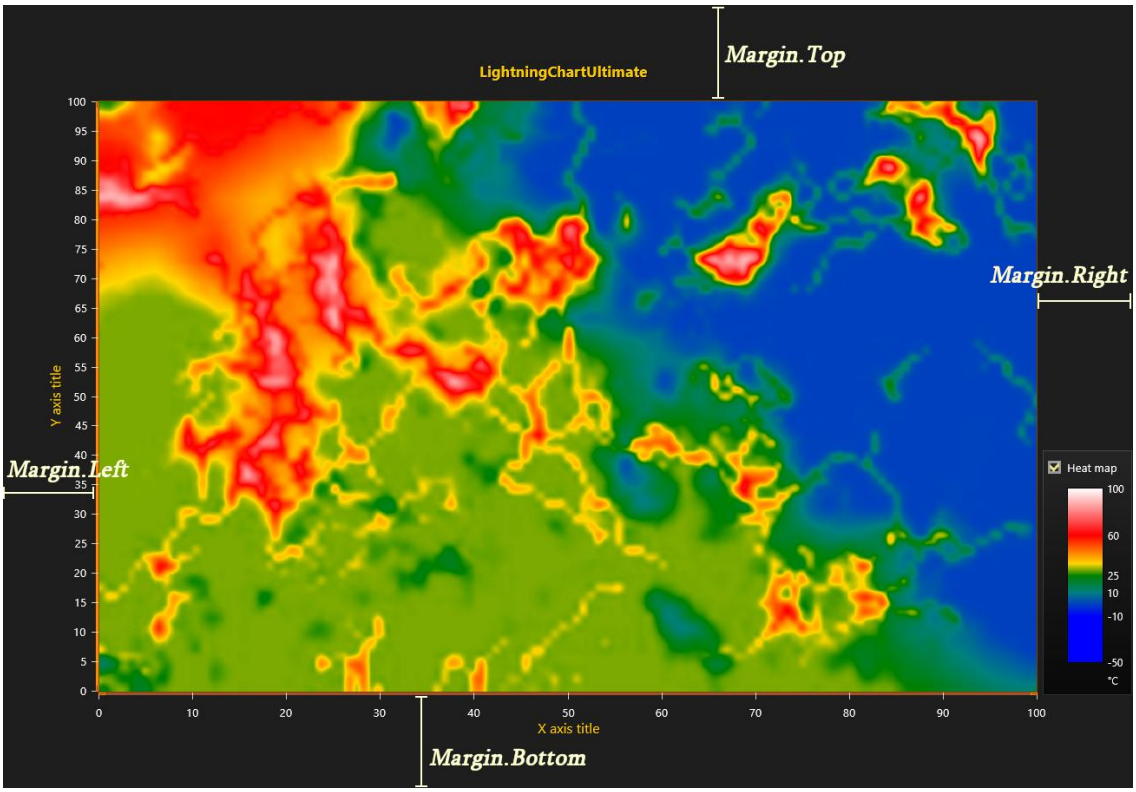
보기 6-37. 거래 비활성화 시간 중 ScaleBreaks 적용. Style = DiagonalLinesUp, Gap = 20. PointLineSeries.ContinuousOverScaleBreak = True.



보기 6-38. PointLineSeries.ContinuousOverScaleBreak = False. 간격 위로 포인트끼리 연결 되지 않음. 대신 스케일 브레이크가 정의 되지 않은 것 처럼 원래 방향대로 계속 됨.

5.4 마진

마진은 그래프 주변 공백 공간이다. 주석, 레전드 상자와 차트 제목을 제외한 뷰의 모든 내용은 마진 이내에 들어간다.



보기 6-39. 그래프 주변을 둘러싸고 있는 마진. 내용이 마진 지역 이내에 들어감. 차트 내용 및 레전드 상자는 마진 위에 위치 설정 가능하다.

AutoAdjustMargins 가 활성화 되면 그래프 크기는 모든 축 및 차트 제목을 위한 공간을 확보하기 위해 수정된다. **비활성화** 되었을 때 **ViewXY.Margins** 속성이 적용되어 마진 수동 설정을 가능하게 만든다.

기본적으로 커스터마이징 가능한 테두리 직사각형, Border,가 마진 위치로 그래프 주변으로 그려진다. 이 기능은 **Border.Visible = False** 로 설정하여 끌 수 있다. Border 의 색은 **Color** 속성으로 변경 가능하다. 또한 **RenderBehindSeries** 를 **True** 로 설정하여 **Border** 를 시리즈 위에서 렌더링 가능하다. 런타임 중 픽셀로 된 마진 직사각형을 **ViewXY.GetMarginsRect** 메소들를 불러 얻을 수 있다. 이는 자동 및 수동 마진 둘다에 적용 된다. 화면 좌표 기반 컴퓨팅 또는 객체 놓기를 해야할 시 유용하다.

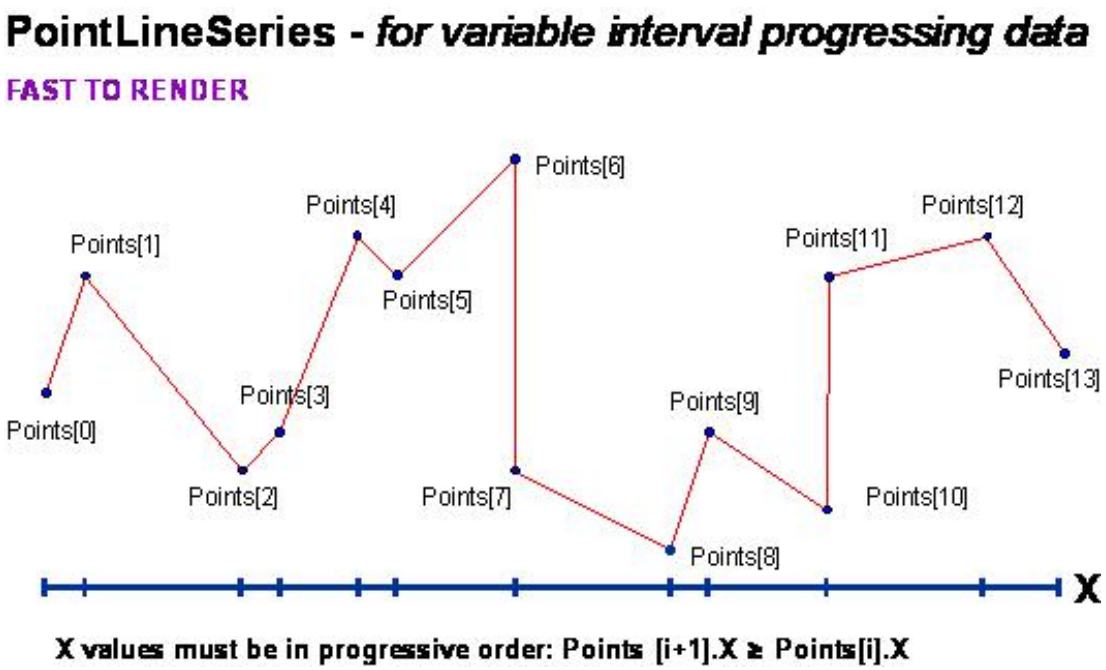
마진 직사각형이 예를 들어 크기 조절 되어 변경 되었을 때에 **ViewXY.MarginsChanged** 를 트리거 하게금 설정 가능하다.

5.5 ViewXY 시리즈, 일반

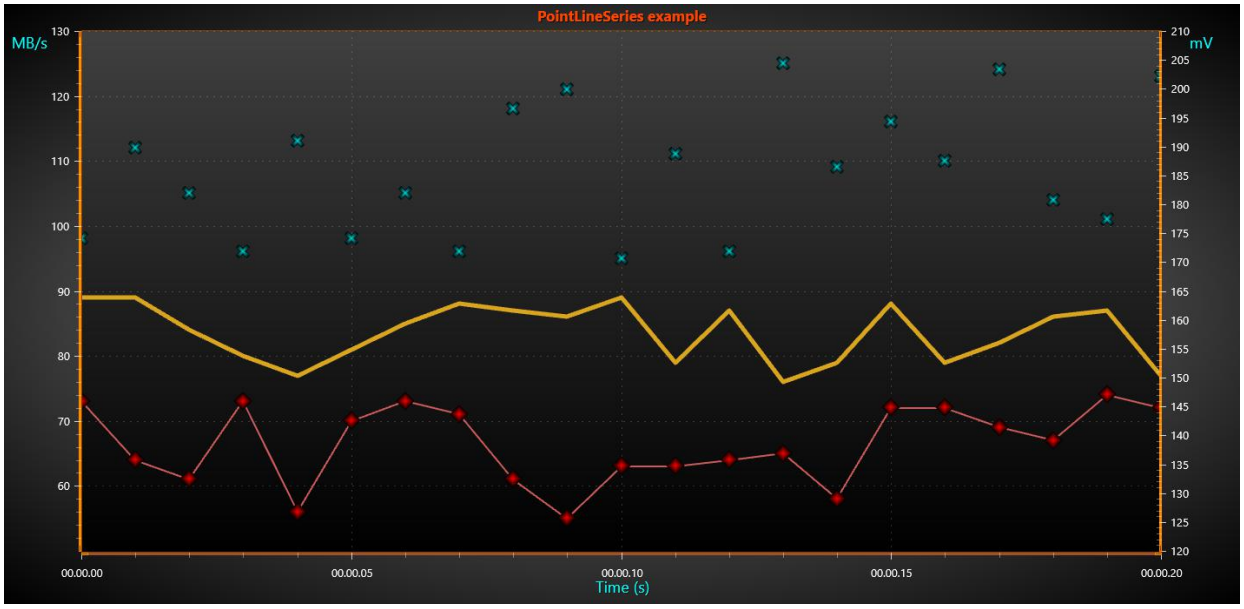
ViewXY 의 시리즈는 다른 방식과 형식으로 데이터 시각화를 허용한다. 모든 시리즈는 축 값 범위에 묶여 있다. 또한 시리즈는 한 Y 축에도 묶여야 한다. 시리즈는 X 와 Y 축을 할당하기 위해 **AssignXAxisIndex** 과 **AssignYAxisIndex** 속성이 있다. 혹은 코드로 시리즈 구성자 매개 변수로 할당하라.

5.6 PointLineSeries

데모 예시: 포인트 라인; 온도 그래프; 라인, 팔레뜨 컬러링



보기 6-40. PointLineSeries 개요



보기 6-41. 세가지의 다른 PointLineSeries.

PointLineSeries 는 간단한 선 또는 점 (스캐터) 또는 점선으로 둘 다를 표현 가능하다. **PointLineSeries** 목록에 **PointLineSeries** 객체를 추가해 차트에 시리즈를 추가하라.

```
chart.ViewXY.PointLineSeries.Add(series); // Add series to the chart
```

5.6.1 라인 스타일

LineStyle 속성으로 라인 스타일을 정의하라. 선이 보이지 않을 시 **LineVisible = false** 설정하라.

5.6.2 포인트 스타일

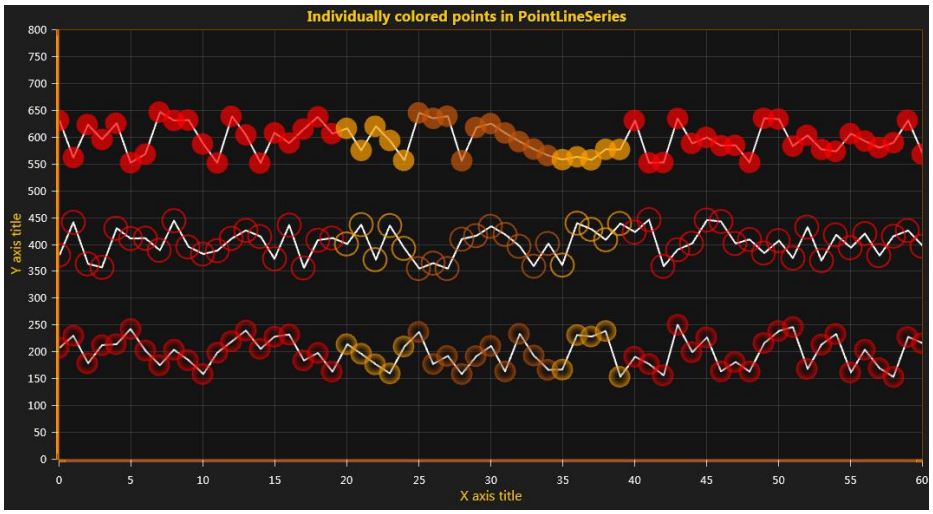
포인트를 보이게 설정하기 위해 **PointsVisible = true** 설정하라. **PointStyle** 속성을 설정하여 포인트 스타일은 수정하라. **PointStyle.Shape** 에서 사전 정의된 여러 스타일에서 모양을 선택하라. 모양 스타일 중 하나는 **Bitmap** 이다. 이는 포인트 위치에서 어느 비트맵 이미지를 그리는 것을 허용한다. **BitmapImage** 속성으로 비트맵 이미지를 정의하라. **BitmapAlphaLevel** 속성은 비트맵의 투명도를 수정하는 데에 사용 가능하다. 비트맵 색 톤을 흰색 외의 색으로 **BitmapImageTintColor** 을 변경하여 수정하라. **Circle**, **Triangle**, **Cross** 등의 사전 정의된 포인트 스타일을 사용 할 때에 그림 색 및 채우기 스타일이 정의 가능하다. 모든 모양 스타일에 모든 색 또는 채우기 스타일 들이 해당 되지 않는 다는 점을 주의 바란다. 포인트 넓이 및 높이는 설정 가능하며 포인트도 회전 가능하다.

5.6.3 개별적으로 포인트 색 칠하기

데모 예시: 포인트 라인, 개별적으로 색칠된 점들, 스캐터 포인트, 개별적 색칠

v.7.2 부터 **PointLineSeries**, **FreeformPointLineSeries**, **AreaSeries** 및 **HighLowSeries** 의 데이터 포인트 구조에는 **PointColor** 필드가 있다.

개별적인 포인트 색칠을 활성화하기 위해 **IndividualPointColoring** 를 **Color1**, **Color2**, **Color3** 또는 **BorderColor** 세팅으로 설정하라. 비활성화 하기 위해서는 **IndividualPointColoring** = **Off** 로 설정하라. 색 설정은 **PointStyle** 속성에 있는 색과 일치한다.



보기 6-24. 위에는 IndividualPointColoring = Color1 (단색 포인트). 중간에는 IndividualPointColoring = BorderColor. 아래에는 IndividualPointColoring = Color2 (Color1 = transparent 로 그라디언트 색)

5.6.4 포인트 추가

시리즈 점들은 코드로 추가해야 한다. **AddPoints(SeriesPoint[], bool invalidate)** 메소드를 사용해 이미 존재하는 포인트 뒤에 포인트를 추가하라.

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

시리즈 데이터 전체를 설정하고 구 포인트들을 덮어 쓰기 위해서는 시규 포인트 배열을 직접 할당하라:

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

주의! PointLineSeries 의 X 값들은 오름차순이어야 한다. 다르게 나열해야 하면 대신 FreeformPointLineSeries 를 사용하라.

예를 들어 Points[0].X = 0, Points[1].X = 5, Points[2].X = 5, Points[3].X = 6 정의는 유효하다.

하지만 Points[0].X = 2, Points[1].X =1, Points[2].X = 6, Points[3].X =7 는 **PointLineSeries** 에 **유요하지 않은** 값 배열이다.

5.6.5 다른 방식으로 포인트 추가

포인트는 또한 X 와 Y 값 배열에도 추가 가능하다. 이는 많은 응용 프로그램에 편리한 방식이다.

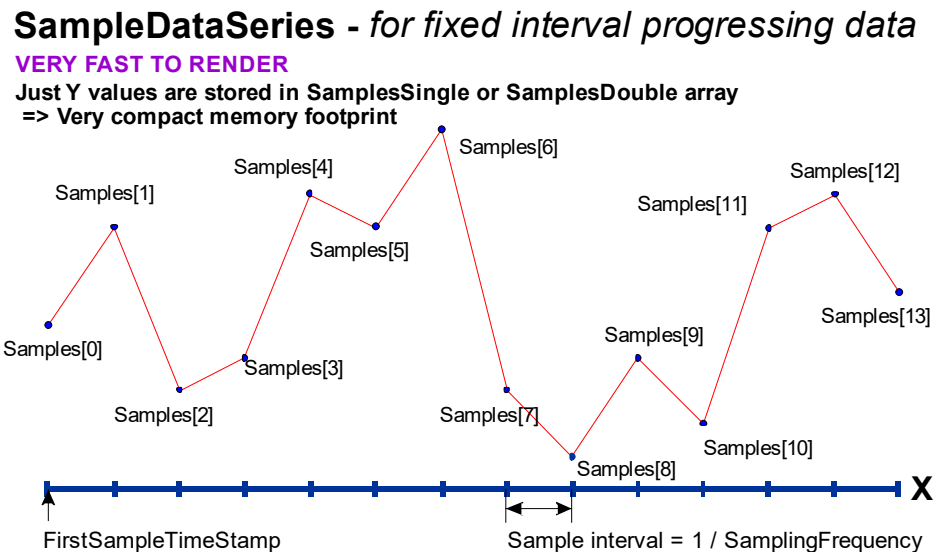
```
chart.ViewXY.PointLineSeries[0].AddPoints(xValuesArray,yValuesArray,false);
```

시리즈 데이터 전체를 설정하고 구 포인트를 덮어 쓰기 위해 X 와 Y 값 배열들을 직접 할당하라 (윈폼 및 WPF 비바인딩 API 에 해당).

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

5.7 SampleDataSeries

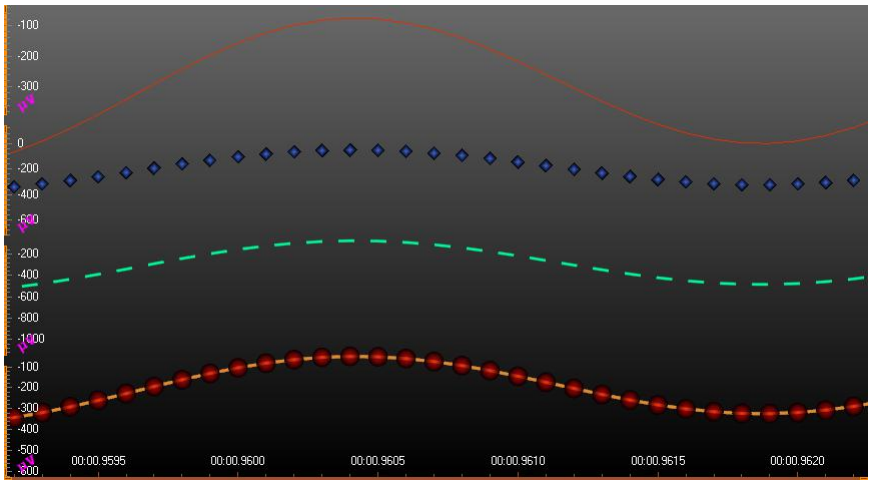
데모 예시: 10 억 포인트; 스레드 페드 멀티 채널 데이터; 시그널 리더



보기 6-43. SampleDataSeries 개요

SampleDataSeries 목록에 **SampleDataSeries** 객체를 추가하여 차트에 시리즈를 추가하라.

```
chart.ViewXY.SampleDataSeries.Add(sampleDataSeries);          //Add          a
SampleDataSeries to the chart
```



보기 6-44. 몇개의 샘플 데이터 시리즈.

SampleDataSeries 는 샘플 시그널 데이터 (이산 시그널 데이터)를 표현하는 데에 사용되는 라인 시리즈다. 이는 주로 실시간 DSP 응용 프로그램에 사용된다. 시각적으로는 **PointLineSeries** 과 비슷해서 모든 라인 및 포인트 형식 옵션들이 적용된다. **SampleDataSeries** 는 고정 샘플 간격이 있기에 x 값을 저장할 메모리를 예약해 둘 필요가 없다.

주의! **SampleDataSeries** 는 주어진 데이터를 재샘플 또는 다운샘플을 하지 않는다. 모든 주어진 데이터 값은 **SamplesSingle** 또는 **SamplesDouble** 배열에 저장이 된다. 라이트닝차트는 데이터의 질을 낮추지도 피크를 잃지도 또는 데이터의 정확도를 낮추지 않는다.

5.7.1 Y 정밀도

SampleDataSeries 는 싱글 및 더블 정밀 샘플 Y 값을 지원한다. 메모리 예약을 최대한 낮게 유지할 때에 싱글 정밀 값을 사용하는 것을 추천한다. **SampleFormat** 속성으로 샘플 형식을 선택하라.

SamplingFrequency (1/샘플 간격) 시리즈를 이용해 고정 샘플 간격을 설정하라. X 값 (타임 스탬프를) 샘플이 시작하는 곳으로 설정하기 위해 **FirstSampleTimeStamp** 속성을 설정하라.

5.7.2 포인트 추가

샘플은 코드로 추가되어야 한다. **AddSamples** 메소드를 사용해 이미 존재하는 샘플 끝에 샘플 추가하라.

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

시리즈 데이터 전체를 한번에 설정하고 구 샘플을 덮어 쓰기 위해 신규 샘플 배열을 직접 할당하라:

SampleFormat 이 **SingleFloat** 면

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

또는 **SampleFormat** 이 **DoubleFloat** 면

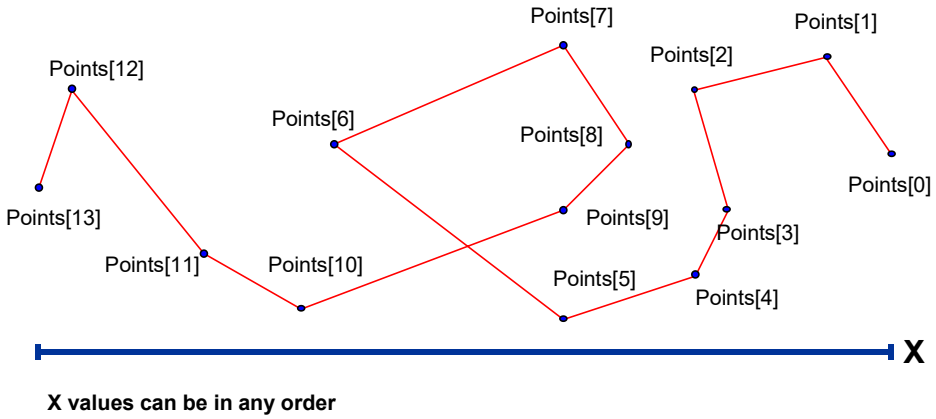
```
chart.ViewXY.SampleDataSeries[0].SamplesDouble = samplesDoubleArray;
```

5.8 FreeformPointLineSeries

데모 예시: 스캐터 포인트; 지도 루트; 마커로 값 트래킹; 커브 노드 수정

FreeformPointLineSeries - *for arbitrary data*

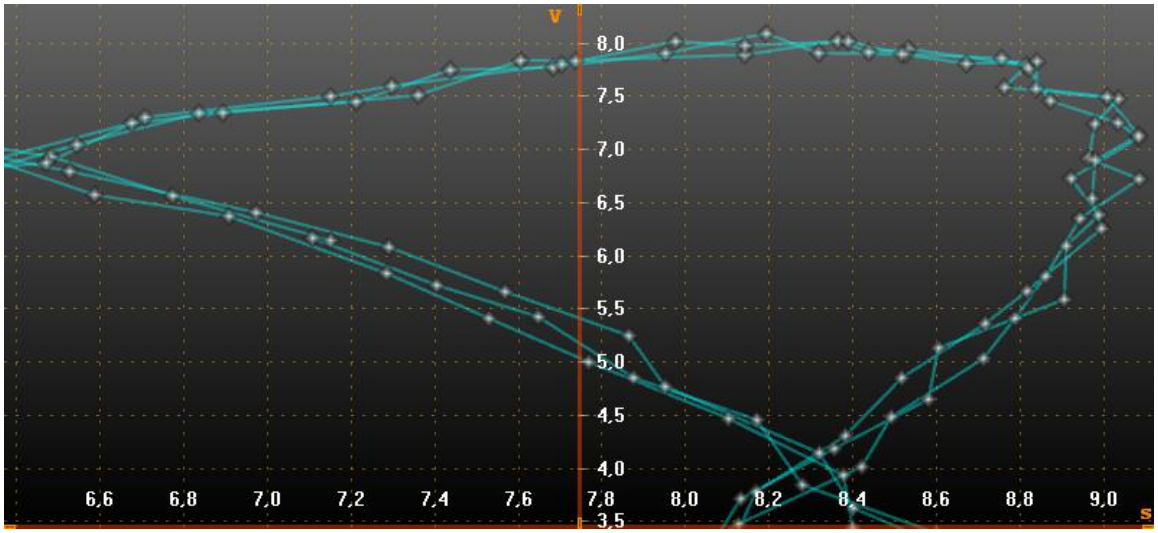
HEAVY TO RENDER WHEN POINT COUNT IS VERY HIGH



보기 6-45. FreeformPointLineSeries 개요

FreeformPointLineSeries 은 간단 라인, 포인트 (스캐터) 또는 둘다를 포인트 라인으로 표현할 수 있다. **FreeformPointLineSeries** 는 이전 포인트에서 아무런 방향으로 라인 포인트를 그리는 것을 허용한다. **PointLineSeries** 의 모든 라인 및 포인트 서식 옵션이 적용된다. **FreeformPointLineSeries** 목록에서 **FreeformPointLineSeries** 객체를 추가해 차트에 시리즈를 추가하라.

```
chart.ViewXY.FreeformPointLineSeries.Add(freeformPointLineSeries);   //Add   a
FreeformPointLineSeries to the chart
```



보기 6-46. 자유형 포인트 라인 시리즈.

자유형 포인트 라인 시리즈 라인 포인트는 **DropOldSeriesData** 가 활성화 되고 포인트가 현재 뷰에서 스크롤 아웃 되어도 자동으로 파괴되지 않는다. 실시간 모니터링 솔루션에서 구 시리즈 포인트를 자동 파괴하기 위해서는 포인트 수 리미터를 사용하라. **PointCountLimitEnabled = true** 으로 설정하고 리미트를 **PointCountLimit** 속성에 설정하라. 리미터가 활성화 되었다면 포인트 카운트 리밋에 달하면 **Points** 배열은 링 버퍼처럼 행동을 한다. Points 배열에서 가장 오래된 포인트는 **OldestPointIndex** 에서 값을 불러 찾을 수 있다. 포인트 카운트 리미트 버퍼에서 존재 데이터를 읽을 필요가 있다면 다음 방식을 이용하라:

- **OldestPointIndex** 가 0이면, **Points[0]** 에서 **Points[PointCount-1]** 까지 읽어라.
- **OldestPointIndex > 0** 이면, 먼저 **Points[OldestPointIndex]** 에서 **Points[PointCountLimit-1]** 까지 읽어라. 그 후 **Points[0]** 에서 **Points[OldestPointIndex-1]** 까지 읽어라.

마지막 시리즈 포인트를 직접 부르기 위해서는 **GetLastPoint()** 메소드를 불러라.

5.9 라인 시리즈 고급 라인 색 칠하기

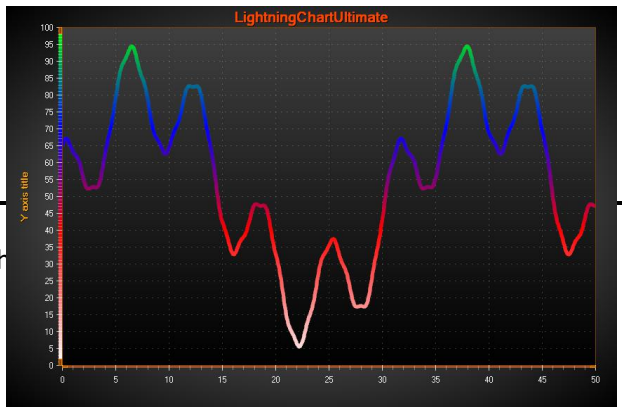
데모 예시: 라인, 팔레뜨 컬러; 라인, 정수 별 이벤트 기반 색칠; 라인, 이벤트 기반 컬러

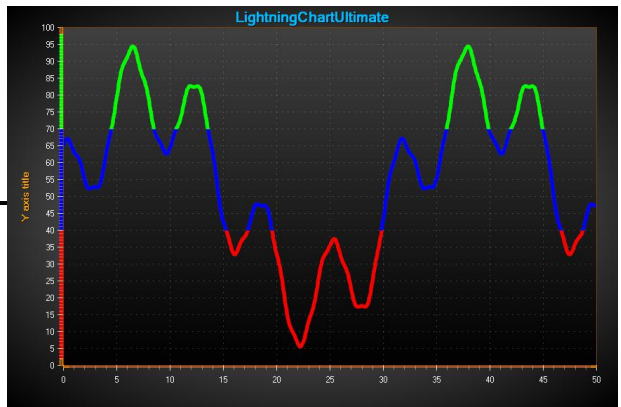
라인 색은 데이터 값에 따라 또는 기타 외부 로직에 따라 바꿀 수 있다.

5.9.1 Y-값 기반 라인 컬러 및 값-범위 팔레뜨로 채우기

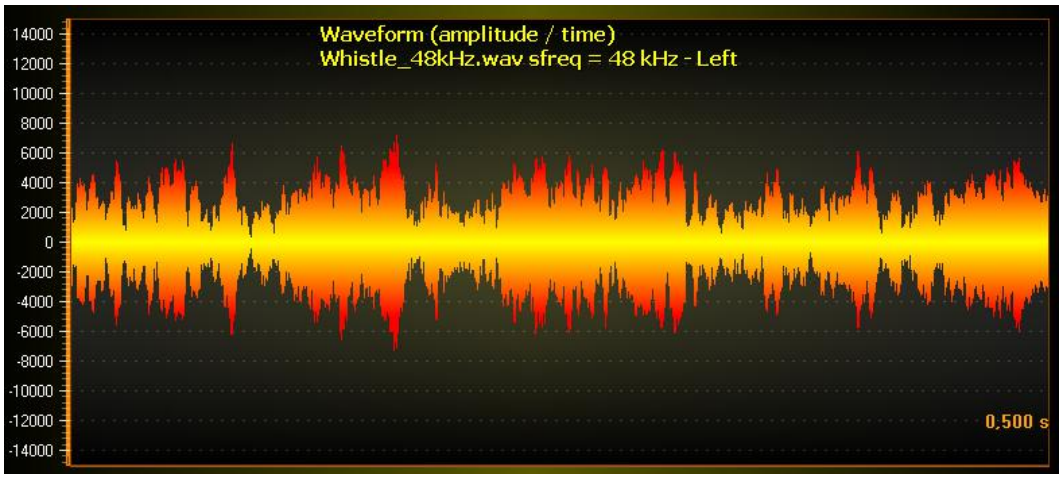
SampleDataSeries, **PointLineSeries** 또는 **FreeformPointLineSeries** 의 **UsePalette** 속성을 활성화 함으로써 라인의 색칠은 **ValueRangePalette** 속성 대로 적용된다. **ValueRangePalette** 은 Y 값 및 컬러 페어를 담고 있다. **ValueRangePalette**.**Type** 은 **Gradient** 또는 **Uniform** 스텝 팔레뜨를 설정한다.

팔레뜨 색칠은 Y 축 라인에도 설정 가능하다. Y 축의 **UsePalette** 속성을 활성화 하고 선호 시리즈를 **PaletteSeries** 속성에 할당하라.





6-47. 좌, Gradient 팔레뜨를 이용한 Y 값 기반 라인 컬러. 우, Uniform 팔레뜨 사용. Y 축에도 UsePalette 가 활성화 되었다.

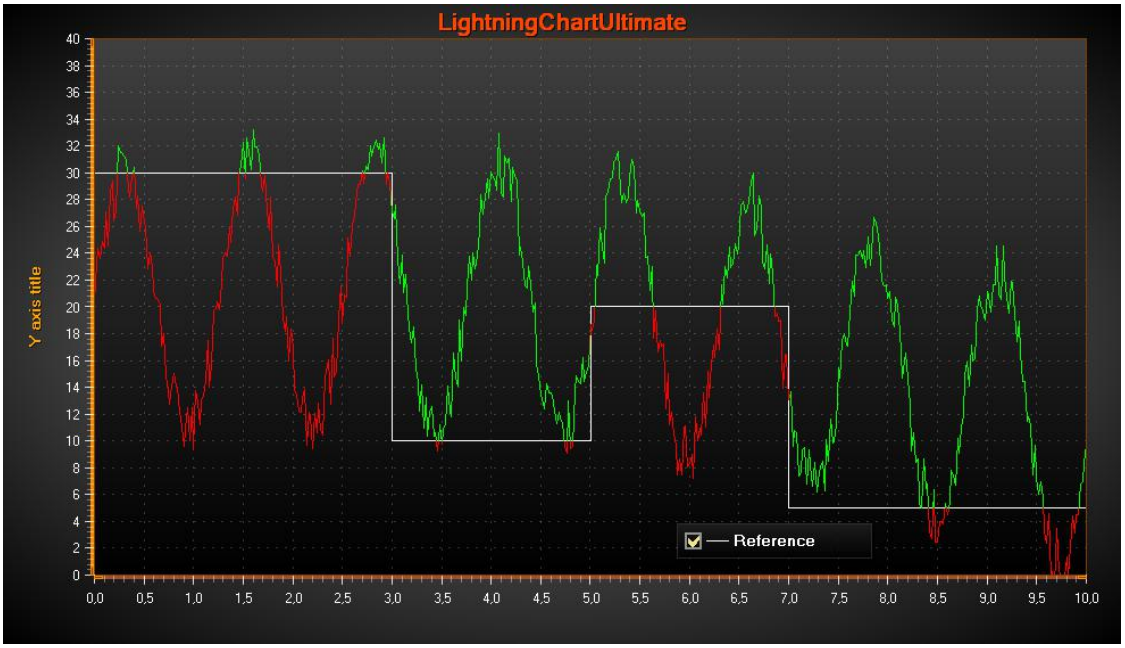


보기 6-48. 양극성 시그널 데이터를 위한 Gradient 팔레뜨 컬러. Y 축에는 UsePalette 가 비활성화 되었다.

5.9.2 CustomLinePointColoringAndShaping 이벤트로 커스텀 형성 및 컬러

CustomLinePointColoringAndShaping 이벤트로 커스텀 컬러 및 좌표 조정이 가능하다. 이는 차트의 렌더링 스테이지 입성 직정 불러진다.

LineStyle.Pattern = Solid 일 때에 커스텀 컬러링을 사용 가능하다. 벡터 파일 내보낼 때에 많은 제한들이 있다.



보기 6-49. CustomLinePointColorAndShaping 이벤트 핸들러를 이용해 바뀌는 참조 레벨에 의해 라인 색을 바꾸기.

이벤트 독립변수는 다음과 같은 정보를 갖고 있다:

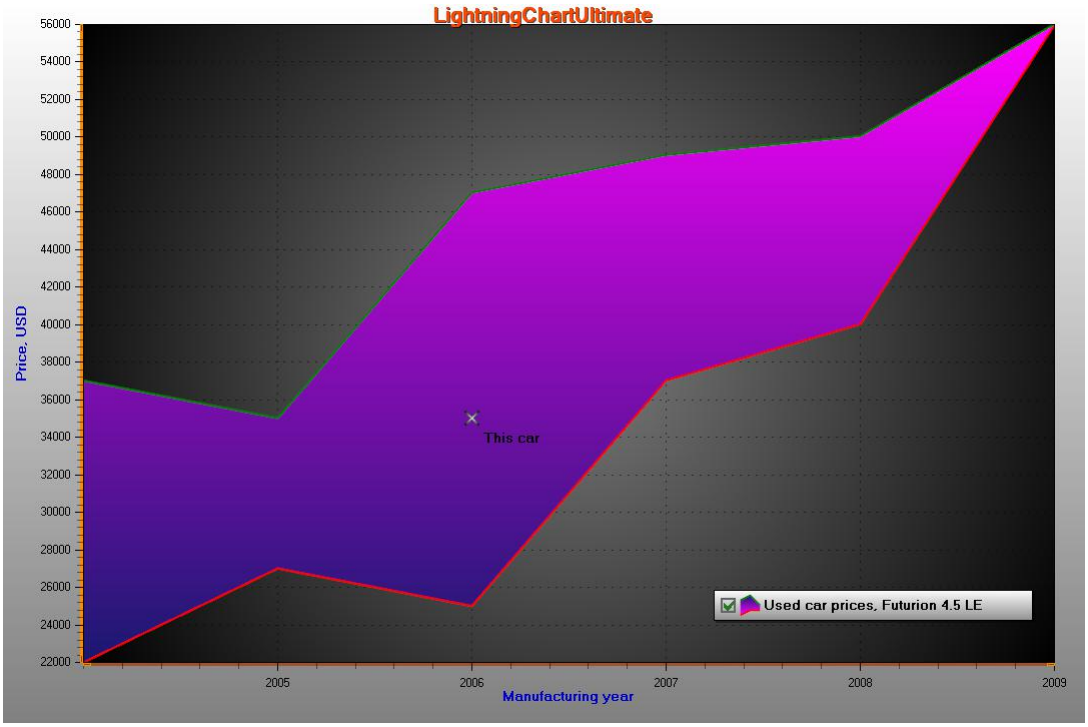
- **CanModifyColors**: 색갈 변경을 할 수 있다.
- **Colors**: LineStyle 색으로 사전에 채워진 컬러 배열. **CanModifyColors** 가 참이면, 사전 채워진 색들에 새로운 값을 할당 해서 변경하거나 새로운 색 배열을 생성해서 변경 가능하다. **CanModifyColors** 가 거짓이면 채우지 마라.
- **CanModifyCoords**: 좌표 변경이 가능하다.
- **Coords**: 사전에 채워진 화면 좌표 배열. **CanModifyCoords** 가 참이면 사전 채워진 좌표에 새로운 값을 할당하거나 새로운 좌표 배열을 생성해서 변경 가능하다. 새로운 배열의 길이는 사전 채워진 배열의 길이와 같지 않아도 된다. **이벤트 핸들러 종료 전 좌표 및 색 배열의 길이가 같은 것을 확인하라. CanModifyCoords** 가 거짓이면 채우지 마라.
- **HasDataPointIndices**: **FreeformPointLineSeries** 에만 해당.
- **DataPointIndices**: 좌표 및 색 배열에 포함된 데이터 포인트 정수. X 및 Y 값 또는 좌표가 같으면 차트는 라인 구성할 때 뒤 포인트들을 건너 뛴다. DataPointIndices를 이용하여, 예를 들어 데이터 포인트의 **PointColor** 필드 또는 외부 색 배열에 라인 포인트의 색을 선택할 수 있다.
- **SweepPageIndex** : **XAxis.ScrollMode** = **'Sweeping'** 이면 페이지 인덱스를 말해준다 (0 또는 1)

5.10 하이 로우 시리즈

데모 예시: 하이-로우; 스택 구역; 이전 클로즈 있는 스탁 코스; 구역/ 하이-로우; 스케일 브레이크

하이-로우 시리즈는 최고 및 최저 값들 사이가 채워진 구역으로 데이터를 표현한다. **HighLowSeries** 목록에 **HighLowSeries** 객체들을 추가하여 시리즈를 차트에 추가하라.

```
//Add high-low series to the chart
chart.ViewXY.HighLowSeries.Add(highlowSeries);
```

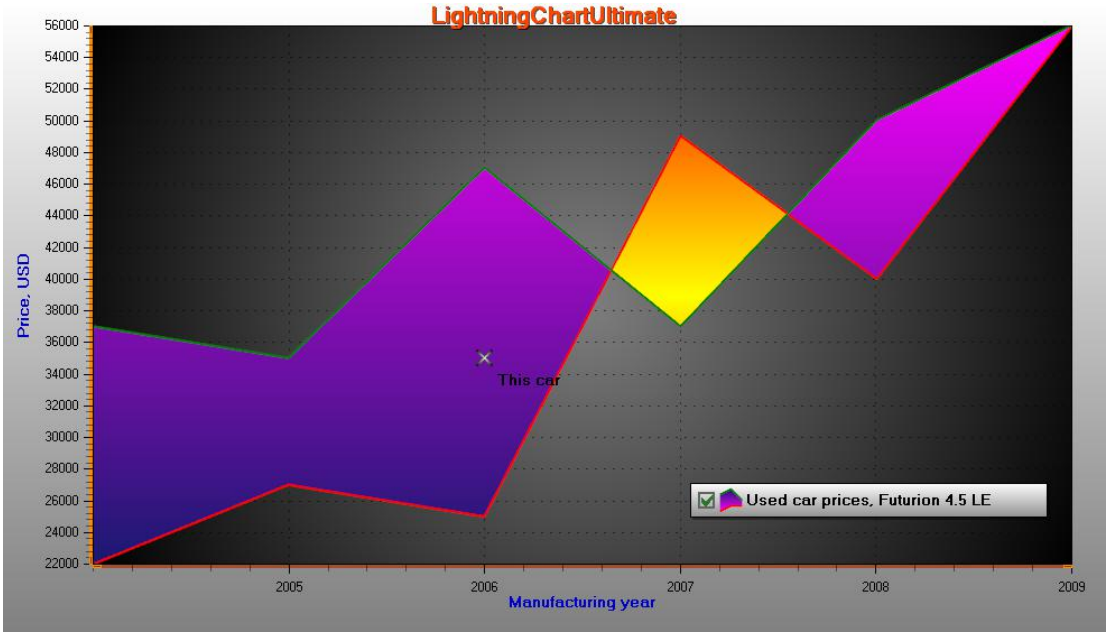


보기 6-50. 위에 마커가 있는 하이-로우 시리즈.

5.10.1 채우기, 라인 및 포인트 스타일

Fill 속성 및 하위 속성들로 채우기를 설정할 수 있다. **LineStyleHigh** 및 **LineStyleLow** 속성들로 라인 스타일을 정의하라. 라인이 보이지 않게 설정하기 위해서는 **LineVisibleHigh = false** 및 **LineVisibleLow = false** 로 설정하라. **PointStyleHigh** 및 **PointStyleLow** 속성들로 포인트 스타일을 정의하라. 포인트가 보이지 않게 설정 하기 위해 **PointsVisibleHigh = false**, **PointsVisibleLow = false** 와 같이 설정하라.

더 많은 라인 및 포인트 스타일 상세 정보를 위해 제 6.6.1 및 6.6.2 장을 보세요. 데이터의 최고 값이 최저 값



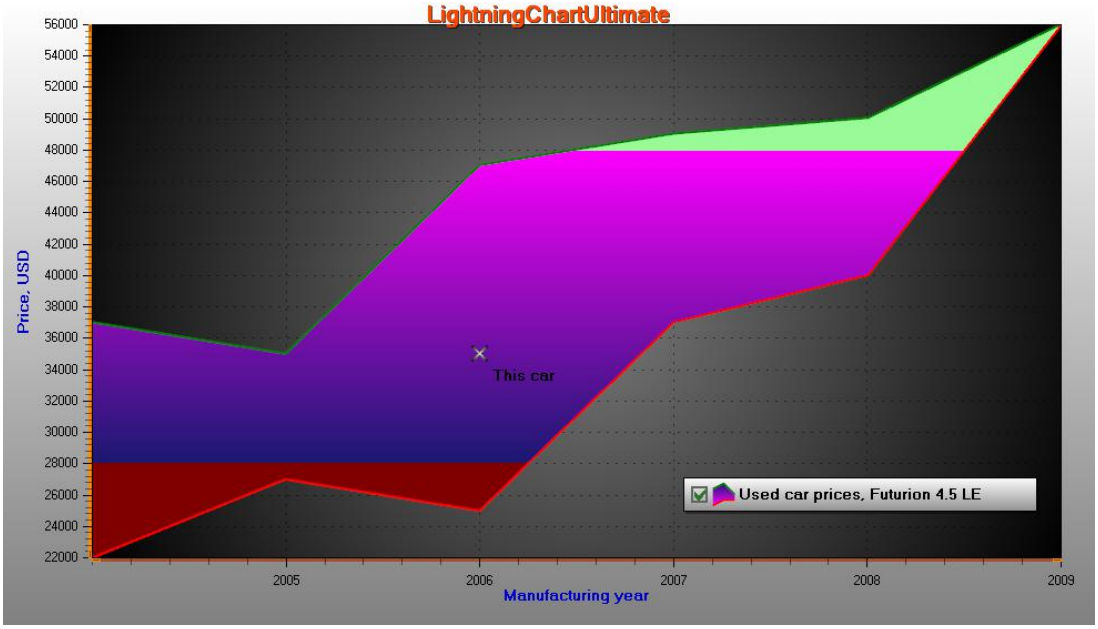
보다 낮을 시 해당 부분에는 역 채우기가 적용된다. 역 채우기를 **REversFill** 속성으로 수정하라.

보기 6-51. 네번째로 데이터 건이 역으로 주어졌다: 최고 값 < 최저 값.

5.10.2 리미트

UseLimits 를 활성화 함으로써 시리즈는 한도 초과 및 기만 한도 이하에 다른 채우기 스타일을 보여준다. 일

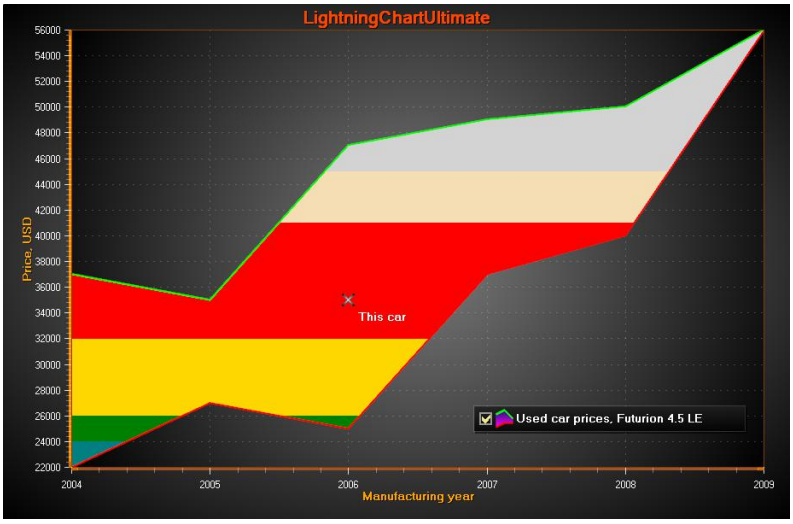
반 **Fill** 및 **ReverseFill** 은 리미트 사이 범위에만 적용된다.



보기 6- 52. UseLimits = true, ExceedLimit = 48000 및 DeceedLimit = 28000.

5.10.3 값-범위 팔레뜨로 색 칠하기

UsePalette 를 활성화 함으로써 채우기 기능은 **ValueRangePalette** 단계를 따른다. **Uniform** 및 **Gradient** 칠하기 둘다 지원된다.



보기 6-53. UsePalette = True, ValueRangePalette 에 여러 단계 정의됨. 일정한 색칠.

5.10.4 데이터 추가

데이터 값은 코드로 추가해야 한다. 데이터는 X 값 오르는 순으로 주어져야 한다, **Points[i+1].X ≥ Points[i].X.**

AddValues(HighLowSeriesPoint[], bool invalidate) 메소드를 이용해 존재하는 값 배열 끝에 데이터 값을 추가하라.

```
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);
```

이전 데이터를 덮어 쓰고 시리즈 데이터 전체를 한번에 설정하기 위해 새로운 데이터 배열을 직접 할당하라.

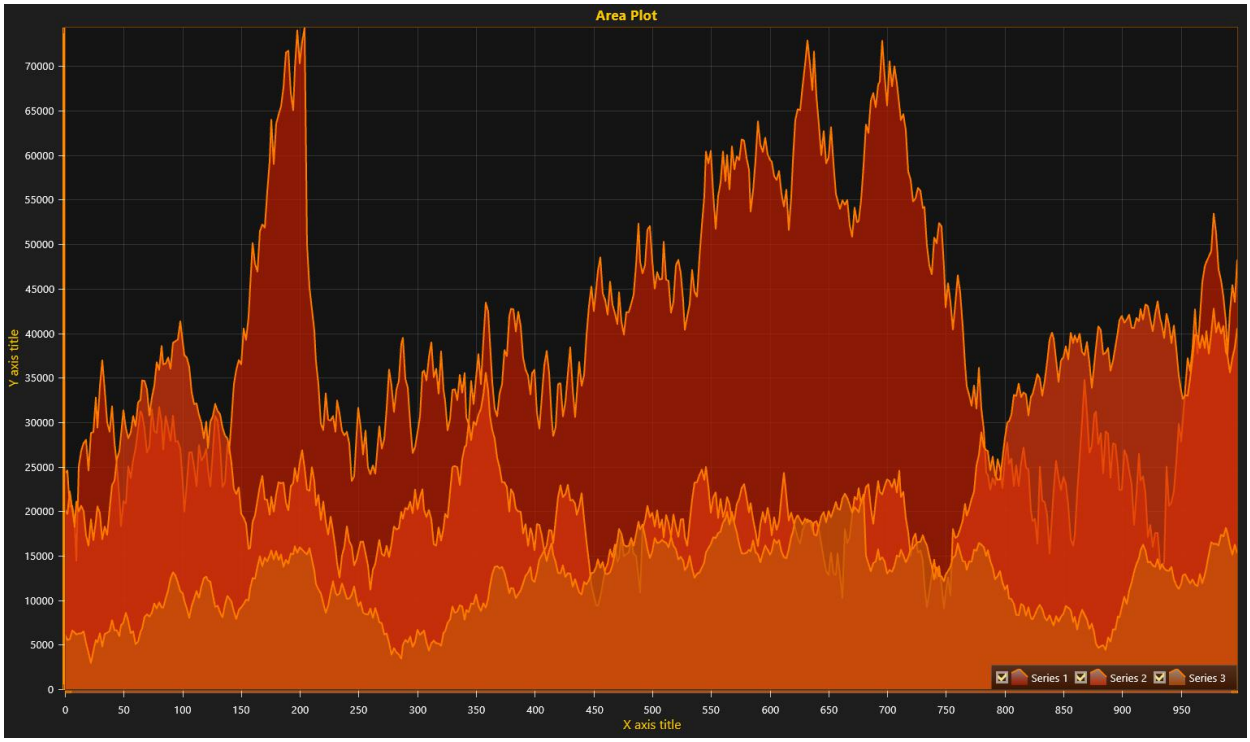
```
//Assign the data into points array
chart.ViewXY.HighLowSeries[0].Points = dataArray;
```

5.11 AreaSeries

데모 예시: 구역; 구역들; 시리즈로 데이터 브레이크; 여러 레전드; 커스텀 축 티크

에리어 시리즈는 데이터를 베이스 레벨과 값들 사이 채워진 구역으로 표시한다. 에리어 시리즈는 제 6.10장에 설명된 **HighLowSeries** 와 비슷하지만 간단하다. **AreaSeries** 목록에 **AreaSeries** 객체를 추가하여 차트에 시리즈를 추가하라.

```
chart.ViewXY.AreaSeries.Add(areaSeries);  //Add area series to the chart
```



보기 6-54. 세 에리어 시리즈 모두 BaseValue = 0.

BaseValue 속성으로 베이스 레벨을 설정하라. 선호 채우기 스타일을 **Fill** 속성으로 설정하라. **LineStyle** 속성으로 라인 스타일 설정 가능하며 **PointStyle** 속성으로 포인트 스타일 설정 가능하다. **HighLowSeries** 에서와 같이 초과 기만 제한 사용 가능하다.

5.11.1 데이터 추가

데이터 값은 코드로 추가해야 한다. 데이터는 x 값이 오르는 순으로 주어져야 한다. **Points[i+1].X ≥ Points[i].X**.

AddValues(AreaSeriesPoint[], bool invalidate) 메소드를 이용해 존재하는 값 배열 끝에 데이터 값을 추가하라.

```
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);
```

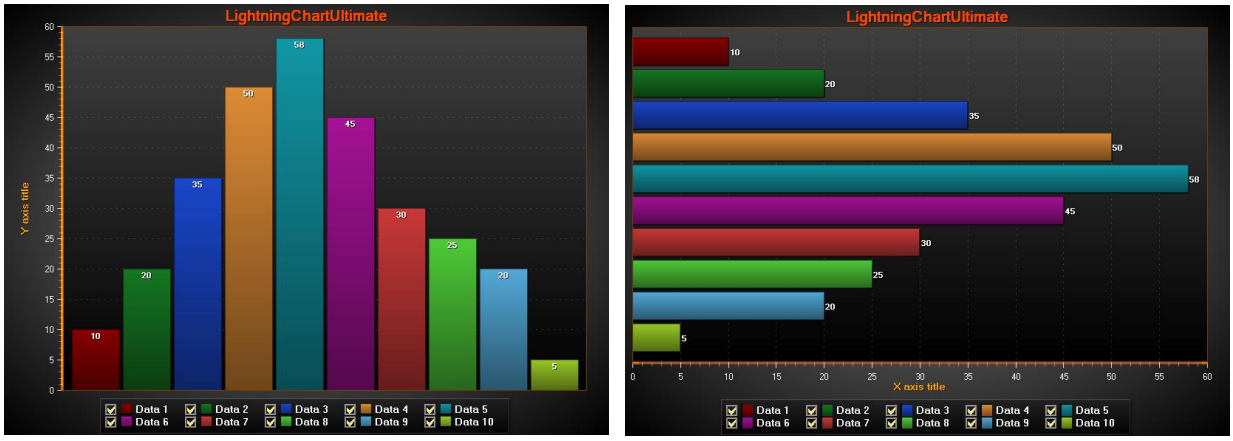
이전 데이터를 덮어 쓰고 시리즈 데이터 전체를 설정하기 위해 새로운 값 배열을 바로 할당하라.

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

5.12 BarSeries

데모 예시: 수직; 수평; 음수 값; 스택 막대

BarSeries 는 데이터를 수평 또는 수직 막대로 표시한다.



보기 6-55. 막대 시리즈, 수직 및 수평

Values 배열 속성을 이용해 막대 시리즈의 값을 저장하라. **AddValue(…)** 메소드로 값을 추가하라. 존재하는 값을 다른 값 인덱스로 **SetValue(…)** 메소드를 이용해 업데이트 하라. **BarSeriesValue** 종류의 값이며 다음과 같은 필드들을 갖고 있다:

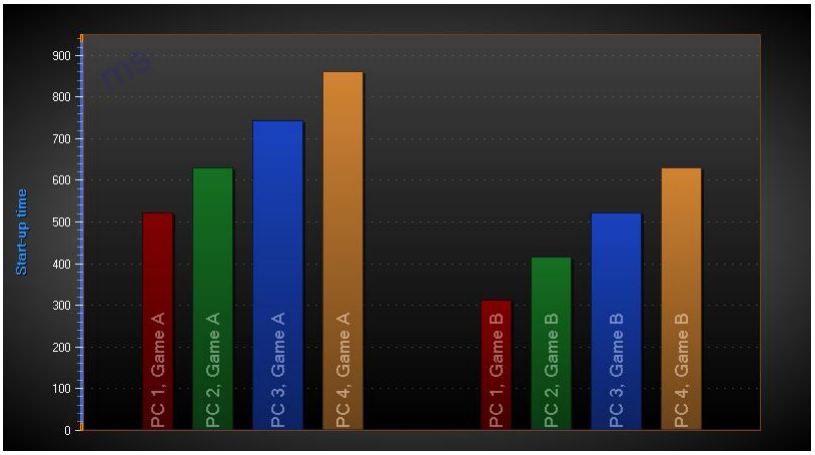
- **값** 막대 길이.
- **위치** 막대의 X 축 위치 (수직 표시) 또는 Y 축 위치 (수평 표현).
- **텍스** 막대 안에 나타나는 문구.

막대 시리즈의 **LabelStyle** 속성을 이용해 막대 값 라벨이 차트에 어떻게 나타나는지 제어하라. 라벨 값 텍스트는 **AddValue(…)** 또는 **SetValue(…)** 메소드 매개 변수로 설정된다. 여러 채우기 스타일이 **Fill** 속성 및 하위 속성을 설정해 사용 가능하다.

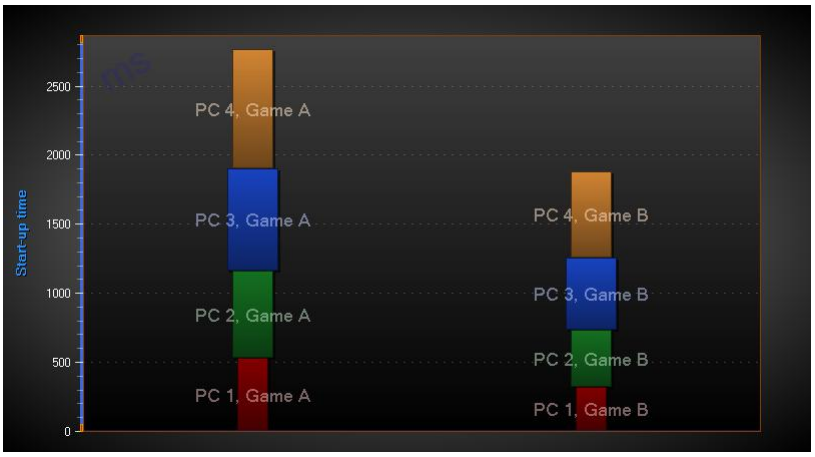
차트의 **BarViewOptions** 속성을 이용해 막대의 표시법을 제어하라. **BarView.Options.Orientation** 을 이요해 수평 및 수직 옵션 사이 선택 가능하다.

막대 지수가 값 넓이 피팅 또는 위치 값을 이용하면 **BarViewOptions.Grouping** 로 값 지수대로 막대를 그룹화 가능하다. 시각적으로 다른 막대 시리즈의 값들을 모아준다. 그룹화를 원치 않을 경우, **BarViewOptions.Grouping.ByLocation** 를 이용해 각 BarSeriesValue 객체마다 **Location** 필드를 다르게 설정하

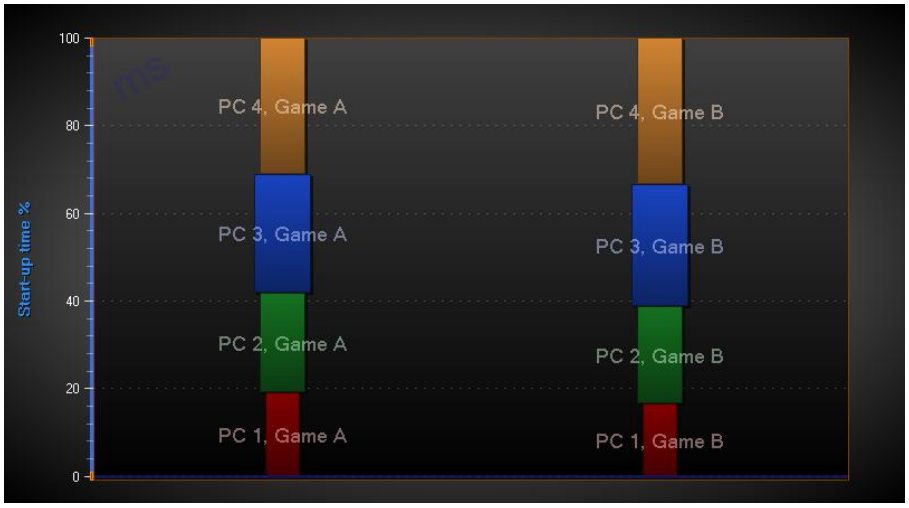
라. 넓이 피팅 속성으로 기둥 사이 공간의 넓이를 수정하라. 넓이 피팅이 사용 되지 않을 시 막대 시리즈의 **BarThickness** 속성이 막대 넓이를 정한다. **BarViewOptions.Stacking** 를 **Stack 또는 StackStretchToSum** 로 설정해 그룹을 스택 가능하다. **StackStretchToSum** 를 사용할 때 목표 합을 **StackSum** 속성을 설정해 정의하라. 기본 설정 값은 100%를 표현하는 100 으로 설정 되어있다.



보기 6-56. 막대 시리즈 Grouping = ByIndex, Stacking = None.

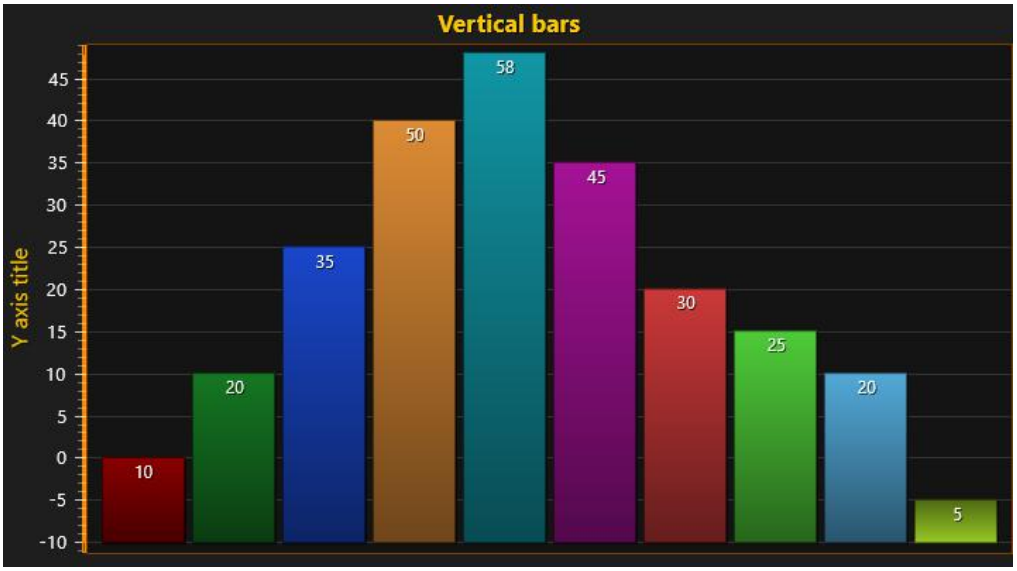


보기 6- 57. 막대 시리즈 Grouping = ByIndex, Stacking = Stack.



보기 6- 58. 막대 시리즈 Grouping = ByIndex, Stacking = StackStretchToSum. StackSum = 100.

BarSeries 의 **BaseLevel** 속성은 모든 값의 시리즈 최소 값이며 막대 시작 위치를 설정한다. **Stacked** 뷰에서 막대의 크기를 (양수면) 증가하고 (음수면) 감수한다. **StackedToSum** 인 경우, 막대 크기는 상대적이고 **Stacked** 와 같이 계산된다.



보기 6- 59. BaseLevel -10 으로 설정. 막대 값은 10, 20, 35, 50, 58, 45, 30, 25, 20, 5.

5.13 StockSeries

데모 예시: 스플리터 있는 세그먼트; 스탁 및 막대; 스케일 브레이크; 통계 분석

스톡 시리즈는 주가 데이터 시각화를 촛대 또는 스톡 막대 형식으로 이룬다. 같은 차트에 여러 스톡 시리즈를 추가 가능하다. **StockSeries** 목록 속성에 **StockSeries** 객체를 추가하라. Style 속성으로 스타일을 선택하라. 선택 가능 옵션은: 막대, 촛대 및 **OptimizedCandleStick** 이다.

색 및 채우기 옵션을 **ColorStickDown**, **ColorStickUp**, **FillDown** 및 **FillUp** 속성들로 설정하라. 막대 픽셀 넓이를 **StickWidth** 속성으로 수정하라. 전체 데이터 아이템 넓이를 **ItemWidth** 속성으로 수정하라.

최고 렌더링 성능을 위해서 **Bars** 스타일 및 **StickWidth=1** 을 사용하라.

OptimizedCandleStick 은 성능적인 이유들을 위해 v.8.4 부터 기본으로 설정 되어있다. 하지만 **OptimizedCandleStick** 은 몇가지의 효과들 밖에 없다 – **Solid** 및 좌-우 방향의 **Linear** 채우기를 지원한다. **Bitmap**, **Radial**, **RadialStretched** 및 **Cylindrical** 채우기 옵션들은 지원되지 않는다. **OptimizedCandleSticks** 는 촛대 태두리를 지원하지 않는다- **FillBorder** 가 해당되지 않는다. **Style = CandleStick** 로 설정하여 더욱 고급 생김세 관련 옵션을 사용하세요.

StockSeries 는 **Behind = True** 로 설정하여 라인 시리즈 보다 먼저 렌더링 설정 할 수 있다.

보기 6-60. Style = CandleStick 로 설정된 StockSeries. 연파란 선은 배경에서 Close 값들 사이 지나가는 PointLineSeries 이다.

보기 6-61. Style = Bars 로 설정 된 StockSeries. 선형 회귀 피트 및 라인의 오프셋을 보여주기 우해 라인 시리즈가 사용됐다 (2 * 표준 편차). 라인 피트를 위한 날짜 범위를 선택하기 위해 밴드가 사용됐다.

5.13.1 StockSeries 에 데이터 설정

데이터 배열을 생성해 배열 아이템을 설정하라. 각 아이템은 다음과 같은 필드를 갖고 있다:

날짜	DateTime 값 (년, 월, 일)
오픈	하루의 오픈 값
클로즈	하루의 클로즈 값
로우	하루 중 최소 값
하이	하루 중 최대 값
거래	총 거래액
볼륨	거래된 주식 수

데이터를 Date 값이 오르는 순으로 유지하라 (최신 날짜 나중).

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

5.13.2 X 축을 날짜 디스플레이로 설정

```
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 division 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;
```

x 축 범위를 데이터 알맞게 설정하라:

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

5.13.3 생김세 커스텀 서식

StockSeries 에는 **CustomStockDataAppearance** 이벤트 핸들러가 있다. 이는 시리즈 데이터 아이템 별로 생김세 서식하는 데에 사용 할 수 있다. 이는 속성으로 적용된 기본 채우기 및 색 스타일을 덮어 쓴다. 이 벤트 핸들러에서 주어진 포인트 마다 넓이 및 색을 변경하라.

보기 6-62. CustomStockDataAppearance 이용해 주어진 데이터 아이템을 넓이와 더욱 밝은 그라디언튼 색으로 강조.

5.13.4 스케일 브레이크 적용

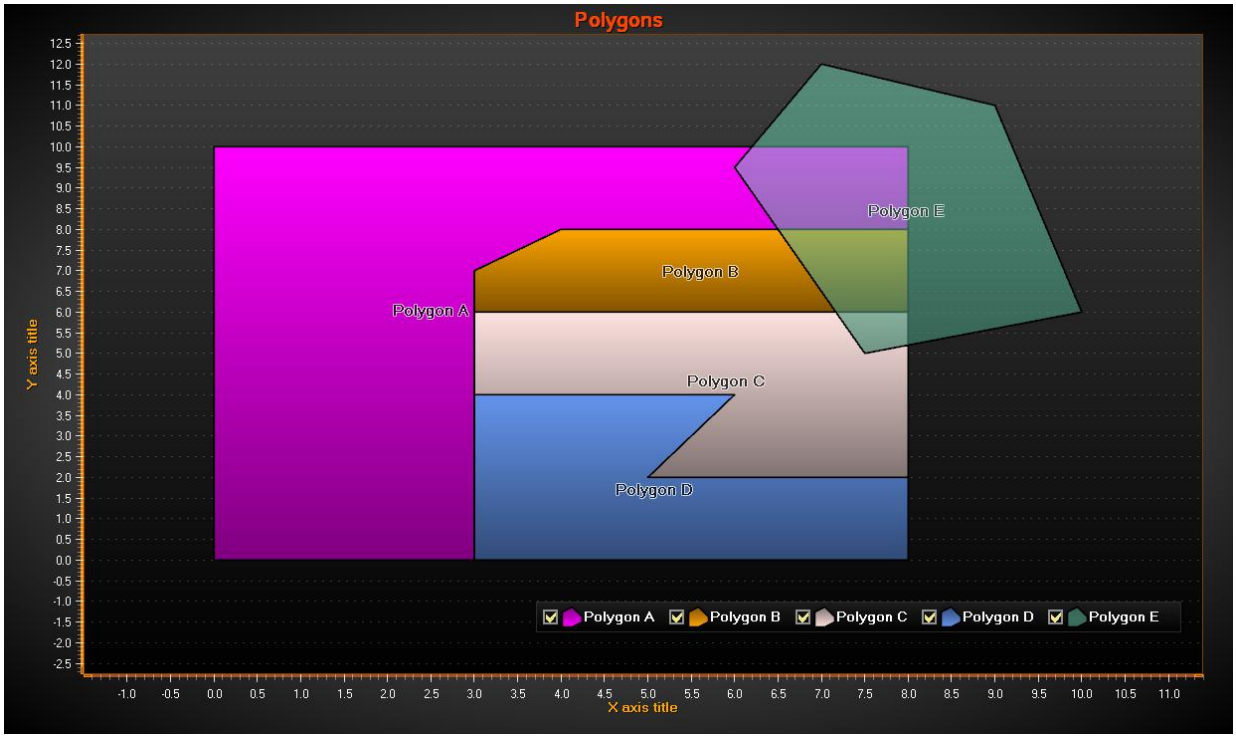
비 거래 시간 및 날들을 끊기 위해 5.3.2 를 보세요.

5.14 PolygonSeries

데모 계시: 다각형: 상자 수염 그림; 삼항 그래프; 이미지 뷰어; Zoomable 2D 파이

PolygonSeries 는 채우기 및 태두리를 주어진 태두리 길로 렌더링 한다.

Fill 속성에서 채우기 환경 설정하라. **PolygonSeries** 의 **Border** 속성을 이용해 태두리 라인 스타일을 설정하라.



보기 6-63. 여러 다각형.

5.14.1 Polygon 에 데이터 설정

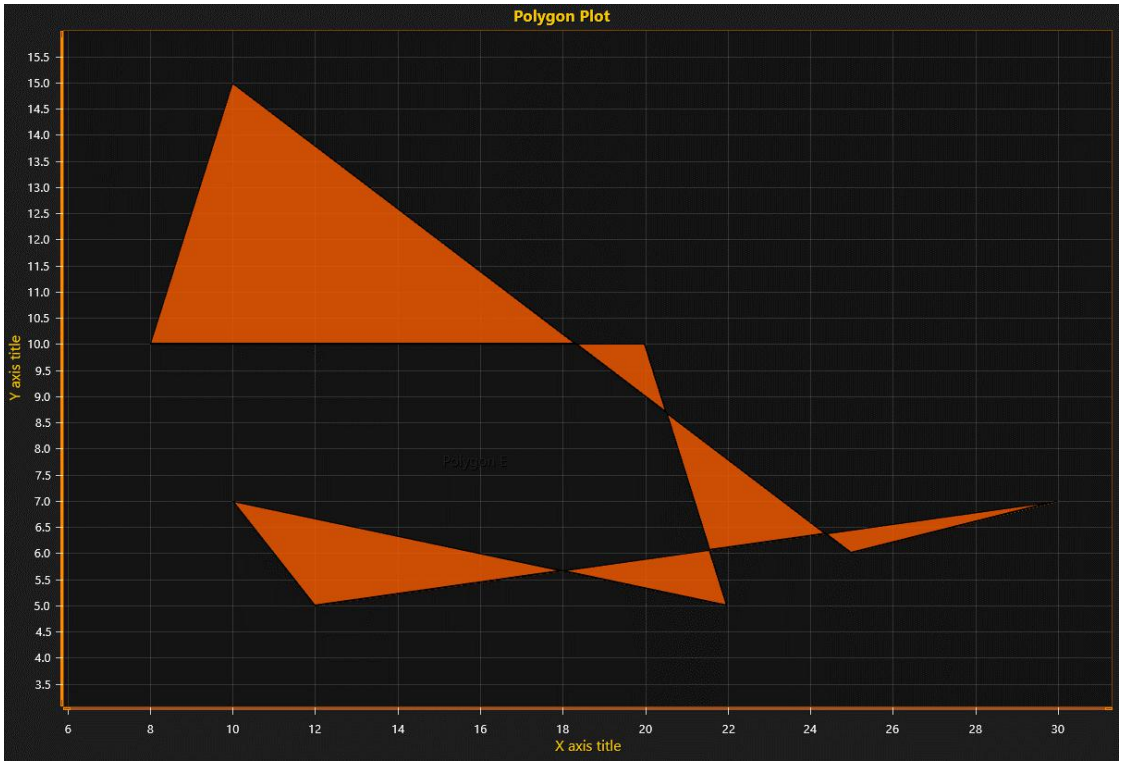
길 포인트를 **Points** 속성에서 설정하라. **PolygonSeries** 은 자동 길 클로징 기능이 있다. 마지막 포인트가 첫 포인트에 연결 되지 않았다면 차트가 자동으로 연결을 한다.

다음과 같이 이전 그림의 투명 다각형 길을 할당 할 수 있다:

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

5.14.2 복잡 / 교차 채우기 활성화

IntersectionsAllowed = True 로 설정해 다각형 길이 자신과 교차 할 수 있게 활성화 하라. 이 속성의 활성화 없이는 길이 교차될 때 채우기가 이상하게 나온다. 이 속성의 기본 설정은 성능을 의해 **False** 로 설정 되어 있다. 교차 경우 감지 및 렌더링은 많은 성능을 잡아 먹기 때문이다.



보기 6-64 길이 교차하는 다각형. IntersectionsAllowed = True 설정.

5.15 LineCollections

데모 예시: 라인 컬렉션; 라인 스펙토그램; 스템 플롯

LineCollection 은 라인 세그먼트의 모음이다. 각 라인 세그먼트는 A 와 B 지점을 이으는 선이다. 한 **LineCollection** 은 수 천개의 라인 세그먼트를 가질 수 있다. **LineCollection** 는 구별된 수 천개의 라인 세그먼트를 효율적으로 렌더링한다. **이는 PointLineSeries**, **FreeformPointLineSeries** 또는 **SampleDataSeries** 와 다르다. **PointLineSeries**, **FreeformPointLineSeries** 또는 **SampleDataSeries** 는 수 백만개의 이어진 포인트를 렌더링하는 데에 더욱 효율적이다.

LineStyle 속성을 이용해 라인 색, 스타일 및 넓이를 제어하라. **Lines** 속성에서 라인 세그먼트 설정하라.

ViewXY.LineCollections 목록 속성에 **LineCollection** 객체를 추가하라.

보기 6-65. 세개의 LineCollections 사용중. 초록은 아주 바른 렌더링 막대로 쓰여지고, 노랑은 다각선, 빨강은 임의의 삼각형 화이어 프레임 매쉬.

5.15.1 LineCollection 에 데이터 설정

SegmentLine 구형은 4 개의 필드로 구성 되었다:

AX 시작점, X

AY 시작점, Y

BX **끝 지점**, X

BY **끝 지점**, Y

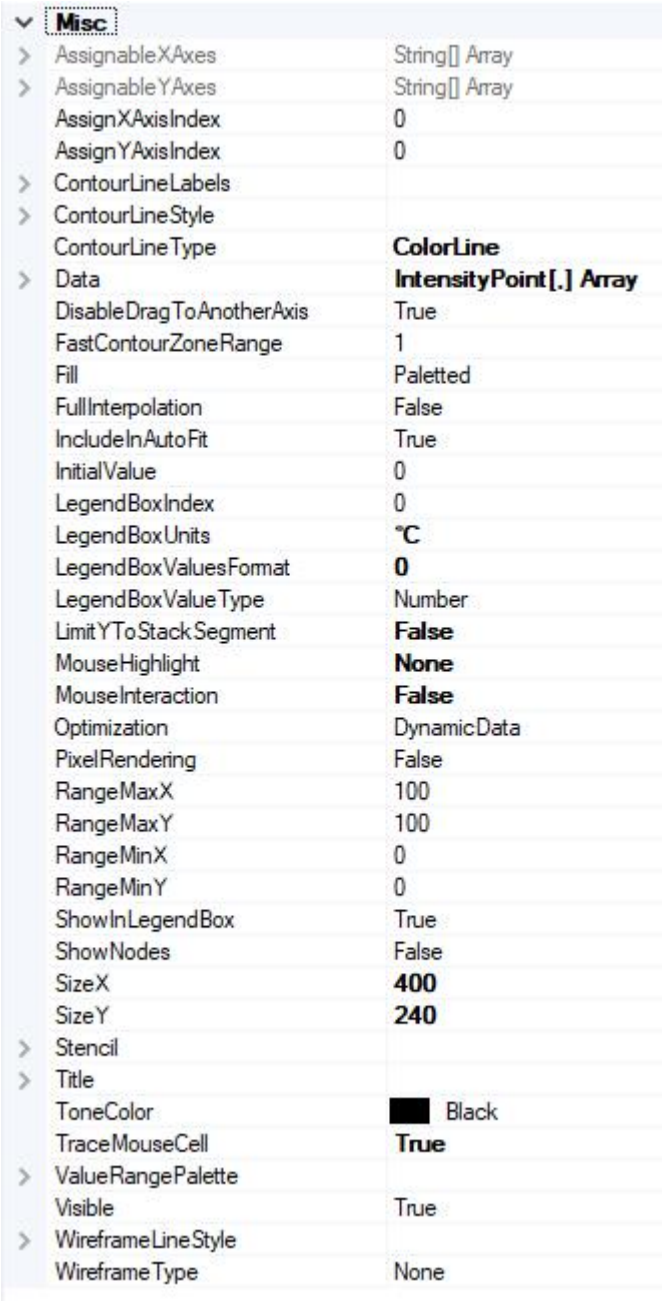
Lines 속성에 다음과 같이 **SegmentLines** 배열을 추가하라:

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

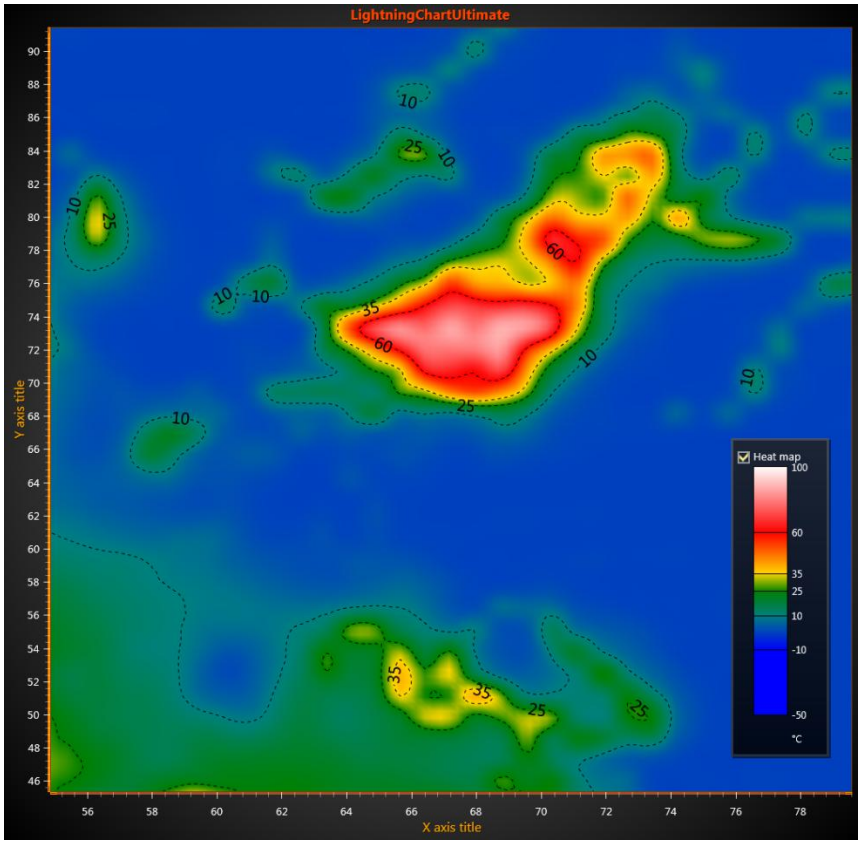
5.16 IntensityGridSeries

데모 예시: 열지도; 스펙토그램; 강도 그리드 마우스 제어

IntensityGridSeries 는 M x N 배열의 노드의 시각화를 허용한다. 값 범위 팔레뜨에 할당된 색으로 칠해진다. 노드 사이 색들은 보간 되었다. **IntensityGridSeries** 는 X 와 Y 차원에 있는 일정한 간격의 직사각형 시리즈다. 이 시리즈는 등고선, 등고선 라벨 및 와이어 프레임 렌더링도 허용한다.

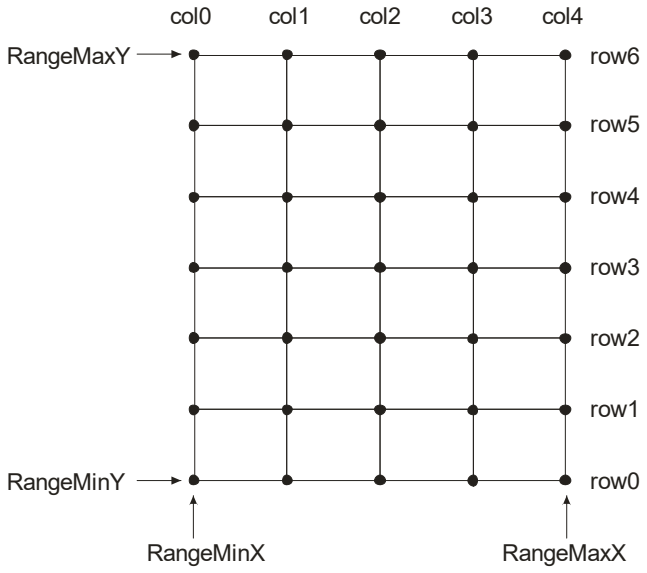


보기 6- 66. IntensityGridSeries 속성.



6-67. 열지도 표현을 하는 IntensityGrid 시리즈. 레전드 상자가 값 범위 색 팔레드를 보여준다.

데이터는 **Data** 속성에 이차원 배열로 저장된다. 각 배열 아이텐은 **IntensityPoint** 종류다. **IntensityPoint** 구형의 **Value** 필드에 각 노드의 데이터 값을 저장하라. 이는 **ValueRangePalette** 에서 무슨 색을 사용해야 하는지 알린다.



보기 6- 68. IntensityGridSeries 노드. SizeX = 5, SizeY = 7.

노드 거리는 다음과 같이 자동 계산 된다

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

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

5.16.1 강도 그리드 데이터 설정

- **RangeMinX** 및 **RangeMaxX** 속성을 이용해 X 범위를 설정하라. 할당된 X 축의 최소 및 최대 값을 정리하기 위함이다.
- **RangeMinY** 및 **RangeMaxY** 속성을 이용해 Y 범위를 설정하라. 할당된 X 축의 최소 및 최대 값을 정리하기 위함이다.
- **SizeX 및 SizeY** 속성들을 설정해 그리드를 행과 열로 크기를 주어라.
- 각 노드의 값을 설정하라:

메소드, 데이터 배열 인덱스로

```
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
```

다른 방식, SetDataValue 이용

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

```
}
gridSeries.InvalidateData(); //Notify new values are ready and to refres
```

존재하는 그리에만 값 설정

데이터는 빠르게 변하는 와중 IntensityMesh 의 형상, 또는 IntensityGrid 시리즈의 SizeX 또는 SizeY 가 변하지 않을 때 **SetValuesData** 메소드를 사용하는 것이 가장 효율적일 것이다. Double[][] 형식의 배열을 받기에 스크롤링 또는 열과 행을 재배치 하는 것이 빠르다. 특히 **PixelRendering** 속성 (5.16.4 를 보세요)과 함께 썼을 때 이는 고해상도 스크롤링 스펙토그램 시각화에 아주 효과적인 접근 방식이다. **PixelRendering** 이 SetValuesData 로 설정된 외부 데이터 배열 세트로 비활성화 되었을 때 **Data** 속성은 무효할 수 없다.

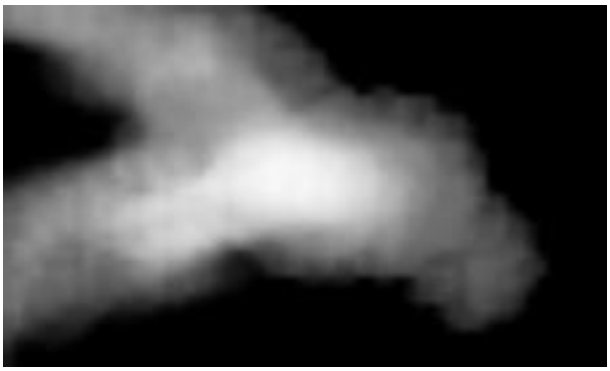
존재하는 그리에만 색 설정

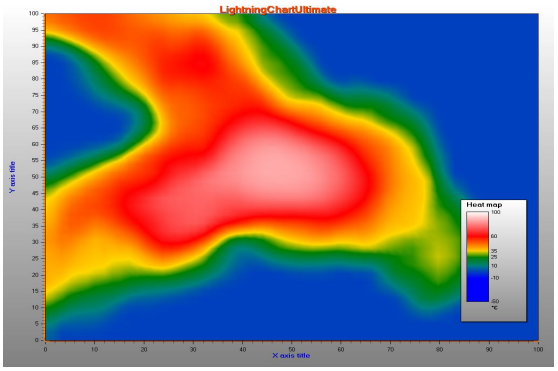
데이터는 빠르게 변하는 와중 IntensityMesh 의 형상, 또는 IntensityGrid 시리즈의 SizeX 또는 SizeY 가 변하지 않을 때 **SetColorsData** 메소드를 사용하는 것이 가장 효율적일 것이다. 이는 int[][] 형식의 값을 받는다, 예) GPU 가 직접 받는 ARGB 값들. 이런 형식의 데이터 배열로 스크롤링 또는 열과 행의 재 배치하는 것은 빠르다. 특히 데이터는 빠르게 변하는 와중 IntensityMesh 의 형상, 또는 IntensityGrid 시리즈의 SizeX 또는 SizeY 가 변하지 않을 때 **SetValuesData** 메소드를 사용하는 것이 가장 효율적일 것이다. **PixelRendering** 이로 SetColorsData 설정된 외부 데이터 배열 세트로 비활성화 되었을 때 **Data** 속성은 무효할 수 없다.

5.16.2 비트맵 파일에서 강도 그리드 데이터 생성

데모 예시: 열지도

비트맵 이미지에서 표면을 생성하라. **SetHeightDataFromBitmap** 메소드를 이용하면 가능하다. 시리즈의 **Data** 배열 속성은 비트맵 크기의 크기를 갖게 된다 (앤티 앨리어싱 또는 재샘플링이 사용되지 않았을 때). 각 비트맵 이미지 픽셀의 빨강, 초록 및 파랑 값들이 더해진다. 합이 클수로 그 노드의 데이터 값이





더 커진다. 검정 및 어두운 색들은 낮은 값들이 주어지고 흰색들이 높은 값들을 갖게 된다.

보기 6-69. 출처 비트맵 및 계산된 강도 값 데이터. 어두운 값들은 낮고 밝은 값들은 높은 값들이 주어진다.

5.16.3 채우기 스타일

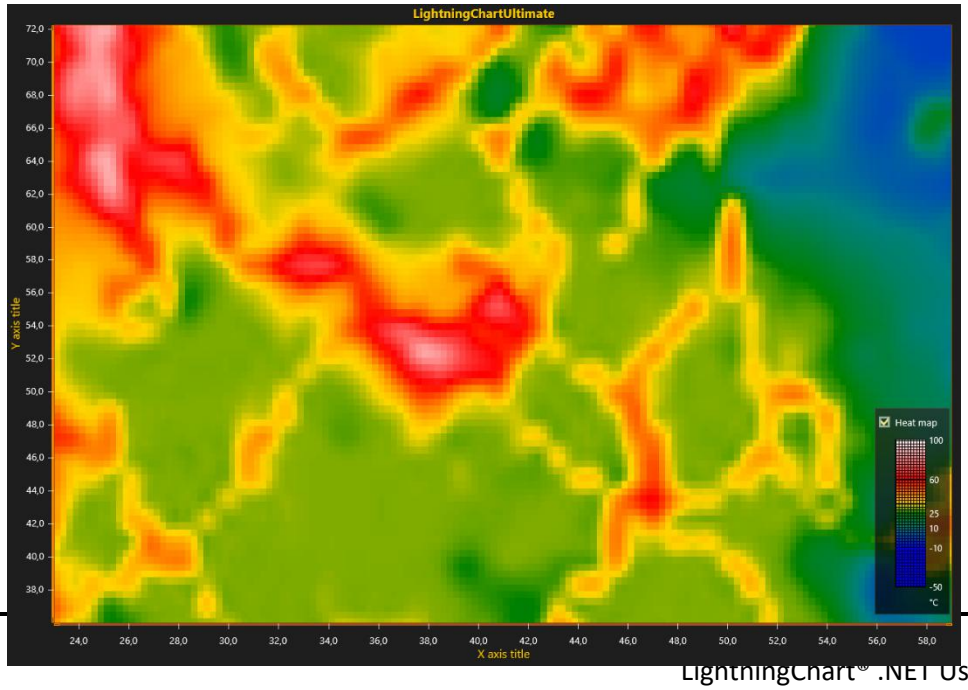
Fill 속성으로 채우기 스타일을 선택하라. 다음과 같은 옵션들에서 선택할 수 있다

- **None**: 이 옵션을 선택하면 아무런 채움이 적용되지 않는다. 와이어프레임 메쉬 또는 일반 등고선에 적합하다.
- **FromSurfacePoints**: Data 속성 노드의 색들이 사용된다.
- **Toned**: ToneColor 적용된다
- **Paletted**: 제 6.16.5 장을 보세요.

FullInterpolation 속성을 활성화 하여 채우기에 고급 보간 메소드를 사용하라. CPU 및 GPU 사용량이 많아진 다는 점을 주의하라. 완전 보간을 이용함으로써 채우기 품질이 나아지지만 데이터 배열 사이즈가 작아야만 보인다.

5.16.4 픽셀 지도로 렌더링

PixelRendering 속성을 활성화 함으로 노드들이 픽셀 또는 직사각형으로 렌더링 된다. 이는 괸장이 고성능 렌더링 스타일이다, 예) 실시간 고화질 열 이미징 응용 프로그램. 이 렌더링 모드가 선택 되었을 때 등고선, 와이어 프레임 및 보간 등 많은 기타 옵션들이 비활성화 된다는 점을 주의하라. 로그 축들이 사용 되면 로그 변형은 시리즈의 모서리에만 적용된다. 비트맵 내 픽셀은 일정한 간격을 유지하고 아무런 로그 변형이 적용 되지 않는다.



보기 6- 70. PixelRendering = true.

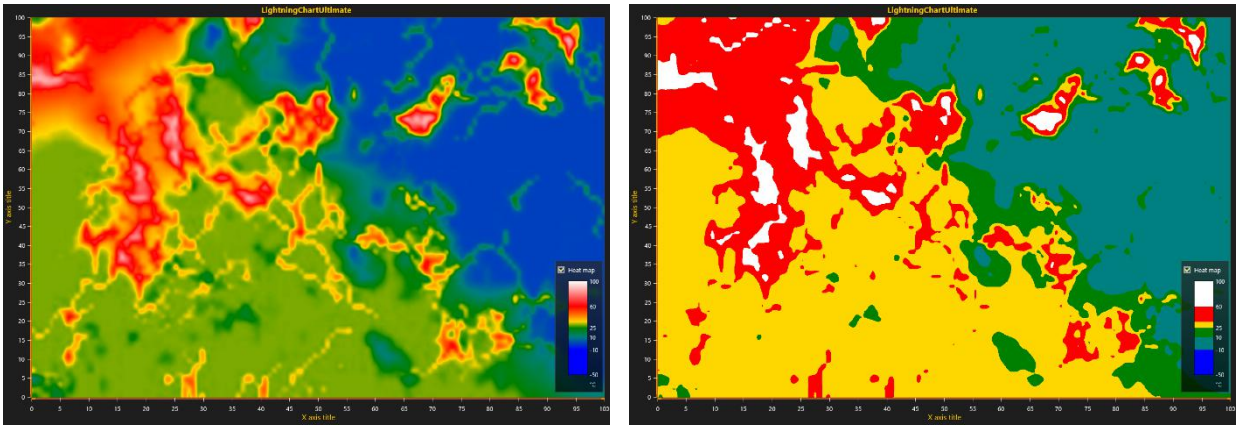
5.16.5 ValueRangePalette

ValueRangePalette 속성에 값 색칠을 위한 색 단계를 정의하라. **ValueRangePalette** 은 다음의 용도로 사용 가능하다

- **채우기** (제 6.16.3 장을 보세요)
- **와이어프레임** (제 6.16.6 장을 보세요)
- **등고선** (제 6.16.7 장을 보세요)

등고선 팔레뜨를 위한 몇 단계를 정의하라. 각 단계는 높이 값과 해당 색을 갖고 있다.

주의! 20 개의 단계는 사전 컴파일 되었고 빠르게 로딩 된다. 단계 수가 많아질수록 차트를 초기화할 때 몇 초간의 딜레이를 예상할 수 있다.



보기 6-71. 좌, IntensityGridSeries 채우기가 Paletted 로 설정 및 팔레뜨 종류는 그라디언트로 설정. 우, 팔레뜨 종류가 일정으로 설정

팔레뜨는 **MinValue**, **Type** 및 **Steps** 속성들로 정의 된다. **Type** 에는 두 선택이 있다: **Uniform** 및 **Gradient.** 이전 값들의 등고선 팔레뜨 (레전드 상자 확인)는 다음과 같이 보여준다:

- **MinValue**: -50
- **Type**: Uniform
- 단계:
    - 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

첫 단계 값 밑의 값들은 첫 단계의 색으로 칠해졌다.

5.16.6 와이어프레임

WireframeType 를 이용해 와이어 프레임 스타일을 선택하라. 선택 가능 옵션은 다음과 같아:

- **None**: 와이어 프레임 없음
- **Wireframe**: 단색의 와이어 프레임. **WireframeLineStyle.Color** 을 이용해 색 설정하라.
- **WireframePaletted**: 와이어 프레임 색은 **ValueRangePalette** 을 따른다 (6.16.5 보세요)
- **WireframeSourcePointColored**: 와이어 프레임의 색이 그리드 노드의 색을 따른다
- **Dots**: 단색의 점이 그리드 노드 위치에 그려진다
- **DotsPaletted**: 그리드 노드 위치에 점들이 그려지고 **ValueRangePalette** 대로 색 칠해진다
- **DotsSourcePointColored**: 그리드 노드 위치에 점들이 그려지고 색은 그리드 노드를 따른다

와이어 프레임 선 스타일 (색, 넓이, 패턴)은 **WireframeLineStyle** 이용해 수정 가능하다.

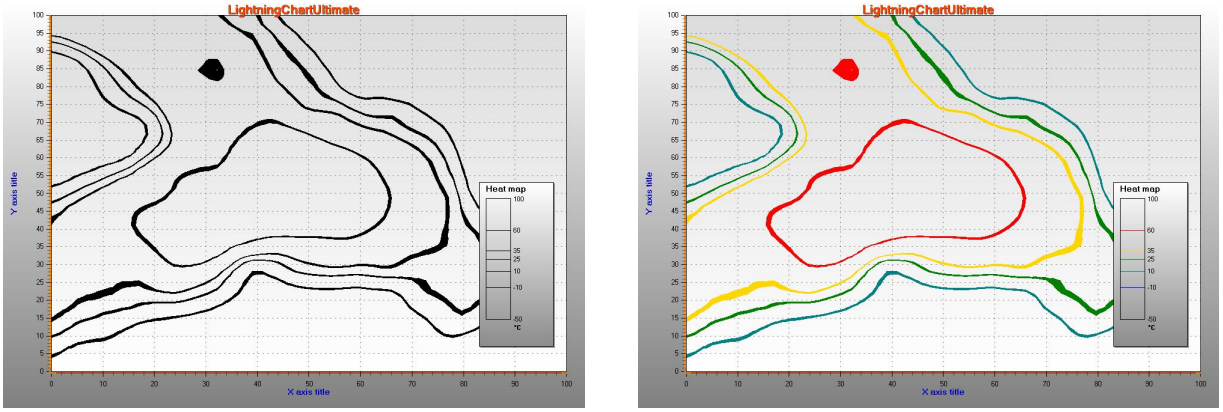
주 의 ! 팔 레 뜨 로 칠 해 진 와 이 어 프 레 임 라 인 및 점 은 **WireframeLineStyle.Width = 1 이 고 WireframeLineStyle.Pattern = Solid** 로 설정이 되었을 때에만 사용 가능하다.

5.16.7 등고선

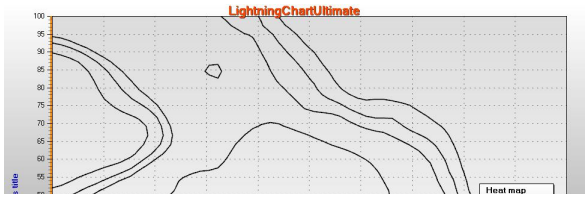
데모 예시: 열지도 색 스프레드; 라벨 있는 등고선

등고선은 채우기와 와이어프레임 속성과 같이 사용 가능하다. **ContourLineType** 속성을 설정함으로 등고선을 여러 스타일대로 그릴 수 있다:

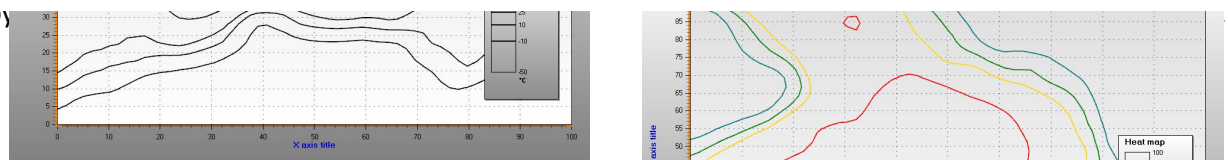
- **None**: 아무 등고선도 보여지지 않는다
- **FastColorZones**: 팔레뜨 단계 끝의 얇은 구역들로 선들이 얇게 그려진다. 계속되서 업데이트 또는 생기있는 표면에 아주 적합한 아주 강력한 렌더링을 허용한다. 가파른 값 변화는 얇은 선으로 표현되고 천천한 높이 변화는 두꺼운 선으로 표현된다. 각 라인은 **ContourLineStyle.Color** 속성으로 정의된 색을 사용한다. 구역 넓이는 **FastContourZoneRange** 속성으로 설정 가능하다. 값은 Y 축 범위 내 있다.
- **FastPalettedZones**: **FastColorZones** 와 같지만 라인 색칠 옵션은 **ValueRangePalette (6.16.5 장을 보세요)** 옵션을 따른다.
- **ColorLine**: **FastColorZones** 와 같지만 등고선이 실제 선이다. 렌더링이 더 오래 걸리며 계속되서 업데이트 또는 생기있는 표면에는 적합하지 못한다. 라인 넓이는 **ContourLineStyle.Width** 속성으로 수정 가능하다.
- **PalettedLine**: **ColorLine** 와 같지만 라인 색칠은 **ValueRangePalette** 옵션들을 따른다.



보기 5- 71. 좌, ContourLineType = FastColorZones. 우, ContourLineType = FastPalettedZones.



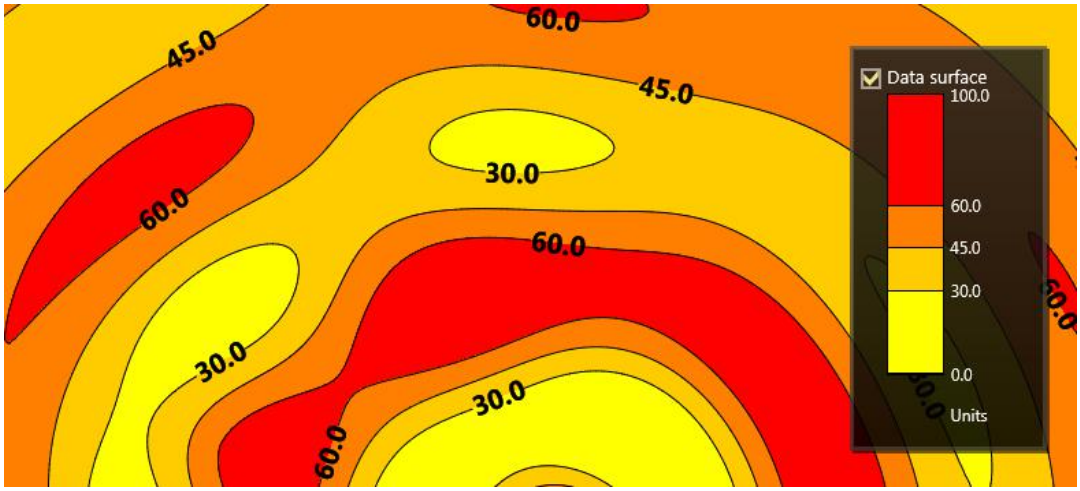
보기 6- 72. 좌, ContourLineType = ColorLine. 우, ContourLineType = PalettedLine



5.16.8 등고선 라벨

등고선들이 보일 때 라인 길 내 숫자 값을 보이게 설정 가능하다.

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



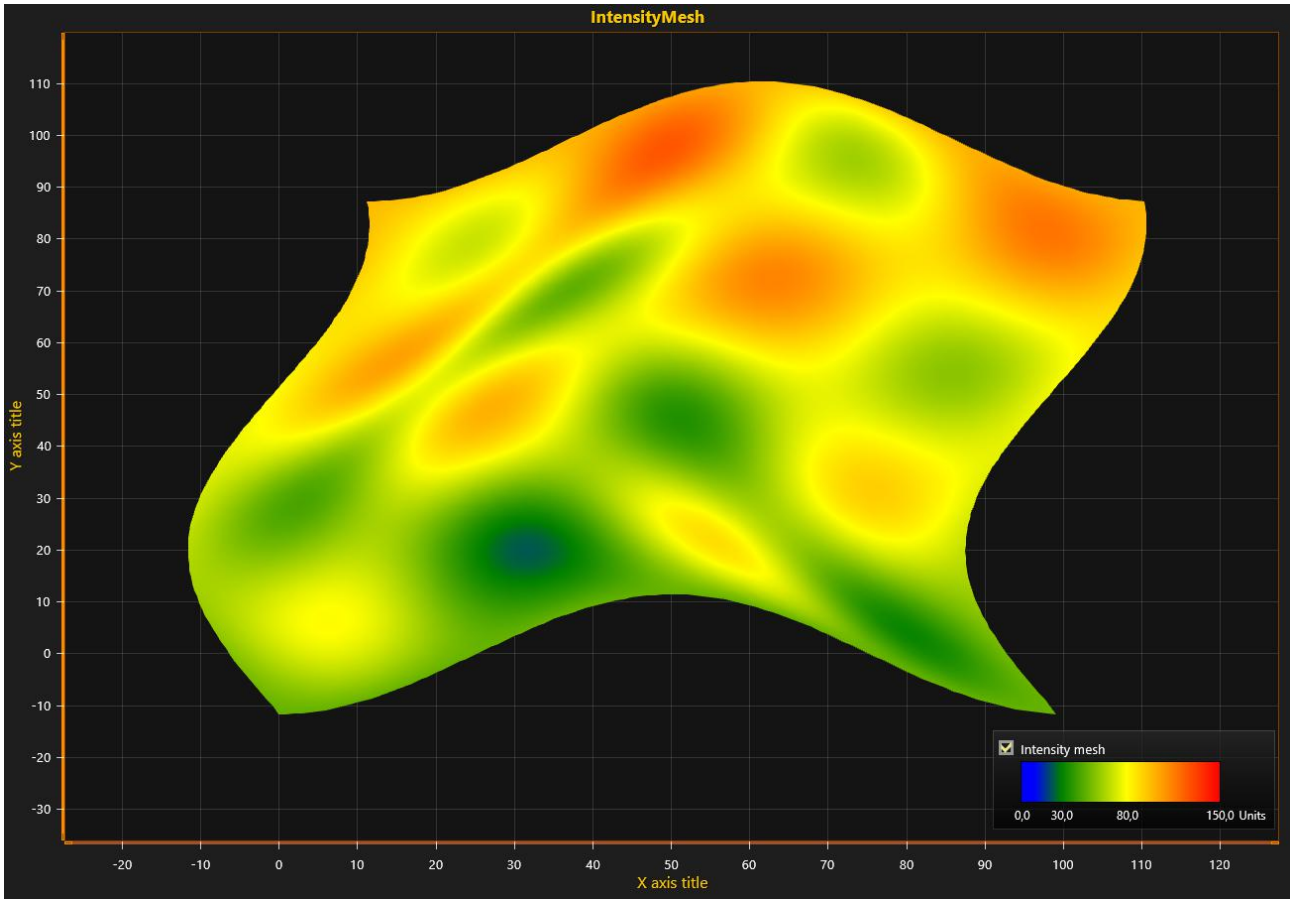
보기 6-73. ContourLineLabels 의 속성 및 결과

LabelsNumberFormat 를 예를 들어 소수점 수 설정 등의 커스텀 스트링 서식을 위해 사용하라.

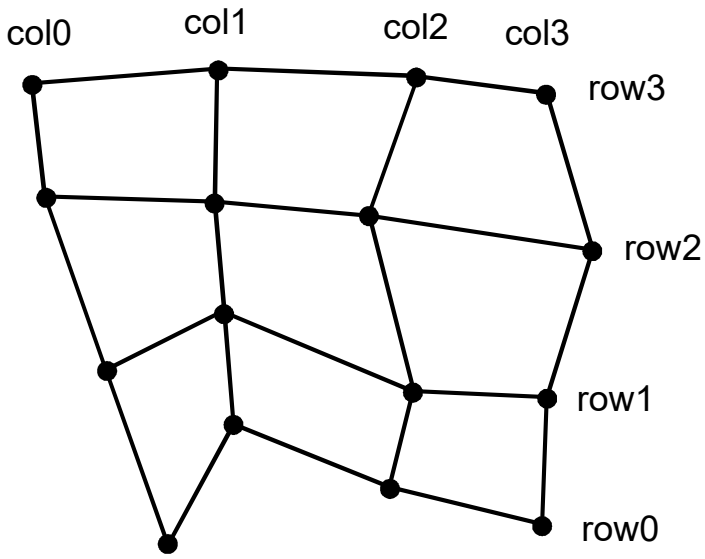
5.17 IntensityMeshSeries

데모 예시: 애니메이티드 강도 메쉬; 강고 메쉬, 정적 형상; 강도 메쉬, 원형/극선 형상

IntensityMeshSeries 는 **IntensityGridSeries** 와 거의 흡사하다. 가장 큰 차이점은 시리즈 노드가 X-Y 공간에 임의로 위치 되어있을 수 있다는 점이다. 다시 말해 시리즈가 직사각형이지 아니어도 된다는 것이다. 와이어프레임 라인은 **WireframeType** 속성으로 보이게 설정 가능하고 노드도 **ShowNodes** 를 참으로 설정해 보이게 할 수 있다.



보기 6-74. 각 노드마다 자유 위치된 X 및 Y 값들의 IntensityMeshSeries. WireframeType = Wireframe and ShowNodes = true.



. 보기 6-75. 강도 메쉬 노드 SizeX = 4, SizeY =4.

5.17.1 형상 변형 때 강도 메쉬 데이터 설정

X, **Y** 및 **Value** 필드들이 한꺼번에 없데이트 됐을 때 다음과 같은 설명을 따라라.

- **SizeX** 및 **SizeY** 속성들을 설정해 메쉬를 열과 행 기반 크기를 주어라.
- 모든 노드에 X Y 와 값을 설정하라:

데이터 배열 인덱스를 사용한 방법

```
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
```

SetDataValue 를 사용한 다른 방법

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

5.17.2 형상이 변하지 않을 때 강도 메쉬 데이터 설정

Data 배열 **IntensityPoint** 구성의 **Value** 필드만 업데이트 되었을 때 다음 설명을 따르라. 각 노드의 X 및 Y 값이 같은 위치에서 움직이지 않을 때 데이터를 업데이트 하는 성능 최적화된 방식이다. 예를 들어 열 화상 또는 환경 데이터 모니터링 솔루션 등이 있다.

5.17.2.1 시리즈 생성 및 그의 형상

- **Optimization** 을 **DynamicValuesData** 로 설정하라.
- **SizeX 및 SizeY** 속성들을 설정해 메쉬를 열과 행의 크기를 주어라.
- 모든 노드를 위해 X, Y 와 Value 를 설정하라:

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

5.17.2.2 주기적으로 값을 업데이트 하기

- 모든 노드의 값만 설정하라:

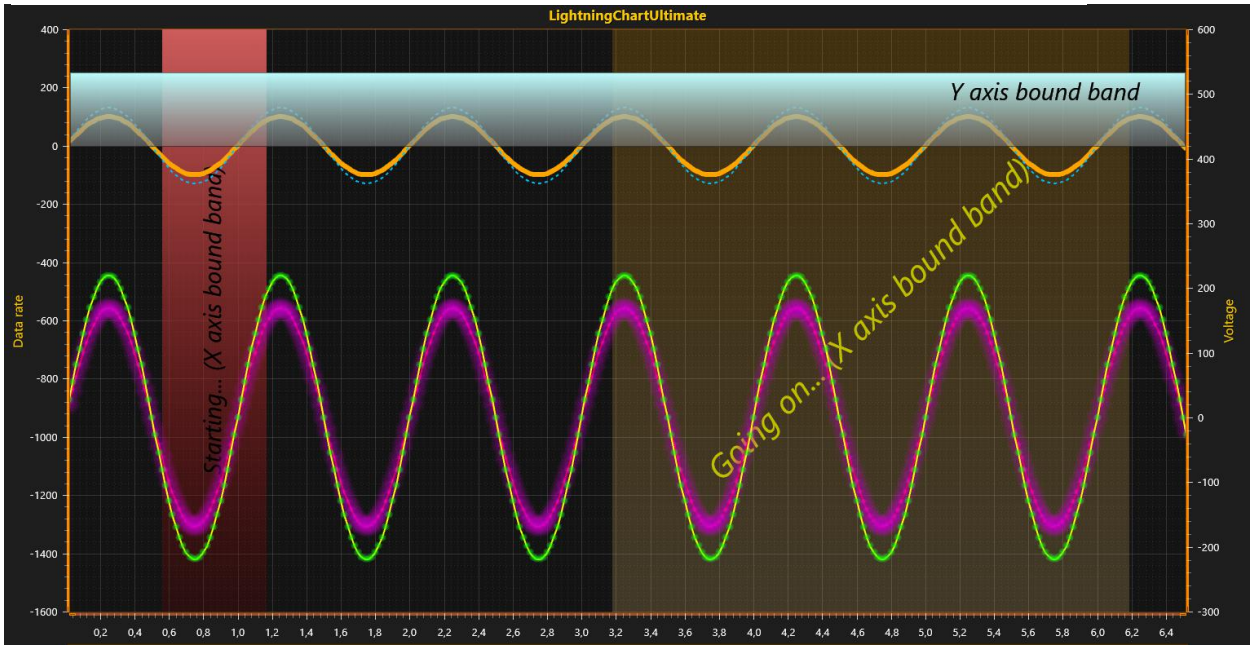
```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)
{
     for (int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++)
     {
         meshSeries.Data[nodeIndexX, nodeIndexY].Value = value;
     }
}
meshSeries.InvalidateValuesDataOnly(); //Only data values are updated
```

5.18 밴드

데모 예시: 밴드; 통계 분석; 긴 데이터 분석; 줌 막대 그래프

밴드는 시리즈로 취급 가능하다. 다른 시리즈들과 같은 유저 인터페이스 동작들을 갖고 있지만 한 밴드 시리즈는 오직 한 밴드만으로 구성 되었다. 밴드는 한 마진에서 반대 쪽까지 뻗는 수직 또는 수평 구역이다. **Binding** 속성을 이용해 밴드를 Y 또는 X 축에 묶을 수 있다. 밴드가 Y 축에 묶였을 시, **AssignYAxisIndex** 속성도 설정 해야 한다. X 축에 시리즈가 묶여 있다면 **AssignYAxisIndex** 속성을 무시하거나 -1 로 설정하라.

보기 6-76. 라인 시리즈와 있는 밴드 두개.

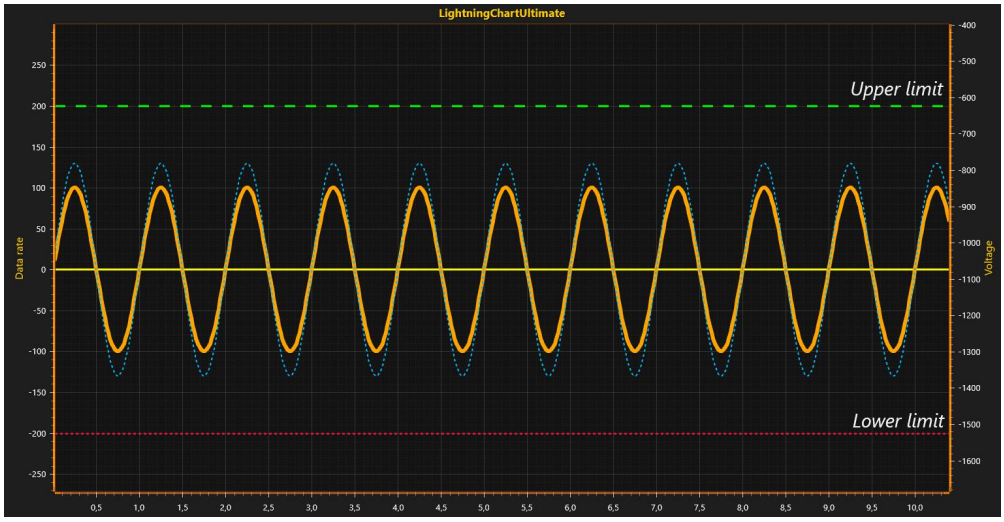


밴드가 라인 또는 막대 시리즈 뒤에 있어야 한다면 **Behind** 속성을 참으로 설정하라. 밴드 가장자리는 **ValueBegin** 및 **ValueEnd** 속성들로 설정 된다. 묶여진 축의 값들이다. 밴드를 마우스로 다른 위치로 드래그 가능하다. 가장자리에서 드래그로 밴드 크기를 재설정 하라. 이는 드래그 된 가장자리 값 (**ValueBegin 또는 ValueEnd)**을 업데이트 한다.

5.19 일정 라인

데모 예시: 오실로스코프; 리사쥬 모니터; 시그널 리더; 구역; 스플리터 있는 세그먼트,

밴드와 같이 일정 라인들도 시리즈로 취급 가능하다. 일정 라인들은 y 축에 묶여 있으며 그래프 좌쪽 가장자리에서 우측 가장자리로 뻗는 한 수평선을 가리킨다. **Value** 속성으로 레벨을 설정하라. 일정 라인은 마우스 드래그로 수직 방향으로 움직일 수 있다. **Behind** 속성을 참으로 설정하면 일정 라인이 라인 및 막대 시리즈 뒤로 그려진다. 이 설정 없이는 기타 그래프 앞에 그려진다.



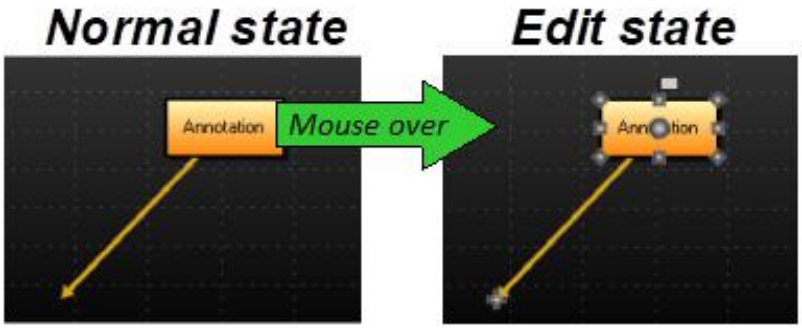
보기 6-77. 사인 라인 시리즈 주위 일정 라인.

5.20 주석

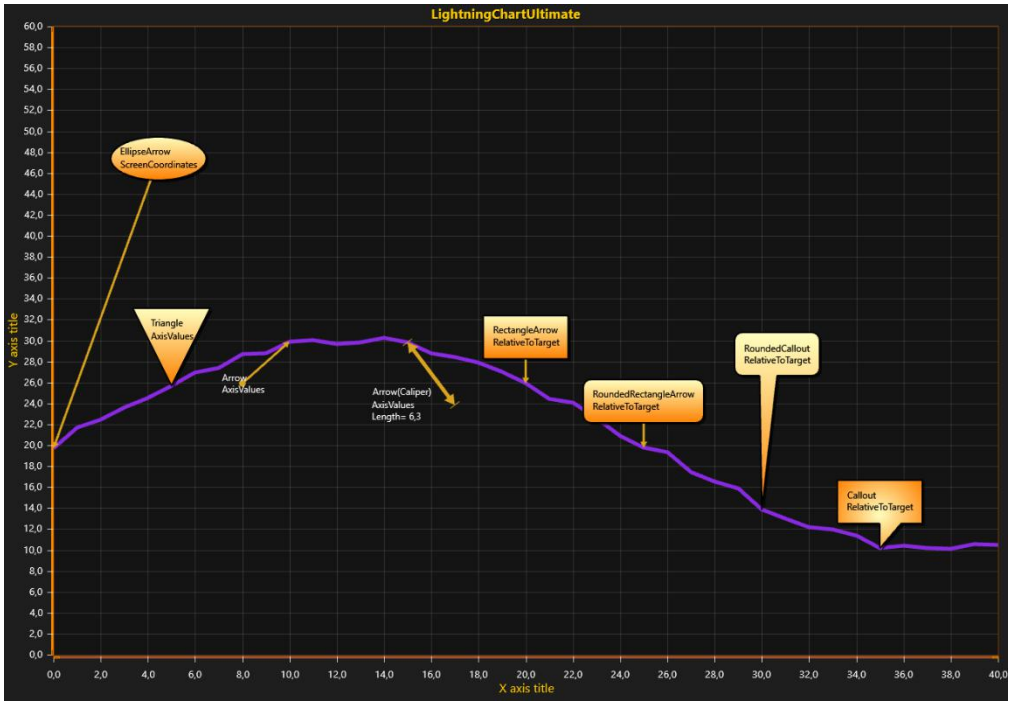
데모 예시: 주석; 커스텀 렌더링; 강도 그리드 마우스 제어; 멀티 채널 커서 트래킹; 스톡 및 막대; 주석 표

주석은 차트 어디든 마우스 대화식 텍스트 라벨을 또는 그래픽을 보이는 것을 허용한다. 주석은 마우스로 이동, 사이즈 재설정, 회전, 타겟 및 위치 등 설정 가능하다. 또는 코드로 제어 가능하다. 주석은 커스텀 그래픽 렌더링을 해야할 때 좋은 도구다. 이는 다른 스타일 및 모양으로 렌더링 가능하기 때문이다. **ViewXY.Annotations** 컬렉션에 **AnnotationXY** 객체들을 생성하라.

주석 위에 마우스를 올리면 마우스 대화식 수정 상태가 된다. 이는 주석의 위치, 크기, 회전 및 화살표 방향 재설정을 허용한다.



보기 5-78. 마우스를 주석 위에 올려 수정 상태를 만든다. 마우스를 다른곳으로 움직이면 수정 상태를 종료하게 된다.



보기 5-79. 라인 시리즈 주변 여러 스타일의 AnnotationXY 객체.

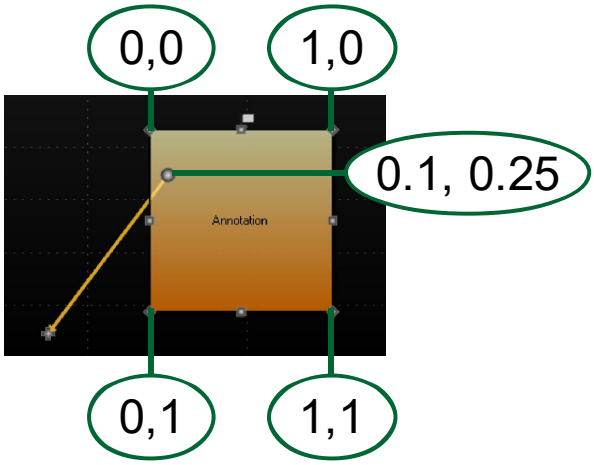
5.20.1 목표 및 위치 제어

Target 이란 화살표의 끝부분, 화살표 또는 콜아웃 팁이 가리키는 곳을 얘기한다. **Target** 은 축 값 또는 화면 좌표로 설정 가능하다. **TargetCoordinateSystem** 를 사용해 **AxisValues** 또는 **ScreenCoordinates** 사이를 선택

하라. **AxisValues** 가 선택 되었을 때 **TargetAxisValues** 속성이 화살표 라인이 가리키는 곳을 설정한다 (화살표의 끝부분). 화면 좌표를 사용하고 싶다면 **TargetScreenCoords** 를 사용하라.

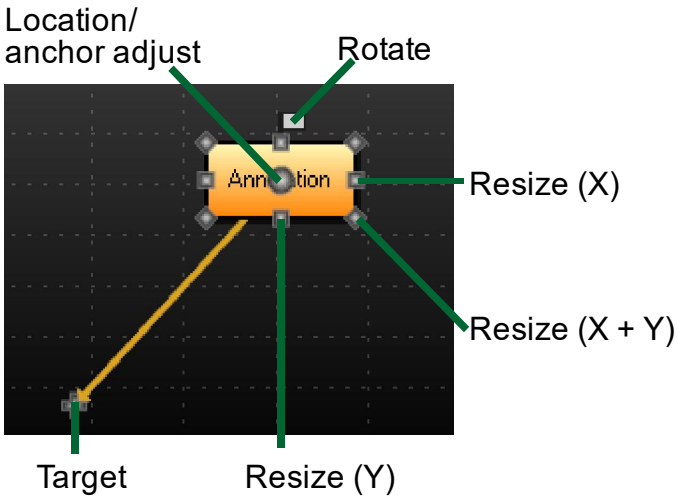
Location 은 화살표의 시작점이다. 이는 화면 좌표, 축 값 또는 **Target** 에서 부터 상대 오프셋으로 설정 가능하다. **LocationCoordinateSystem** 를 이용해 선택하고 선택 방식으로 위치를 **LocationScreenCoords**, **LocationAxisValues** 또는 **LocationRelativeOffset** 로 제어하라. **Location** 은 또 텍스트 구역 회전의 중심점이 된다.

Anchor 속성은 텍스트가 **Location** 에 어떻게 놓여지는지 제어한다. **Anchor.X** = 0.5 및 **Anchor.Y** = 0.5 으로 설정함으로 화살표의 시작이 중간이 된다. **Anchor.X** 0.1 및 **Anchor.Y** = 0.25 로 설정 할 시 다음 보기와 같이 화살표의 시작이 왼쪽 위 근처로 설정 된다:



보기 6-78. Anchor 값 설명. 현재 Anchor.X = 0.1 and Anchor.Y = 0.25. 앵커 값이 0 과 1 사이일 때 화살표의 시작점은 텍스트 구역이 된다.

5.20.2 마우스를 이용해 회전, 크기 및 위치 재설정



보기 6-79. 주석 마우스 대화식 노드.

화살표 끝을 움직이기 위해 **Target** 에서 드래그를 하라. 신규 **Location** 을 설정하기 위해 텍스트 구역에서 드래그를 하라. 라운드 위치/앵커 노드에서 드래하면 동시에 **Anchor** 및 **Location** 속성들이 재설정된다. 이때 텍스트 상자의 위치는 변하지 않는다.

X 또는 Y 사이즈 재설정 노드에서 드래그 중 **Shift** 키를 누르고 있으면 대칭 조작이 사용 가능하다. 동시에 양쪽이 수정 가능하다. 구석 재설정 노드 (X+Y)에서 드래그 중 **Shift** 키를 누르면 종횡비가 유지 된다. 회전 조작에서 **Shift** 키는 15 도의 배수의 각도로 회전 시킨다.

5.20.3 외관 수정

Style 속성을 설정해 주석 모양을 선택하라. 선택 가능한 옵션은: **Rectangle**, **RectangleArrow**, **RoundedRectangle**, **RoundedRectangleArrow**, **Arrow**, **Callout**, **RoundedCallout**, **Ellipse**, **EllipseArrow**, **Triangle** 및 **TriangleArrow** 이 있다.

화살표 있는 스타일들은 **ArrowLineStyle**, **ArrowStyleBegin** 및 **ArrowStyleEnd** 을 사용해 화살표 디자인을 선택하라. 화살표 끝 스타일은 다음과 같은 옵션들이 있다: **None**, **Square**, **Arrow**, **Circle** 및 **Caliper**.

Fill 을 사용해 주석의 채우기 설정을 수정하라. 마우스 대화식 노드 수정 상태의 외관은 **NibStyle** 에선 변경 가능하다. **TextStyle** 은 글꼴 설정 및 텍스트 구역 안 텍스트 정렬을 제어한다. **BorderLineStyle** 과 **CornerRoundRadius** 는 태두리 선의 생김새를 제어한다.

5.20.4 크기 설정

Sizing 속성은 주석 텍스트 상자의 크기를 제어한다:

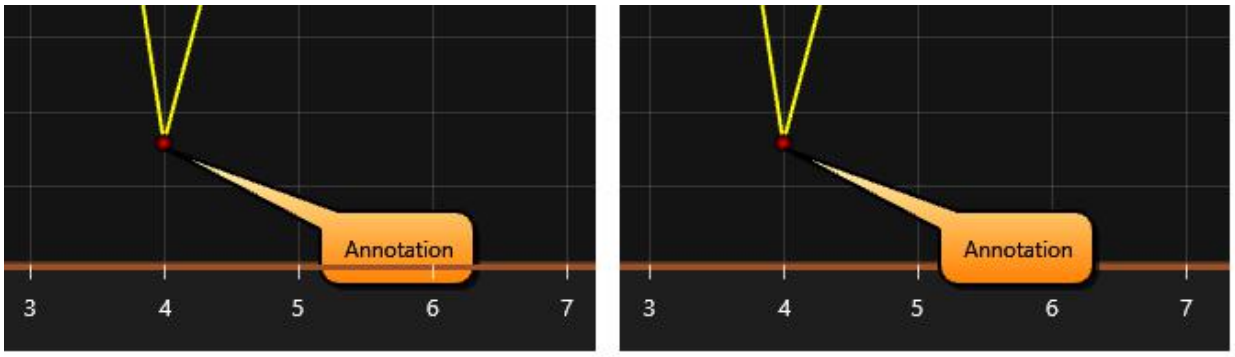
- **Automatic** 은 내용대로 크기를 조정하고 **AutoSizePadding** 간격을 태두리에 둔다.
- **AxisValuesBoundaries** 는 주석의 크기를 축 값대로 설정하는 것을 허용한다. **AxisValuesBoundaries.XMin**, **XMax**, **YMin** 및 **YMax** 를 사용해 정의하라.
- **ScreenCoordinates** 는 화면 좌표 대로 크기 설정 가능하다. **SizeScreenCoords.Height** 및 **Width** 를 사용하라.

5.20.5 텍스트 구역이 보이게 유지 하기

KeepVisible 가 활성화 되었을 때 주석 텍스트 구역은 그래프 안에 강제로 집어 넣어 진다. 마우스 또는 코드로 그래프 밖으로 주석 이동 불가능 하다. 그래프 뷰를 움직이거나 축을 수정할 때에도 주석은 그래프 안에 보이기로 위치가 재설정 된다.

5.20.6 축 위로 주석 표시

RenderBehindAxes = True 를 설정함으로 주석이축 밑으로 보여진다. 클리핑 및 z 순서 특징은 이 경우에 실행 되지 않는다. **ClipInsideGraph** 가 참으로 설정 되었을 때 **RenderBehindAxis** 는 효과가 없다.



보기 6-80. 좌, RenderBehindAxes = True. 우, RenderBehindAxes = False. 두 경우 다 ClipInsideGraph 는 False 로 설정 되었다.

5.20.7 그래프 내 클리핑

ClipInsideGraph 가 활성화 되었을 때 주석은 그래프 내에서 클리핑 된다. 비활성화 되었을 때 주석은 차트의 마진 구역 내에서도 렌더링이 된다.

ClipWhenSweeping 을 활성화 함으로써 **ScrollMode** 가 **Sweeping** 으로 설정 되었을 때 주석이 스위핑 갭 구역에 나타나지 않는다.

5.20.8 Z 순서 제어

Behind 속성을 기본 값 **False**로 설정함으로 주석은 시리즈 맨 위에 나타난다. **True** 으로 설정함으로 시리즈 이전 렌더링이 되어 아래에 나타난다.

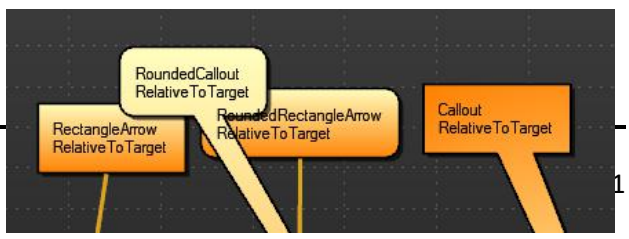
주석은 Annotations 목록에 존재하는 순서대로 나타난다. **Behind** 를 필터 마스터 컨트롤러로 유지한다. 주석의 Z 순서는 **ChangeOrder** 주석 메소드를 이용하여 빠르게 변경 할 수 있다. 예를 들어 마우스 이벤트 핸들러 내에서 빠르게 변경 할 수 있다. 순서 변경 옵션은 다음과 같다:

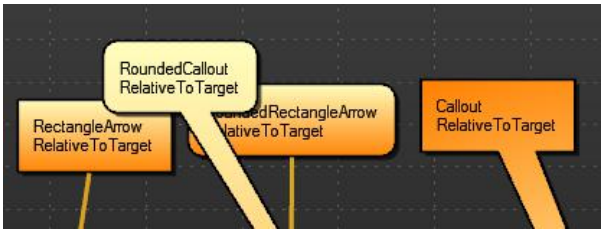
- **BringToFront** 는 주석을 맨 위로 올린다
- **SendToBack** 뒤로 보낸다
- **MoveBack** 한 순서 뒤로 움직인다
- **MoveFront** 한 순서 앞으로 움직인다

5.20.9 LayerGrouping 성능 최적화

수 백개의 텍스트가 보이는 주석이 있을 때 텍스트 렌더링 딜레이가 큰 혁할을 갖기 시작한다. 기본적으로 텍스트 렌더링은 Z 순서를 따라 텍스트를 주석 내에서 벗어나지 못하게 지킨다.

LayerGrouping = **True** 로 설정함으로 차트를 두개의 납작 주석 텍스트 층만을 이용하게 하여 성능을 향상 시킬 수 있다. 하나는 **Behind** 가 **True** 로 설정 된 주석을 위함, 하나는 **Behind** 가 **False** 로 설정 된 주석을 위함이다. 이는 성능을 굉장히 향상 시킨다. 반면에 주석이 서로를 겹칠 경우 텍스트 렌더링에 문제가 생긴다.





보기 6-81. 좌, LayerGrouping = False. 우, LayerGrouping = True. 텍스트의 Z 순서를 잃었다.

Style = Arrow 를 사용하거나 또는 주석의 채우기를 보이지 않게 설정 했을 때 Z 순서의 제한은 대부분 티가 나지 않는다.

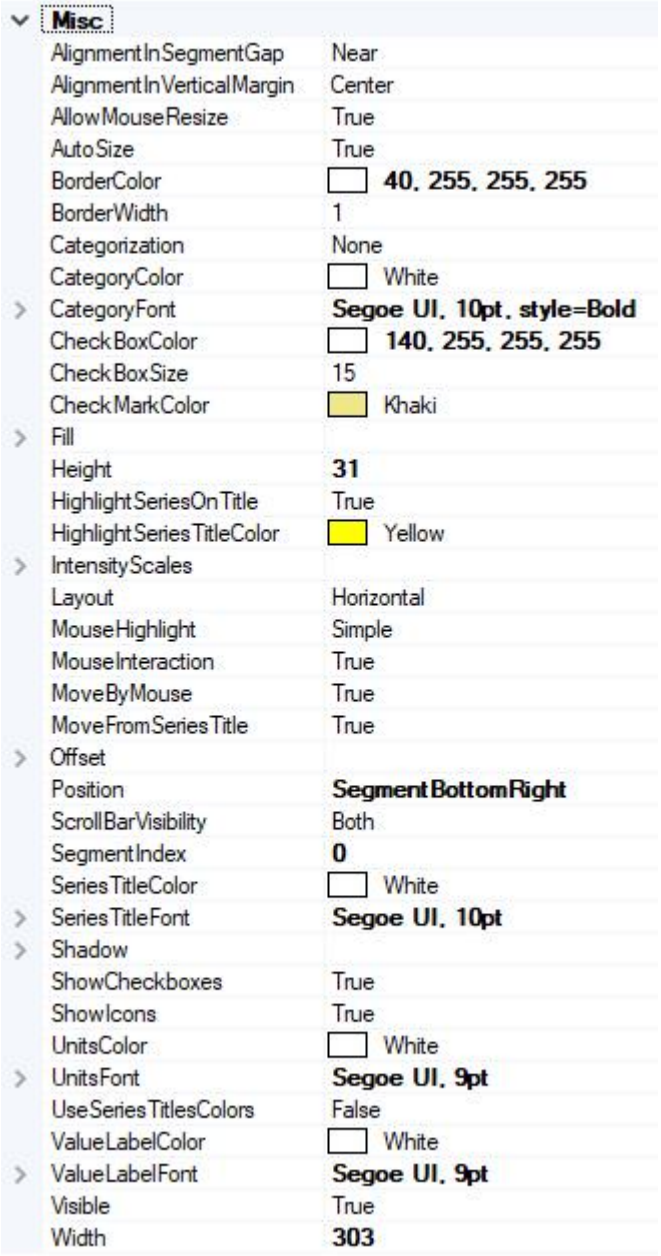
5.20.10 축 값 및 화면 좌표 사이 변경

어느 경우에는 **Location** 또는 **Target** 을 섞인 구성으로 정의 하고 싶을 때가 있을 것이다. X 는 화면 좌표로와 Y 는 축 값 또는 그 반대로의 경우 등이 있다. 6.2.12 에 설명된 대로 축은 축 값을 화면 좌표로 변경하기 위해 **ValueToCoord** 메소드가 있고 화면 좌표를 축 값으로 변경하기 위해 **CoordToValue** 가 있다.

5.21 레전드 상자

데모 예시: 여러 레전드; 열지도 레전드; 스플리터 있는 세그먼트

v.8 부터 ViewXY 는 같은 그래프 내 여러 레전드 상자를 지원한다. **ViewXY.LegendBoxes** 컬렉션에 이 레전드 상자들을 삽입하라.



보기 6-82. 광범위한 LegendBoxXY 속성 트리.

5.21.1 레전드 상자에서 시리즈 숨기기/보이기

보기 6-83. 시리즈 제목 및 아이콘을 보여주는 레전드 상자. 시리즈 옆 박스에 티크를 없애 시리즈를 숨겨라.

5.21.2 레전드 상자에서 시리즈 리스팅 방지

만약 어느 시리즈를 레전드 상자에 리스팅 하길 원치 않을 시 해당 시리즈를 **series.ShowInLegendBox = False** 로 설정하라.

5.21.3 특유 시리즈를 보이게 어느 레전드 상자에서 선택

series.LegendBoxIndex 를 사용해 선호하는 레전드 상자를 선택하라. 시리즈는 한 레전드 상자에만 나타날 수 있다. 모든 시리즈의 기본 인덱스 설정은 0 이다. 이는 다르게 설정하지 않는 한 모두 같은 레전드 상자에 나타난다.

5.21.4 어느 그래프 세그먼트에 레전드 상자 보이게 선택하기

SegmentIndex 를 사용해 레전드 상자를 어느 세그먼트에 보이게 할 지 제어하라. 이는 세그먼트 기반 **Position** 옵션에만 해당 된다.

5.21.5 체크 상자 숨기기

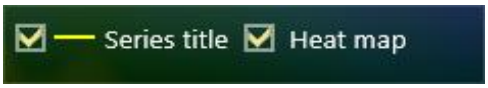
체크 상자를 숨기기 위해 **ShowCheckBoxes = False** 로 설정하라.

5.21.6 아이콘 숨기기

아이콘을 숨기기 위해 **ShowIcons = False** 로 설정하라.

5.21.7 강도 시리즈 팔레뜨 스케일 변경

IntensityGrid 또는 **-Mesh** 의 팔레뜨 스케일을 숨기기 위해 **IntensityScales.Visible = False** 로 설정 하라. 크기를 재설정 하기 위해 **ScaleSizeDim1** 및 **ScaleSizeDim2** 속성들을 설정 하라. 스케일의 태두리 및 제목의 위치도 변경 가능하다.



보기 6-84. 아래 사진에 LegendBox.IntensityScales.Visible = false 설정.

5.21.8 제어 위치

레전드 상자는 자동 또는 수동으로 놓을 수 있다. 자동 놓음은 그래프 세그먼트 또는 마진 위에 좌/상/우/하 정렬을 허용한다. 위치를 **Position** 속성으로 컨트롤 하라. 위치 옵션은: **TopCenter**, **TopLeft**, **TopRight**, **LeftCenter**, **RightCenter**, **BottomLeft**, **BottomCenter**, **BottomRight**, **Manual** 이 있다.

뷰가 여러 세그먼트로 나뉘어 졌다면 레전드 상자를 소속 세그먼트에 정렬 가능하다 (**SegmentIndex** 를 사용하라). 세그먼트 기반 컨트롤은 다음과 같은 옵션이 있다: **SegmentTopLeft**, **SegmentTopCenter**, **SegmentTopRight**, **SegmentBottomLeft**, **SegmentBottomCenter**, **SegmentBottomRight**, **SegmentLeftMarginCenter**, **SegmentRightMarginCenter**.

Offset 속성은 **Position** 속성에 제시된 위치로 부터 주어진 값 만큼 움직인다.

```
// Setting legend box position, offset shifts from RightCenter position
```

```
chart.ViewXY.LegendBoxes[0].Position = LegendBoxPositionXY.RightCenter;
chart.ViewXY.LegendBoxes[0].Offset = new PointIntXY(-15, -70);
```

Manual 위치 선정은 레전드 상자의 왼쪽 위로 부터 뷰의 왼쪽 위 구석까지의 오프셋을 계산한다. 이는 **TopLeft** 옵션과 다른 것을 주의 해야 한다. 이 옵션은 그래프 구역 위에서 부터의 오프셋을 계산하는 데에 쓰인다.

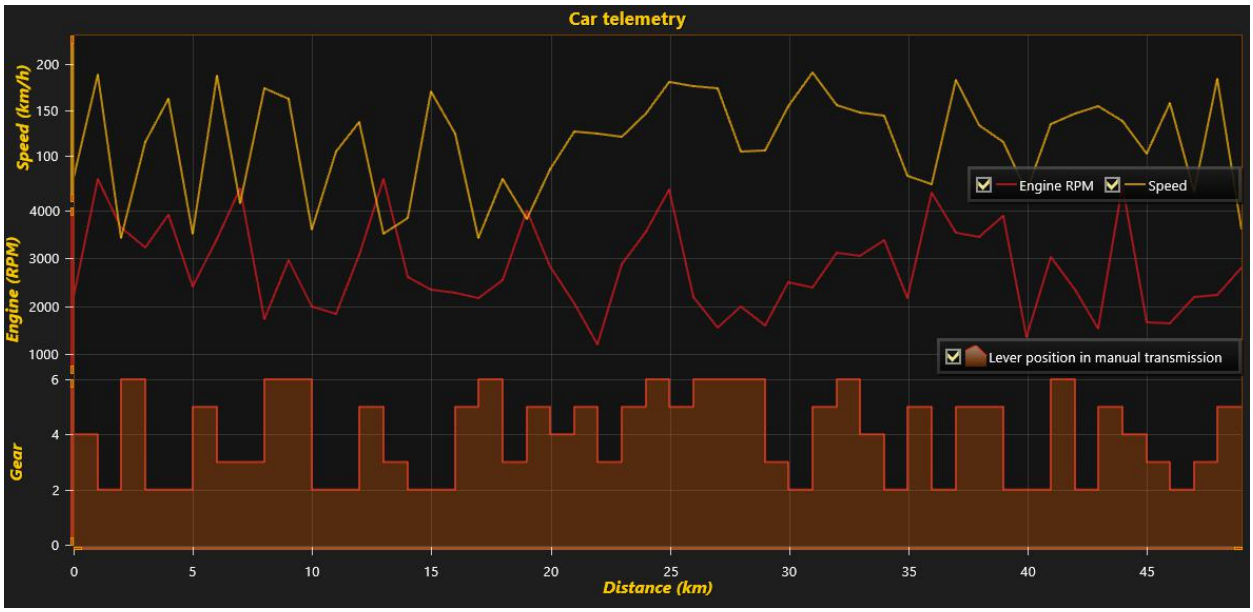
레전드 상자를 움직이거나 크기를 재설정할 때 이의 **Position** 은 **Manual** 로 설정 되고 **Offset** 속성이 새로운 위치를 반영하게 업데이트 됨을 주의하라.

자동 레전드 상자 정렬은 **Position** 을 **Manual** 이외의 옵션으로 돌리기 전까지 비활성화 된다. **Offset** 이 **Position** 옵션들 사이 변경할 때 업데이트가 안되니 레전드 상자가 가끔 사라지는 것처럼 보일 수 있다 (이는 뷰 밖에 위치 되어있다). **Offset** 을 0,0 으로 설정하여 고쳐라.

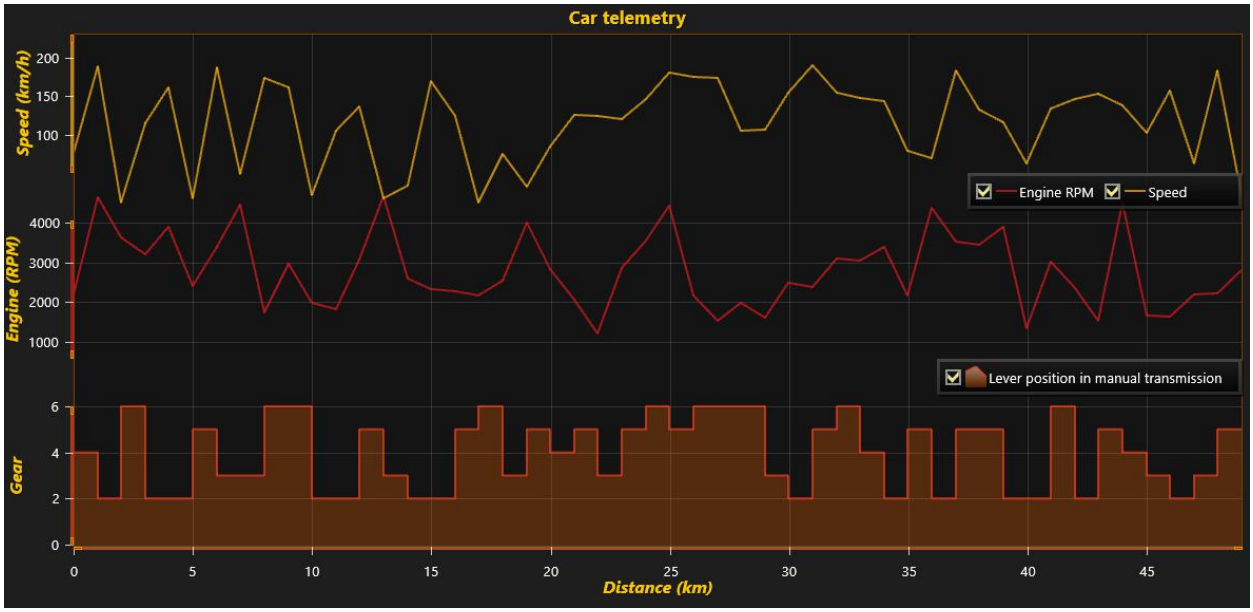
5.21.9 그래프 세그먼트 사이 레전드 상자를 위한 공간 할당

ViewXY.AutoSpaceLegendBoxes = True 로 설정 할 때에 세그먼트 사이 레전드 상자를 넣기 위한 추가 공간이 할당 된다.

ViewXY.AxisLayout.SegmentsGap 또한 세그먼트 사이에 할당 됨을 주의하라.



보기 6- 85. Position = SegmentBottomRight. AutoSpaceLegendBoxes = False.



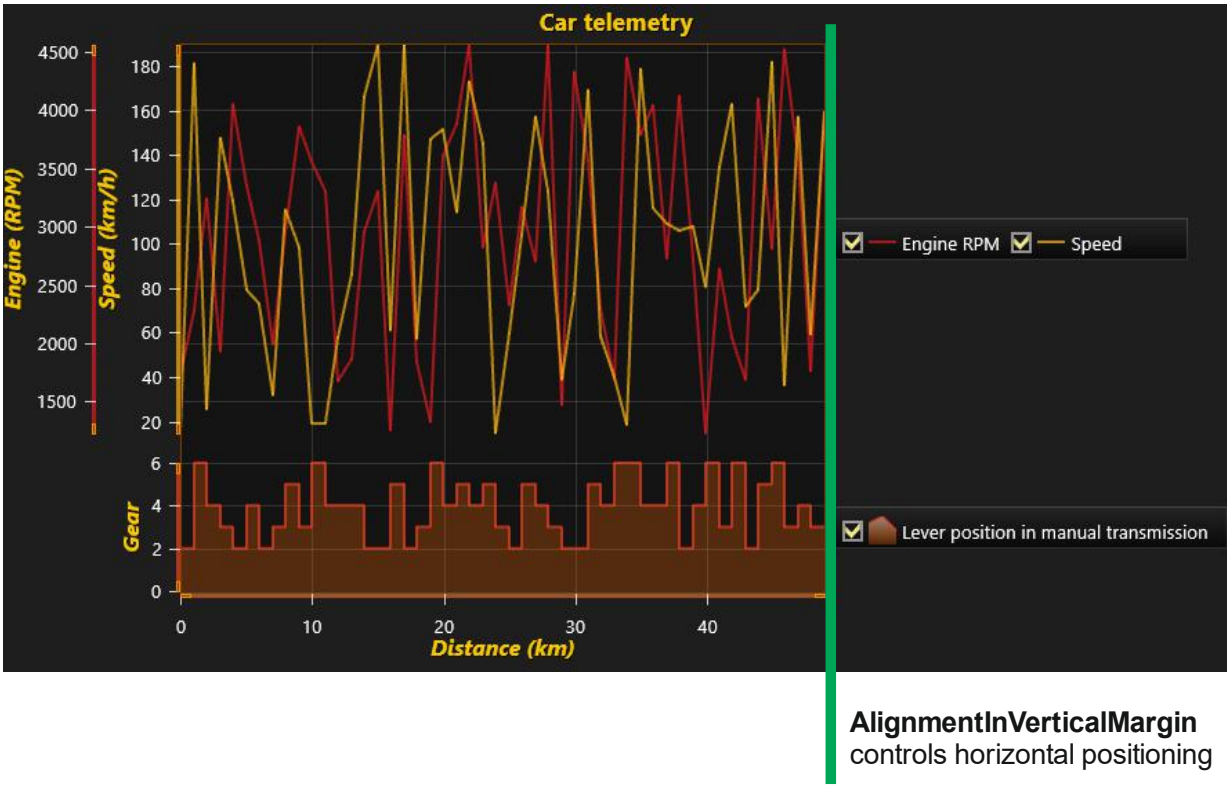
보기 6- 86. Position = SegmentBottomRight. AutoSpaceLegendBoxes = True.

5.21.10 세그먼트 공간에 레전드 상자 정렬

지정된 세그먼트 근처 레전드 상자를 수직으로 정렬하기 위해 **AlignmentInSegmentGap = Near** 설정하라. 세그먼트 사이 공간 중심에 수직으로 정렬하기 위해 **AlignmentInSegmentGap = Center** 설정하라.

5.21.11 같은 마진을 공유하는 여러 레전드 상자의 수평 정렬

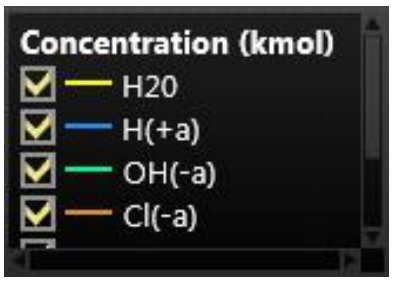
AlignmentInVerticalMargin 속성에은 **Left/Center/Right** 옵션들이 있다. 이 속성은 같은 수직 마진에 설정된 레전드 상자의 수평 위치 선정을 제어한다.



보기 6- 87. 두 레전드 상자에 AlignmentInVerticalMargin = Left 설정.

5.21.12 레전드 상자 크기 변경 및 움직이기

레전드 상자는 크기 변경 및 스크롤 바를 지원한다. 가장자리를 잡아 끌어 크기를 변경하라.



보기 5-90. 레전드 상자 내 스크롤 바

레전드 상자를 움직이거나 크기를 변경할 때 이의 **Position** 은 **Manual** 로 설정 되고 **Offset** 속성이 새로운 위치를 반영하게 업데이트 된다 (6.21.8 장을 보아라).

5.22 줌 및 패닝

ZoomPanOptions 로 줌 및 패닝 설정을 제어하라.

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	

보기 5- 91. ZoomPanOptions 속성 및 하위 속성

줌과 패닝은 구성 가능하며 마우스 좌 또는 우클릭으로 실행 가능하다. 마우스 바퀴로 줌을 땡길 수 있다.

5.22.1 터치 스크린으로 줌

차트에 두 손가락을 올려 모아서 줌 아웃을 하고 모아 줌 인을 하라.

차트는 수평, 수직 또는 둘을 동시에 줌 하려는 것을 감지한다. 이 기능은 '레일로 줌'이라 불리며 **MultiTouchZoomDirection** 로 제어 가능하다 (**Free/XAxis/YAxis/Rails**).

X 와 Y 축 또는 그의 라벨 위 손가락을 모으고 피면 해당 축에만 줌이 적용 된다.

터치로 줌은 **MultiTouchZoomingEnabled = false** 설정으로 비활성화할 수 있다.

5.22.2 터치 스크린으로 패닝

화면에 두 손가락을 올려 같은 속도로 동시에 슬라이드해서 뷰를 패닝하라.

어느 시스템들은 관성으로 패닝을 지원한다. 손가락을 화면 끝으로 "던지는" 것이 가능하다. 이 때 뷰는 계속 패닝 되다 느려져서 멈춘다.

손가락을 X 와 Y 축 또는 이들의 라벨 위 올려 슬라이드 하면 패닝이 해당 축에만 적용이 된다.

5.22.3 마우스 좌클릭

LeftMouseButtonAction 을 **Zoom** 으로 설정해 좌클릭으로 줌을 활성화 하라. Pan 으로 설정해 패닝을 활성화 하라. 좌클릭으로 줌 및 패닝을 비활성화 하기 위해 **None** 으로 설정하라.

5.22.4 마우스 우클릭

RightMouseButtonAction 을 **Zoom** 으로 설정해 우클릭으로 줌을 활성화 하라. Pan 으로 설정해 패닝을 활성화 하라. 우클릭으로 줌 및 패닝을 비활성화 하기 위해 **None** 으로 설정하라.

5.22.5 RightToLeftZoomAction

RightToLeftZoomAction 은 **LeftMouseButtonAction** 또는 **RightMouseButtonAction** 이 **Zoom** 으로 설정 되었을 때 적용 된다. **RightToLeftZoomAction** 은 마우스 줌이 우에서 좌로 만들어 졌을 때 일어날 행동을 지정한다 (마우스 X 버튼 아래-좌표 > 버튼 위-좌표).

다음과 같은 옵션도 선택 가능하다:

ZoomToFit: 모든 Y 와 X 축을 화면에 맞추어 축에 소속된 시리즈 데이터를 보이게 설정한다. 0 보다 큰 값의 **ViewFitYMarginPixels** 을 사용함은 Y 축의 최소 및 최대 끝 둘다 주어진 공간의 픽셀은 비어 있는 데이터를 저장공간으로 스케일 된다.

RectangleZoomIn: 직사각형으로 줌 인. 좌에서 우로 줌.

ZoomOut: **ZoomFactor** 를 이용한 줌 아웃.

RevertAxisRanges: 특유 값으로 축 값을 설정. 뷰가 줌 된 후 또는 축 값이 변경 된 후 복구 된다. 각 축에 **RangeRevertEnabled** 속성이 있다. 이는 축 값을 복구 하는 것을 제어 한다. 활성화 되었으면 **RangeRevertMinimum** 및 **RangeRevertMaximum** 속성들이 마우스를 우에서 좌로 드래그 할 때에 또는 마우스 버튼을 놓을 때에 축에 적용 된다.

PopFromZoomStack: 이전 줌에 사용 됐던 같은 축 범위를 설정한다. 이전 줌 레벨로 돌아간다.

5.22.6 마우스 버튼으로 줌

5.22.6.1 클릭으로 줌 인/아웃

ZoomFactor 속성을 이용해 줌의 정도를 제어하라. 네거티브 줌 효과를 적용하기 위해 값을 역 값으로 설정하라 (1/팩터). 마우스 커서 위치를 줌 중심점으로 적용 된다.

X 차원 줌:

차트 제어 포커스 된 채로 Shift 키를 눌러라. 줌 X 커서가 생긴다. 구성된 마우스 버튼을 눌러 줌 인을 하고 다른 버튼을 눌러 줌 아웃을 하라.

Y 차원 줌:

차트 제어 포커스 된 채로 Ctrl 키를 눌러라. 줌 Y 커서가 생긴다. 구성된 마우스 버튼을 눌러 줌 인을 하고 다른 버튼을 눌러 줌 아웃을 하라. 스택된 **YAxisLayout** 을 사용할 때 줌이 모든 그래프 세그먼트 (Y 축)에 적용된다. Ctrl 및 Alt 키들을 누르면 Y 차원 줌이 마우스가 눌린 그래프 세그먼트에만 적용이 된다.

Shift 와 Ctrl(+Alt) 동시에 눌러 X 및 Y 차원에 줌을 적용하라.

5.22.6.2 직사각형으로 줌 인

구성된 마우스 버튼으로 확대 할 구역 주변 직사각형을 드래그 해라. 왼쪽 위에서 부터 오른쪽 아래로 드래그 하라. X 및 Y 차원 둘다 적용된다. **RectangleZoomDirection** 속성으로 차원들이 선택된다. 줌 직사각형 태두리 및 채우기는 **ZoomRectFill** 및 **ZoomRectLine** 속성들로 변경 가능하다.

5.22.6.3 줌 아웃 직사각형 구성

RightToLeftZoomAction 이 **ZoomToFit**, **ZoomOut**, **RevertAxisRanges** 또는 **PopFromZoomStack** 으로 설정 되었을 때 줌을 하면 줌 아웃 직사각형이 나타난다. 이의 채우기를 **ZoomOutRecFill** 로 설정 및 라인 스타일을 **ZoomOutRectLine** 로 설정하라.

5.22.7 마우스 휠로 줌

MouseWheelZooming 이 활성화 되었을 때 마우스 휠을 위로 스크롤해 줌 인을 하고 아래로 스크롤해 줌 아웃을 하라. 줌 중심점은 마우스 커서의 위치다. **ZoomFactor** 를 이용해 마우스 휠 줌 강도를 수정하라. Shift 키를 누르고 있음으로써 줌이 X 차원에서만 적용이 된다. Ctrl 키를 계속 누르고 있으면 줌은 Y 차원에만 적용 된다. **ScrollMode** 가 **Sweeping** 으로 설정 되었을 때 줌을 사용할 수 없다.

5.22.8 축 위로 마우스 휠로 줌 및 패닝

AxisMouseWheelAction 을 사용해 축 위로 적용되는 마우스 휠 액션을 구성하라.

None: 마우스 휠은 아무 것도 안한다

Zoom: 마우스가 위치한 축에만 줌

Pan: 마우스가 위치한 축에만 패닝

ZoomAll: 마우스가 X 축 위면 모든 X 축 줌. 마우스가 Y 축 위면 모든 Y 축 줌. **YAxisLayout** = **Layered** 일 때만 다른 축에도 적용됨.

PanAll: 마우스가 X 축 위면 모든 X 축 패닝. 마우스가 Y 축 위면 모든 Y 축 패닝. **YAxisLayout** = **Layered** 일 때만 다른 축에도 적용됨.

5.22.9 마우스 버튼으로 패닝

LeftMouseButtonAction 또는 **RightMouseButtonAction** 을 **Pan** 으로 구성해야 패닝이 작동한다. 구성된 마우스 버튼을 누른 채 그래프 구역을 드래그 하라. 패닝을 멈추기 위해 버튼을 놓아라. **PanDirection** 이 **Both** 일 때 패닝은 X 와 Y 축 둘아 드래그 된 거리 만큼 스크롤 한다. **PanDirection** 을 **Vertical** 로 수정하면 Y 축만 타게팅 한다. 반대로 **PanDirection Horizontal** 설정은 X 축만 타게팅을 한다. 패닝 전 **MousePanThreshold** 를 이용해 픽셀로 공차 범위를 주어라. 차트 제어를 위한 ContextMenuStrip 컨트롤을 사용해 패닝이 멈출 때마다 차트가 열리는 것을 알고 있는 것이 유용하다.

5.22.10 Ctrl, Shift 및 Alt 활성화/비활성화

줌 연산은 이 변경 키를 지원한다. 기본적으로 이 키들은 활성화 되어있다. 비활성화 하기 위해 **AltEnabled = False**, **CtrlEnabled = False** 또는 **ShiftEnabled = False** 설정하라.

5.22.11 코드로 줌 인/아웃

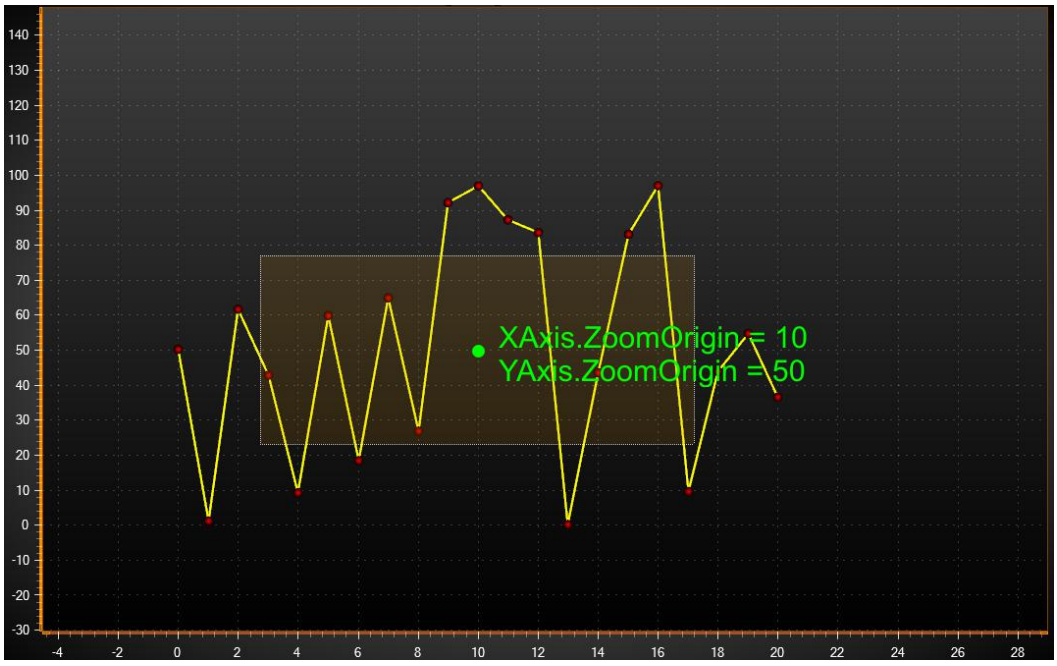
ZoomByFactor(…) 메소드로 센터 포인트 및 줌 팩터로 줌을 하라. **Zoom(…)** 메소드로 직사각형으로 줌을 사용하라. **ZoomToFit()** 메소드는 “맞춤형 줌” 연산을 일으킨다 (모든 Y 및 X 축들을 모든 시리즈 데이터가 보이게 맞춘다).

5.22.12 코드로 축 줌

X 또는 Y 푹 **Minimum** 및 **Maximum** 속성에 값을 설정하라. **SetRange(…)** 를 사용해 둘을 동시에 설정하라.

5.22.13 구성 가능한 원점 위주로 직사각형 줌

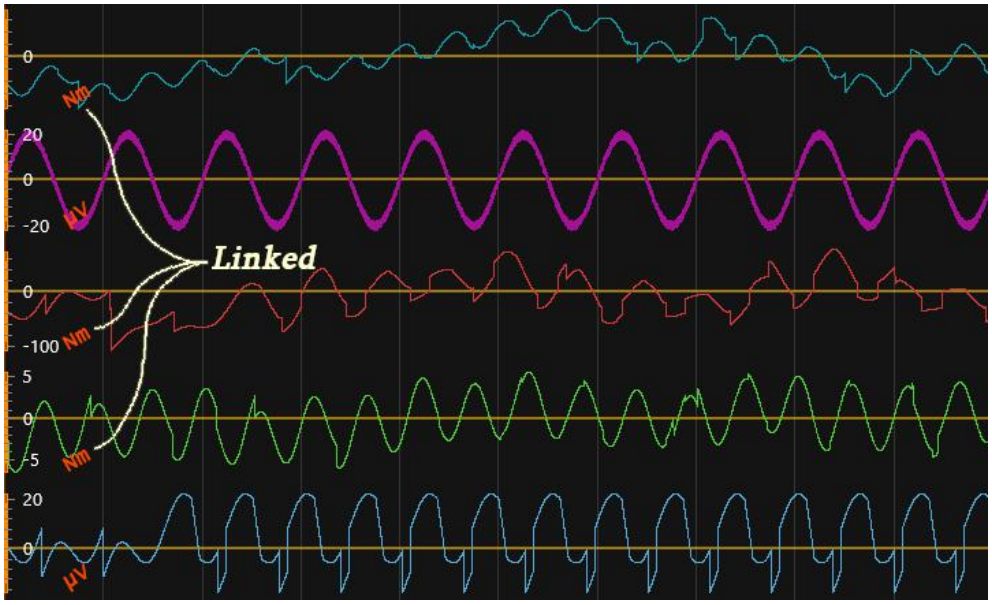
RectangleZoomAboutOrigin 을 활성화 함으로 **ZoomOrigin** 을 통해 X 와 Y 축 값으로 설정된 센터 포인트를 이용해 대칭 직사각형 줌 인/아웃을 하라.



보기 5-92. ZoomPanOptions. RectangleZoomAboutOrigin 활성화. ViewXY.XAxes[0].ZoomOrigin = 10 and ViewXY.YAxes[0].ZoomOrigin = 50.

5.22.14 같은 단위로 Y 축 줌을 연결하기

RectangleZoomLinkYAxes 를 활성화 함으로 모든 Y 축은 같은 **Units** 을 사용한다. **Text** 스트링은 직사각형 줌 된 축과 같은 Y 축 범위를 갖게 된다.



보기 5-93. 5 개의 Y 축이 있는 스택 뷰. 그래프 세그먼트 위 직사각형 줌이 적용되면 Y 축들이 확대 된다. 새로운 Y 축 범위가 같은 Units.Text 를 가진 모든 Y 축들에 복사 된다.

5.22.15 자동 Y 맞춤

데모 예시: 시그널 리더; 오디오 L+R, 구역, 스펙토그램; 웨이브 형, 지속 스펙트럼

AutoYFit 속성을 이용해 자동 Y 축 수정을 제어하라. 자동 Y 맞춤은 Y 축 범위를 수정해 차트의 모든 데이터를 X 범위에 보이게 수정할 수 있다. 이는 특히 실시간 모니터링 용도에 유용하다. 맞춤은 시간 간격으로 적용된다. **UpdateInterval** 을 이용해 간격을 밀리 초로 설정하라. **MarginPercents** 는 시리즈 와 그래프 태두리에 빈 공간 이 남을 시 정의 하는 데에 사용 가능하다. **Through** 를 활성화 함으로 모든 데이터 위해 맞춤 분석이 만들어지지만 성능 중요 시스템에 오버헤드를 일으킬 수 있다. 비활성화 함으로 최신 데이터의 작은 부분만 맞춤 루틴에 사용 되며 어느 응용 프로그램에는 잘못된 행동을 일으킬 수 있다.

AutoYFit 은 **ZoomPaddingOptions** 통해 활성화 할 수 있다:

```
_chart.ViewXY.ZoomPanOptions.AutoYFit.Enabled = true;
```

Which Y-axes are automatically fit should be also defined. With **TargetAllYAxes**, automatic Y fit can be applied to every Y-axis simultaneously. Alternatively, **AllowAutoYFit** can be enabled for each Y-axis separately.

```
// Enable AutoYFit for all Y axis.
_chart.ViewXY.ZoomPanOptions.AutoYFit.TargetAllYAxes = true;
```

```
// Enable AutoYFit only for this Y axis.
_chart.ViewXY.YAxes[0].AllowAutoYFit = true;
```

주의! AxisY 클라스는 Y 차원 맞춤을 위한 **Fit()** 메소드도 있다.

5.22.16 종횡비

AspectRatioOptions.AspectRatio 는 X/Y (지도에서는 경도 / 위도) 비율을 제어 한다.

기본적으로 각 X 및 Y 축 범위들을 따로 설정 가능하게 **Off** 로 설정 되어 있다. 종횡비를 **Manual** 로 설정함으로 **ManualAspectRatioWH** 속성을 원하는 비율로 맞출 수 있다. **ManualAspectRatioWH** 을 변경함은 x 축의 **Minimum** 및 **Maximum** 속성들을 원하는 종횡비를 갖게 한다. 줌 연산 또한 종횡비 설정을 따른다.

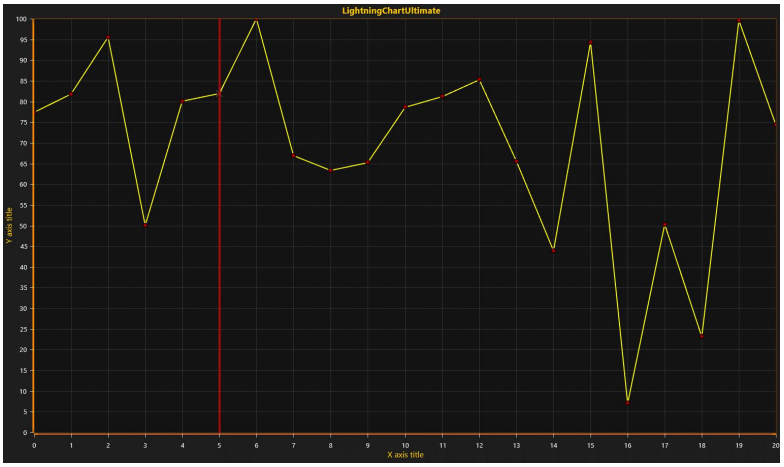
ManualAspectRatioWH 은 다음과 같이 계산 된다:

ManualAspectRatioWH = 픽셀로 뷰 넓이 / 픽셀로 뷰 높이 * x 축 범위 / Y 축 범위

예를 들어:

ManualAspectRatioWH = 1530 / 902 * (20 – 0) / (100 – 0)

뷰의 넓이와 높이는 창의 크기에 달려있다. 축 범위는 그냥 최대값 – 최소 값 이다.



보기 5-94. 차트의 뷰 구역. ManualAspectRatioWH 를 계산하기 위해 픽셀 수로 크기가 사용된다.

AspectRatio 가 **Off** 이외에 다른 설정으로 되어있다면 축 스케일 닙을 사용할 수 없다.

지도에는 (5.25 를 보세요), **AspectRatio** = **AutoLatitude** 는 아주 유용한 옵션이다. **AutoLatitude** 는 지도를 다른 위치에서 볼 때 종횡비를 동적으로 변경해 준다. 종횡비는 뷰의 중심점으로 결정 된다.

5.22.17 특유 x 또는 y 축을 줌 및 패닝 연산에서 제외하기

- 줌 연산에서 특유 X 또는 Y 축에서 제외하기 위해 다음과 같이 설정 하라

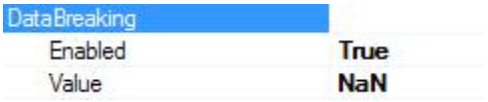
  axis.ZoomingEnabled = False

- 패닝연산에서 특유 X 또는 Y 축에서 제외하기 위해 다음과 같이 설정 하라

  axis.PanningEnabled = False

5.23 NaN 또는 기타 값으로 DataBreaking

데모 예시: 시리즈로 데이터 브레이크



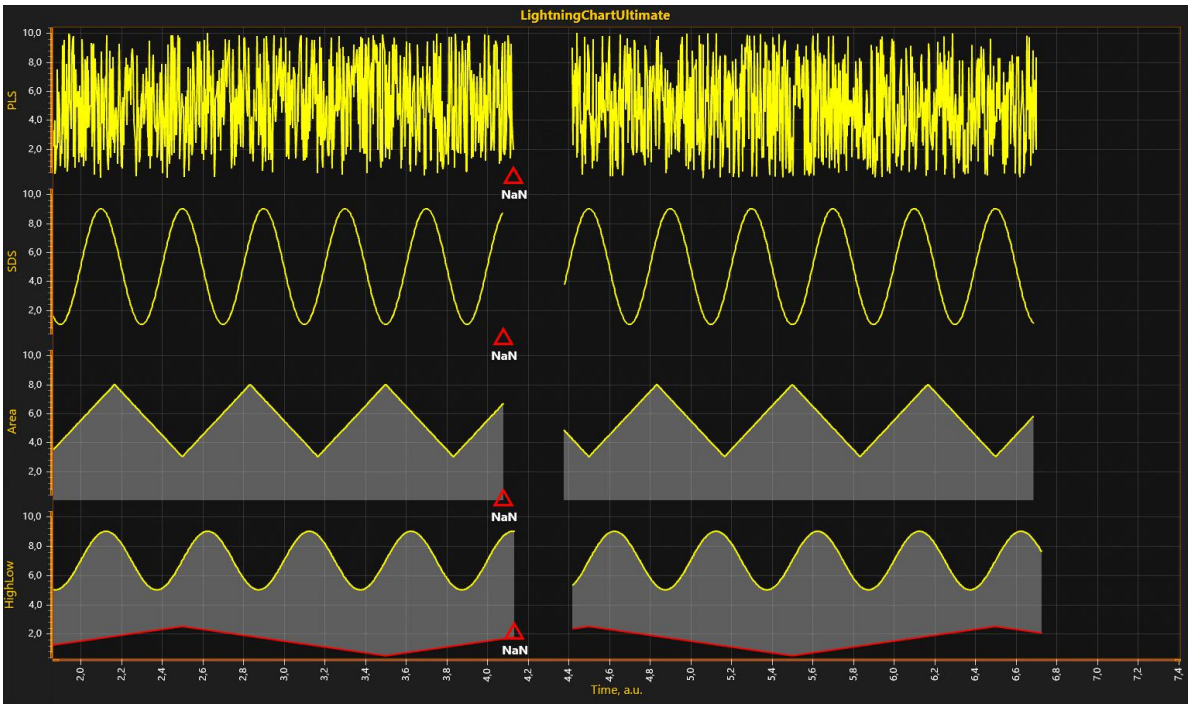
보기 5-95. DataBreaking 을 지원하는 시리즈 내 옵션

다음 시리즈 종류는 데이터 브레이크를 지원한다:

- PointLineSeries
- FreeformPointLineSeries
- SampleDataSeries
- AreaSeries
- HighLowSeries

- PointLineSeries3D

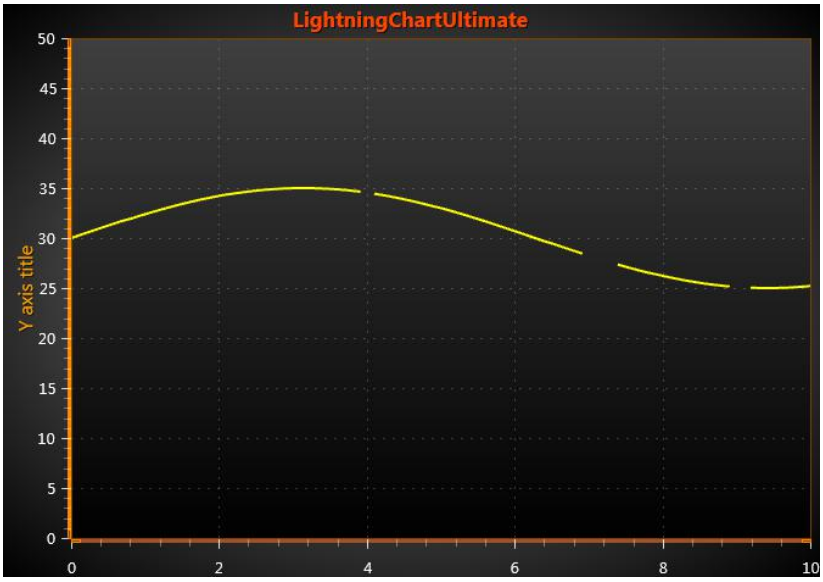
라이트닝차트는 특별 브레크 값과 일치하는 데이터 포인트의 렌더링은 건너 뛴다. 기타 다른 값 모두는 보통대로 렌더링을 한다.



보기 5- 96. PointLineSeries, SampleDataSeries, AreaSeries and HighLowSeries 에 사용할 DataBreaking.

주의! DataBreaking.Enabled = True 일 때 상당한 오버헤드를 일으킨다. 고 실시간 데이터 레이트가 필요한 솔루션에는 이 설정을 추천하지 않는다. ClipAreas 사용을 추천한다. 제 6.24 장에 더 많은 정보를 볼 수 있다.

예를 들어 NaN 을 사용해 PointLineSeries 데이터 브레이크 할 때:



보기 5-97. NaN 을 사용해 PointLineSeries 끊기.

코드:

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

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

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

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

  // set data gap defining value (default = NaN)
  pls.DataBreaking.Value = double.NaN;
  SeriesPoint[] points = new SeriesPoint[pointCount];
  for (int point = 0; point < pointCount; point++)
  {
      points[point].X = xValues[point];
      points[point].Y = yValues[point];
  }

  //Assign the data for the point line series
```

```
pls.Points = points;

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

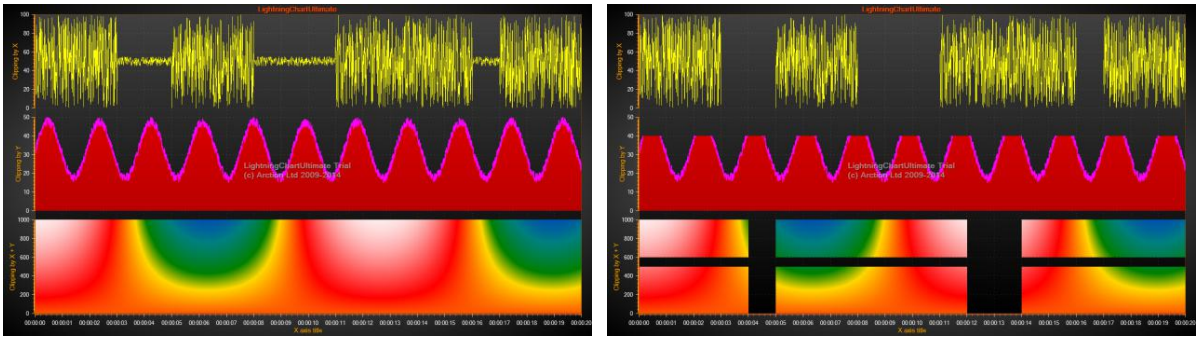
5.24 ClipAreas

데모 예시: 클립 구역

DataBreaking (5.23 보세요) 와 같이 **ClipAreas** 를 사용해 시리즈 데이터 부분의 렌더링 방지 할 수 있다. 사용 불가 한, Y 값 대로 범위-밖 데이터 범위 등을 필터하기 위해 사용할 수있다.

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.

ClipArea 는 할당 된 시리즈에 적용 된다. 이것은 렌더링 단계 클리핑이다. 아래 실제 데이터가 있어야 ClipArea 위 시리즈에 마우스를 올렸을 때 마우스 연산이 반응을 한다.



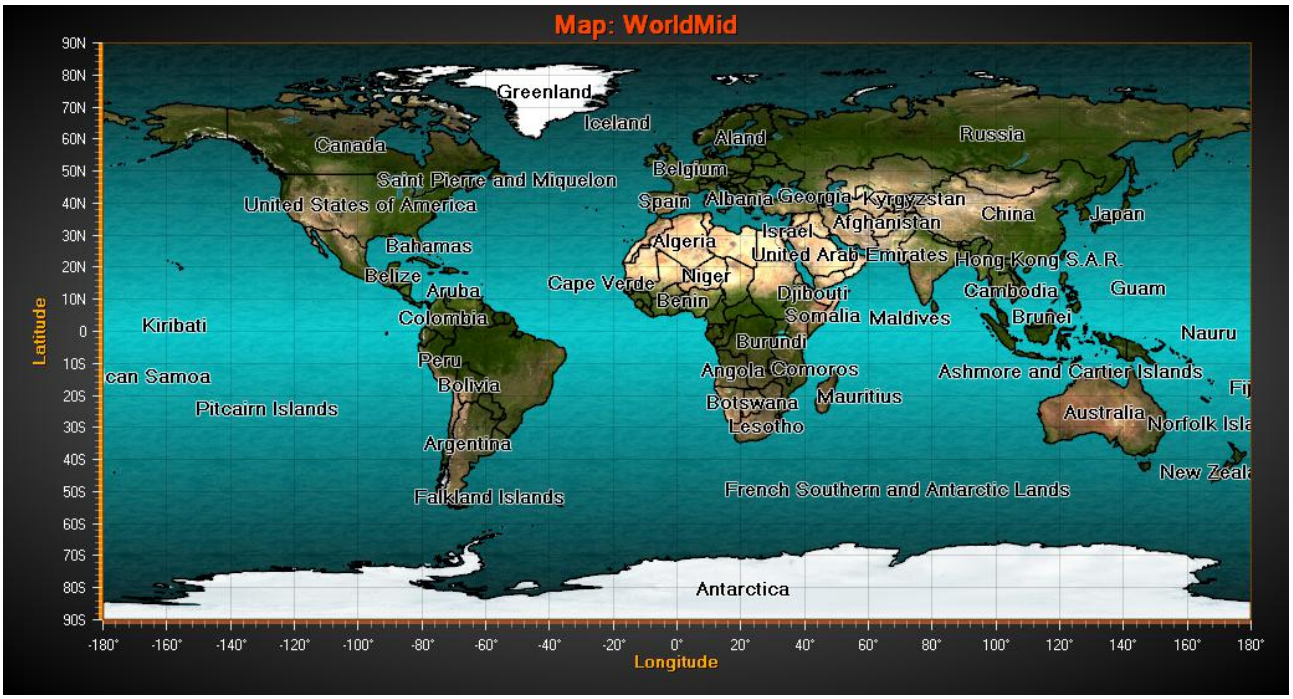
보기 5-98. 3 개의 시리즈에 ClipArea 정의. PointLineSeries, AreaSeries, and IntensityGridSeries 시리즈다. 왼쪽에는 ClipAreas 가 사용되지 않았다. 오른 쪽에는 ClipAreas 가 적용 됐다. 노란색 PointLineSeries 에는 X 차원 클리핑 구역이 정의 되어 저 앰프 데이터를 벗겼다. 빨간색 AreaSeries 에는 Y 차원적 ClipArea 가 너무 높은 앰프의 데이터를 위에서 자른다. IntensityGridSeries 에는 X 및 Y 차원적 ClipArea 가 사용돼 시리즈가 특정 구역에 렌더링 되는 것을 방지 하기 위해 사용 됐다.

ClipAreas 사용은 라인을 여러 데이터 세그먼트로 브레이크 할 수 있는 성능적으로 선호 되는 방식이다.

DataBreaking 특성 또는 실시간 모니터링 중 몇 백개의 개인 시리즈를 생성하는 것 보다 성능적으로 선호 된다.

5.25 지도

Maps 속성 및 이의 하위 속성들을 이용해 지리적 지도를 표현할 수 있다. 라이트닝차트 지도는 두 종류로 온다: **벡터 지도** 및 **타일 지도.** 직사각 투영으로 지도들이 구현된다.



보기 5-99. 세계의 직사각 투영. X 범위는 -180 도 에서 180 도 까지다 (180W 에서 180E). Y 범위는-90 도 에서 90 도 이다 (90S to 90N). 극지방은 이 투영에서 굉장히 늘어져 보인다.

이 투영은 지도 사용과 동시에 라이트닝차트의 시리즈 종류 및 X 와 Y 축에 묶인 기타 객체 사용을 허용한다.

5.26 벡터 지도

데모 예시: 세계 지도; 지도 경로; 환경적 데이터 있는 지도; 바람 데이터

지리적 벡터 데이터는 **.md** 확장으로 LightningChart 지도 파일에 저장 되어 있다. 라이트닝차트는 맵 파일 세트와 함께 온다.

X 축은 경도 Y 축은 위도로 사용된다. 지도 좌표 축의 설명을 위해 6.2.3 장을 보아라. 지도 좌표는 십진수로 표시된다. 경도 원점은 적도에 및 위도 원점은 영국 그리니치다.

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

보기 5-100. Map 의 속성 및 하위 속성. TileLayers 컬렉션 및 TileCacheFolder 이외 트리 전체가 벡터 지도를 위한 것이다. 이 둘은 타일 지도와 사용 가능하다.

5.26.1 유효 지도 선택

디렉토리 명을 지도 파일이 존재하는 **Path** 속성으로 설정하라. 라이트닝차트로 불러오는 지도들 중 유효한 지도는 **Type** 속성으로 선택 가능하다. 본인이 생성한 지도 파일을 사용하기 위해 **FileName** 속성을 설정하라.

지도를 사용하기 원하지 않는다면 **Type 를 Off** 로 설정하라.

보기 5-101. Map Type 옵션. 라이트닝차트로 불러오는 지도들이 보여진다. 종류 명 접미사는 지도의 세부 수준을 대충 알린다.

대체적으로 라이트닝차트의 지도들은 아주 많은 디테일로 만들어졌다. 실시간 모니터링 솔루션을 위해서는 제대로 된 세부 설명과 성능 수준의 지도를 선택 하는 것이 중요하다.

5.26.2 종횡비

ViewXY.ZoomPanOptions.AspectRatioOptions.AspectRatio 는 X/Y (또는 경도 위도) 비율을 제어 한다.

지도를 늘릴수 있는 X 와 Y 축 범위를 따로 설정하기 위해서는 **Off** 로 설정 하라. **AutoLatitude** 는 지도를 다른 위치에서 볼 때 종횡비를 동적으로 변경한다. 종횡비는 뷰의 중심점으로 인해 결정 된다. 종횡비를 **Manual** 로 설정함으로 **ManualAspectRatioWH** 속성을 이용해 선호 비율로 설정하라. 종횡비의 계산법의 세부 정보를 위해 6.22.16 을 봐아라.

5.26.3 레이어 및 생김세 설정

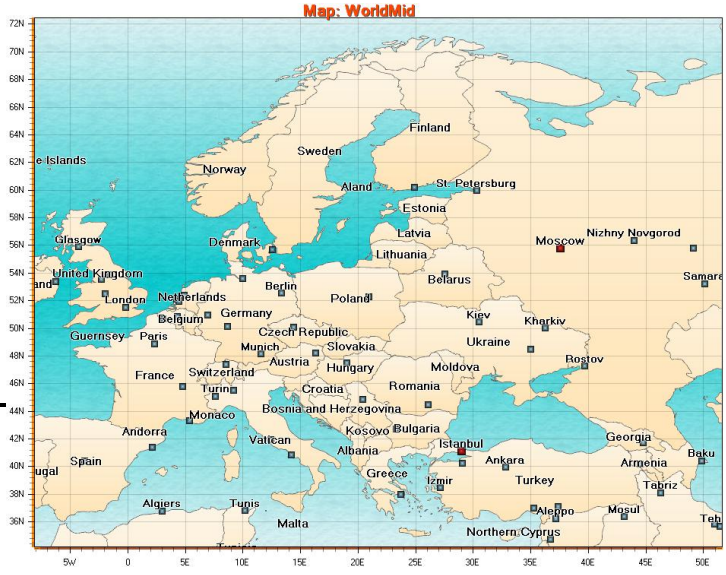
각 지도 파일은 여러 층을 가질 수 있다. 예를 들어 육지 지역을 위한 층, 호수, 강, 길 및 도시를 위한 각각의 층. 레이어와 그의 데이터는 **Layers** 배열 속성에서 접근 가능하다.

⊟ 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**

보기 5-102. Properties 에디로 연 지도 레이어 세부 정보.

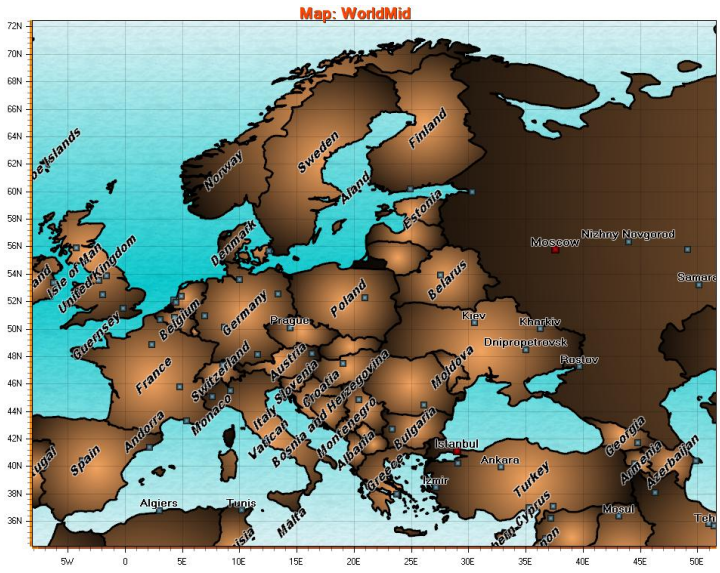
각 층에는 특별 종류가 있다. 층 생김세 옵션은 해당 옵션 속성으로 변경 가능하다. **LandOptions** 를 사용해 육지 지역의 생김세를 변경, 호수를 변경하기 위해 **LakeOptions,** 강은 **RiverOptions,** 길은 **RoadOptions,** 도시는 **CityOptions** 및 기타 불특정 레이어 종류의 변경을 위해서는 **OtherOptions** 을 사용하라.

⊟ 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**



. 기본 LandOptions 및 해당 유럽에 적용 된 뷰

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



LandOptions.

5.25.3.1 각 레이어 아이템 채우기 및 태두리 스타일 설정

각 지도 요소 채우기 또는 태두리 생김세는 개별적으로 설정 가능하다. **BorderDrawStyle** 및 **RegionDrawStyle** 속성들을 **Individual** 로 변경하라. 그 후 Items 컬렉션을 접근해 원하는 아이템으로 가 **BorderLineStyle** 및 **Fill** 속성들을 수정하라. **Items** 컬렉션은 **Name** 속성으로 프로그램 적으로 이동 가능하다. 여기에선 “Germany” 가 보인다.

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.PointFlo
Fill	Arction.LightningChartUltimate.F
Bitmap	Arction.LightningChartUltimate.B
Color	White
GradientColor	DarkViolet
GradientDirec	270
GradientFill	Linear
Style	ColorOnly
Name	Germany

보기 5-105. 레이어 태두리 라인 및 지역 채우기 스타일을 Invidual 로 설정 후 구역을 Items 컬렉션 내에서 수정.

보기 5-106. 독일 지역이 개별 채우기 및 태두리로 표시됐다.

5.26.4 마우스 대화식

MouseInteraction 을 활성화 해 지도 지역 및 객체의 상호 운영을 하라. 지역 (육지, 호수) 및 벡터 층 (강, 길)은 마우스로 포인트 가능하다. **MouseHighlight** 가 **Simple** 로 설정 되었을 때 마우스를 객체 위로 올리면 **SimpleHighlightColor** 로 하이라이트 된다. **MouseHighlight** 가 **Blink** 로 설정 되었을 때 객체가 밝고 어두운 색을 번갈아 가며 깜빡인다. **None** 으로 설정 시 객체는 하이라이트 안되지만 클릭 가능하고 예를 들어 **Maps.MouseDownOnMapItem** 이벤트를 실행 가능하다.

지도 객체는 인구 수 또는 기타 통계 데이터 등 관련 데이터를 포함 할 수 있다. **MouseOverOnMapItem/MouseOverOffMapItem/MouseDownOnMapItem** 이벤트 핸들러를 이용해 데이터에 접근하라. 지도 아이템을 위한 데이터는 **GetInfo** 메소드를 이용해 불러올 수 있다. 키 및 값의 딕셔너리가 주어진다.

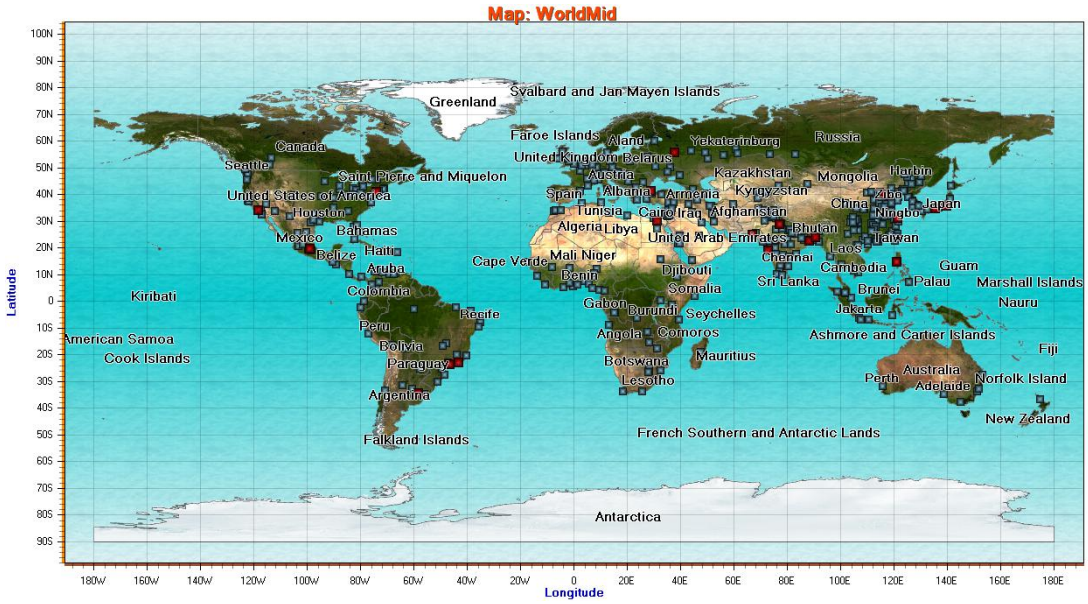
다음은 모든 데이터를 목록 상자에 보여줄 수 있는 예시다. 아이템 명은 다른 텍스트 박스에 표시 되었다.

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

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

5.26.5 배경 사진

Maps.Backgrounds 속성에 **MapBackground** 추가 하는 것은 지도의 배경에 비트맵 이미 표시를 허용한다. GIS 데이터 제공 업체에서 위성 이미지 또는 기타 라스터 이미지가 제공 된다. 이 이미지는 **Image** 속성에 설정 가능하며 위도 경도 범위는 **LatitudeMin**, **LatitudeMax**, **LongitudeMin** 및 **LongitudeMax** 속성들로 설정 가능하다. 이미지는 설정 범위 밖에는 표시 되지 않는다.

지도 레이어를 뚫어 배경을 보여주기 위해 각 레이어 채우기 설정을 수정할 필요가 있을 수 있다. 투명색 또는 저 알파 레벨이 있는 색을 사용 하라.

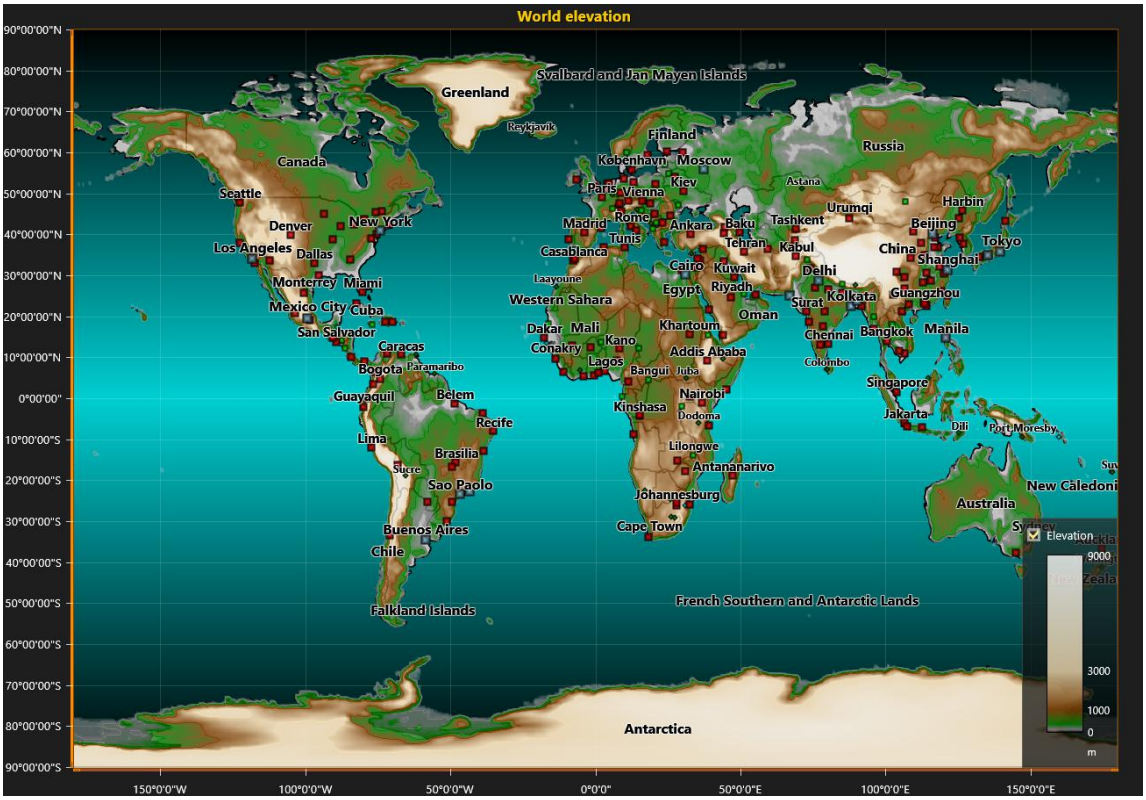


보기 5-107. 세계 지도. LandOptions.FillVisible 은 거짓으로 설정 되었다. 한 배경 이미지의 위도 범위는 -90 에서 90 그리고 경도 범위는 -180 에서 180 로 설정 되었다. 지도 지역 경계선 및 지도가 보여진다.

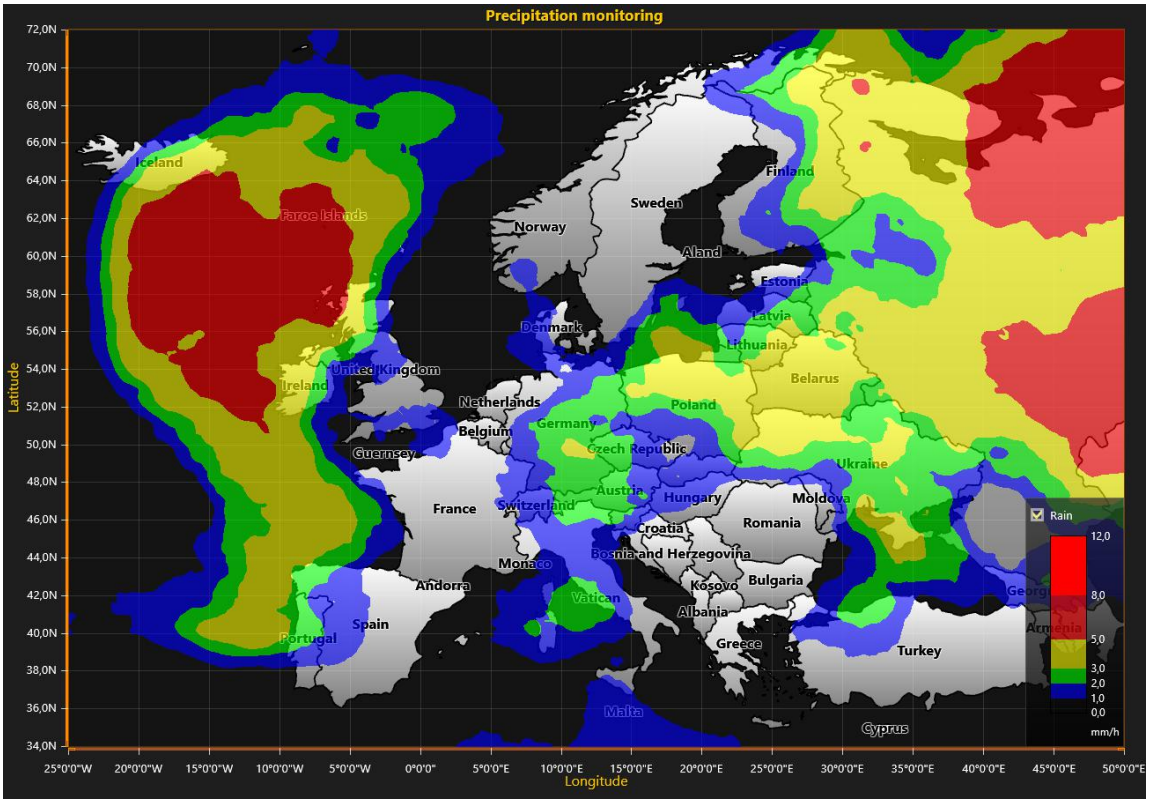
5.26.6 지도와 기타 시리즈 합치기

지리적 지도는 다른 ViewXY 시리즈 종류와 함께 사용 가능하다. 지도는 배경에 그려지고 시리즈는 그 위에 그려진다.

보기 5-108. 유럽 지도. 경로로 2 개의 FreeformPointLine 시리즈가 있다. 깃발 마커가 마우스 대화식 웨이 포인트로 추가 되었다.



보기 5-109. IntensityGrid 로 고지를 표시하는 세계 지도.



보기 5-110. 유럽 지도 위 IntensityGrid 시리즈로 날씨 레이더 데이터 시각화.

5.26.7 ESRI 모양 파일 데이터에서 지도 불러오기

불러오기 특성은 .shp 파일에서 라이트닝차트 지도 파일 (.md)을 만든다. ESRI 쉐입파일 (*.shp)은 벡터 및 다각형 데이터를 지원하는 널리 이용 되는 파일 유형이다.

지도 마법사를 사용해 쉐입파일 데이터를 라이트닝차트 (LC) 지도 데이터 유형으로 변경할 수 있다. LC 유형은 레이어링을 지원하여 여러 쉐입파일을 한 파일로 합칠 수가 있다. 지도 파일 구형 및 객체가 최대 런타임 성능을 위해 사전 프로세싱 되었다.

팁: LightningChart® .NET 의 데모 어플리케이션에는 지도 불러오기의 예시가 담겨 있다. 불러오기 마법사를 실행해 불러오기 기능을 통해 커스텀 LC 지도를 만들어라.

변경은 최소 세 단계로 완료 가능하다:

1. Shapefile Selection Dialog 에 있는 파일들 기반으로 파일 선택 및 레이어 설정.
2. 파일 텍스트 인코딩 결정.
3. 결과 지도 파일에 포함된 아이템 선택.

2 및 3 단계들은 각 소스 shp 파일마다 반복 된다. 쉐입파일은 어느 인코딩을 사용하는지 알리지 않기에 사용자가 선택을 해야만 한다.

단계들을 따른 후 변환이 시작된다. 지도가 커스텀 응용 프로그램에서 불러 왔으면 개발자가 이벤트 핸들러 설정하는 것을 추천한다. 이는 변환 작업이 오래 걸릴수도 있으며 사용자에게 변환 진행 여부에 대해 알리기 위함이다.

또한 사용자가 베이스 레이어를 선택하면 단계 사이 상당한 딜레이가 발생 할 수 있다. 이는 레이어 기반으로 데이터를 사전 필터링으로 인해 발생하는 것이다.

5.26.7.1 shp 데이터 불러오기 위한 프로그래밍 인터페이스

컨버션은 다음 메소드를 따른 Maps.MapConverter 클라스로 초기화된 스레드로 실행 된다:

```
public bool SelectFilesAndConvert()
```

컨버션 진행을 모니터 하기 위해서는 이벤트 핸들러 대리자가 있다:

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

초기화:

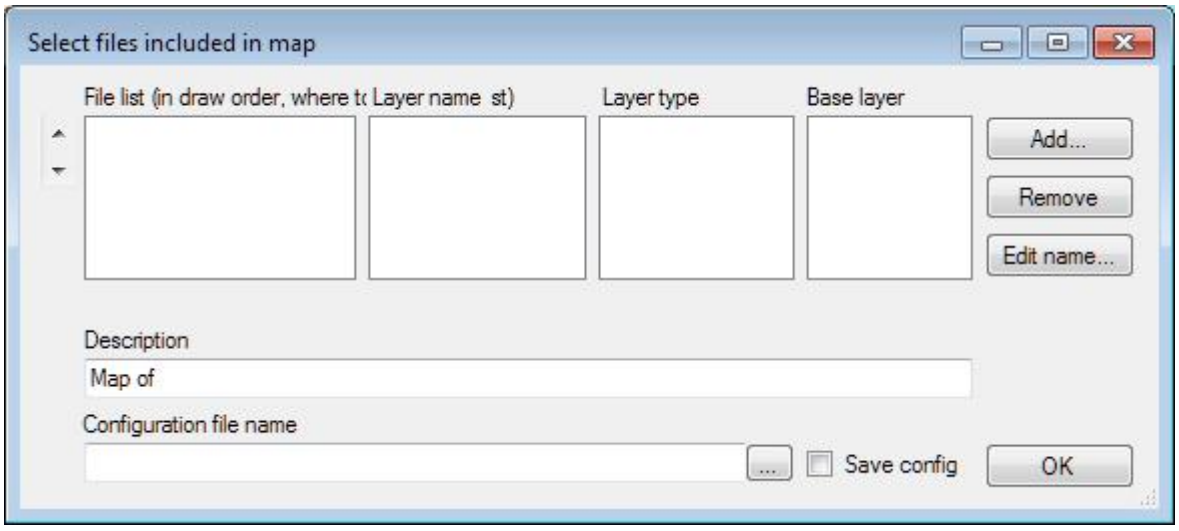
```
MapConverter mapConverter = new MapConverter();
mapConverter.ConversionStateChanged += new MapConverter.ConversionStateChangedHandler(mapConverter_ConversionStateChanged);
```

5.26.7.2 대화창

컨버션 프로세스에는 주로 3 개의 다이어로그가 포함됐다. 필터를 선택하기 위해 뚜렷한 다이어로그가 있다.

5.26.7.2.1 쉐입파일 선택 대화창

`SelectFilesAndConvert()` 함수가 불린 후 파일 선택 대화창이 열린다. 이 다이어로그에서는 사용자가 소스 파일 선택 및 레이어링 설정할 수 있다. 사용자가 대화창에서 파일 선택함으로 지도 구성을 저장할 수도 있다.



보기 5-111. 소스 모양 파일 선택 대화창.

파일 목록

그려진 순으로 파일 목록이 있다. 아래에 있는 파일 데이터가 마지막으로 그려졌다. 파일 순서는 목록 왼쪽에 위/아ㅣ래 화살표를 사용해 변경할 수 있다. 파일을 선택해 위 아래를 눌러 파일 순서를 변경하라.

레이어 명

레이어 이름. 예) "나라명".

레이어 종류

레이어 종류 (레이어 렌더링에 어느 옵션이 사용됐는지 명시)

- 도시: 레이어 아이템은 쉐입파일 종류 POINT
- 호수: 레이어 아이템은 쉐입파일 종류 POLYGON
- 육지: 레이어 아이템은 쉐입파일 종류 POLYGON
- 강: 레이어 아이템은 쉐입파일 종류 POLYLINE
- 길: 레이어 아이템은 쉐입파일 종류 POLYLINE
- 기타: 레이어 아이템은 쉐입파일 종류 POLYGON 또는 POLYLINE

베이스 레이어

위 레이어 아이템 선택 필터에 사용. 사용자가 한개 또는 몇가지의 나라만 있는 지도를 사용하고 싶은데 글로벌 지도만 있을 때 사용할 수 있다. 예를 들어 레이어에 나라가 있으면 선택 나라만 마지

막 지도에 포함된다. POINT 종류에는 작은 오프셋이 적용되어 포인트가 경계선 근처에 위치 되어 있으면 베이스 레이어와 겹치지 않더라도 포함이 된다. 선택된 쉐입파일의 모든 데이터가 결과 지도에 포함 되어 있으면 베이스 레이어를 선택하지 마라. 이는 아이템 선택을 상당히 느리게 한다. 이는 모든 아이템이 베이스 레이어에 겹치면 체크가 되는데 아주 오랜 시간이 소요되는 프로세스다.

설명

지도 속성에 보여지는 프리 텍스트.

구성 파일 명

XML 구성 파일 명. 레이어 불러오기 및 교체에 사용. **주의!** 불러오기로 지도 구성 생성할 때 한 파일을 사용하라. 메소드 교체는 한 shp 입력 파일만을 받는다.

구성 저장

나중에 사용을 위해 지도 구성을 xml 파일로 저장하고 싶을 때 이 옵션을 체크 하라. 구성 파일 선택은 이 옵션을 자동 선택한다.

버튼 추가

목록에 추가할 쉐입파일 선택할 때 눌러라.

버튼 제거

목록에서 선택 파일을 제거한다.

이름 버튼 수정

눌러서 "레이어 이름 에디터"를 열어라. 레이어 이름 설정하라.

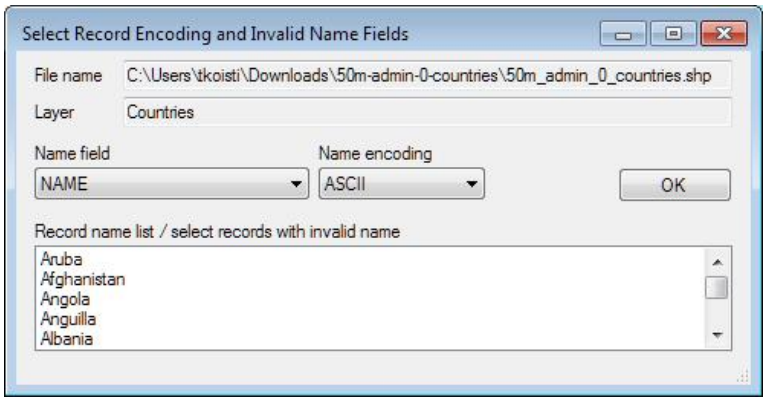
OK 버튼

눌러서 (아이템 선택) 다음 단계로 넘어가라.

5.26.7.2.2 선택 기록 인코딩 및 무효 이름 필드

이 대화창은 파일 텍스트 인코딩 및 무효한 또는 일반 이름을 가진 필드를 선택할 때 사용된다. 모양 파일 인코딩은 다를 수 있으며 파일에는 인코딩 정보가 없으니 사용자가 해당 인코딩을 선택해야 한다. 여러 아이템들의 이름이 " UNK" 과 같을 수 있다. 이 대화창에서 사용자는 어느 아이템 이름을

비울지 선택할 수 있다. 담음 단계에 선택 시 결과 파일에 아이템들이 계속 포함된다는 점을 유의하는 것이 좋다.



보기 5-112. 필드 선택 대화창에 '기록 인코딩' 및 '무효 이름'

파일 명

인코딩이 해당 되는 모양 파일 명.

레이어

레이어 이름.

이름 필드

모양 파일에 아이템 이름 필드. 다른 필드 선택 후 목록이 업데이트 된다.

이름 인코딩

아이템 명 인코딩 (이름이 맞지 않을 때 다른 값들을 사용해 보아라). 다른 인코딩 선택 후 목록이 자동으로 업데이트 된다.

기록 이름 목록 / 무효한 이름 가진 기록 선택

"Name" 필드에 선택된 아이템 목록.

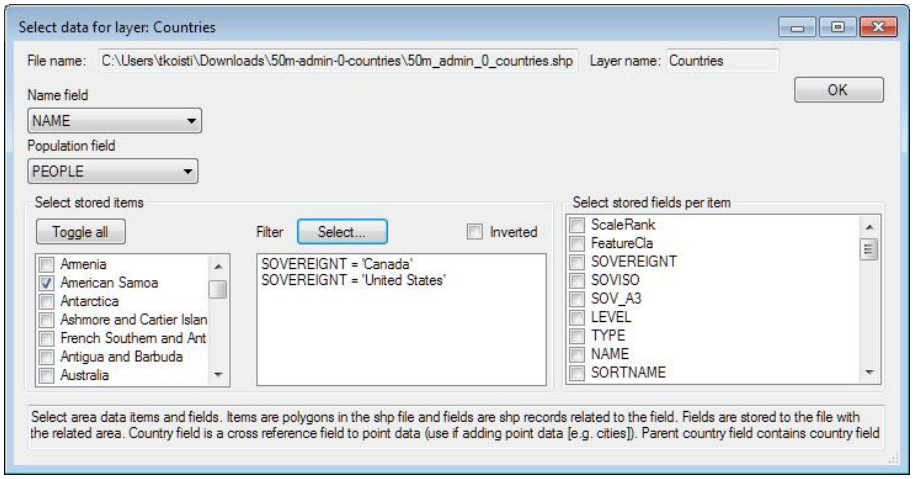
OK

인코딩 선택 (및 무효 이름) 확인.

5.26.7.2.3 레이어 데이터 선택 대화창

본 대화창은 모양 파일에서 결과 지도 파일에 포함 될 아이테을 선택하기 위해 사용 된다. 레이어 명이 제목에 연결 되어있다. 대화창은 적응형이라여러 필드 중 선택할 수 있는 레이어들이 있을 수 있

다. 예를 들어 **River/Road** 종류의 층에는 라인 넓이 선택이 있고 (해당 되면) 라인 넓이 필드에 설정 될 수 있다. 데이터가 대화창에 요구하는 필드를모두 갖고 있지 않을 수 있다. **Name** 필드는 모든 아이템에 필수 값이다.



보기 5-113. 레이어 데이터 선택 대화창.

대화창에 선택 가능한 유저 인터페이스 아이템은 다음과 같다:

파일 명

파일의 이름.

레이어 명

레이어의 이름.

Name 필드

아이템 이름에 사용 되는 필드. 인코딩 선택 대화창에서 자동 선택 되지만 여기에서도 수정 가능하다.

인구 필드

인구 관련 데이터에 사용되는 필드이다.

나라 필드

나라 명 필드.

라인 획 넓이 필드

라인 넓이. 라인의 렌더링을 지도한다.

저장 아이템 선택

아이템을 개별적으로 선택한다. 모두 선택하거나 **Filter** 를 사용해 아이템의 하위 집합을 선택하라.

모두 토글

파일의 모든 필드를 선택한다.

필터/선택…

선택 값이 있는 필드가 있는 필드들을 선택한다. 위 이미지에는 SOVEREIGNT 필드가 ”Canada” 또는 ”United States” 로 설정된 아이템만 지도에 선택 되어 있다.

역 값

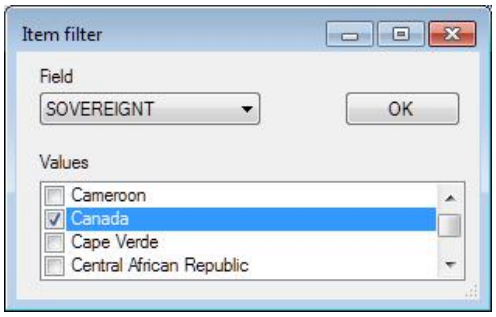
역 필터 선택 (필터에 선택 된 필드가 결과 지도 파일에 포함 되어 있지 않는다).

아이템 당 저장 필드 선택

아이템 당 포함 되어야 할 필드를 클릭하라. 이 필드들은 아이템 당 필드를 갖고 있는 Dictionary 의 필수 값이다.

5.26.7.2.4 아이템 필터

이 대화창은 Layer 데이터 선택 대화창에서 열리며 결과 지도를 위해 아이템 필터 하는 데에 사용된다.



보기 5-114. 아이템 필터 대화창.

필드

필터링 기반 될 필드 선택.

값
결과하는 아이템에 포함 될 값 선택

위 선택은 SOVEREIGNT 필드 명에 ”Canada” 값이 들어 있는 아이템들만 결과 지도에 포함 되어있다.

5.26.8 지도 레이어 불러오기 및 교체

사용자는 지도에 새로운 레이어를 불러올 수 있고 존재하는 레이어를 교체할 수 있다. Maps 인터페이스에서 지도에 레이어 불러오기 및 교체하는 데에 4 가지 방식이 있다. 이는 소프트웨어 어플리케이션이 실행 중 자주 업데이트 되는 shp 데이터를 불러오는 데에 유용하다.

ImportNewLayer 메소드는 주어진 레이어 인덱스에 새로운 지도 레이어를 삽입한다. **ImportReplaceLayer** 메소드는 주어진 레이어 인덱스에 지도 레이어를 교체한다.

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

shpFilename 이 소스 shp 파일 명의 이름일 때 및 **targetLayerIndex** 이 새로운 레이어의 인덱스이다. 이 방식은 지도 구성 설정에 위에 표시된 대화창을 이용한다.

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

shpFilename 이 소스 파일 명의 이름이고 **targetLayerIndex** 가 새로운 레이어의 인덱스이고 **configFile** 가 지도 구성 파일 이름이다. 이 방식은 위에 표시 된 대화창으로 생성된 구성 파일을 사용한다.

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

shpFilename 가 소스 shp 파일 명의 이름이고 **targetLayerIndex** 가 새로운 레이어의 인덱스이다. 이 방식은 위에 표시된 대화창을 이용해 지도 구성을 설정한다.

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

shpFilename 가 소스 shp 파일 명의 이름이고 **targetLayerIndex** 가 새로운 레이어의 인덱스이고 **configFile** 가 지도 구성 파일 명이다. 이 방식은 위에 표시 된 대화창으로 생성 된 구성 파일을 사용한다.

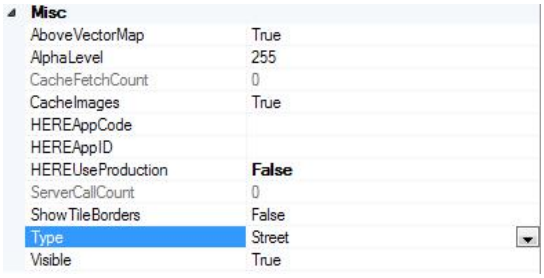
구성 파일은 일반 xml 파일이다. 텍스트 에디터로 수정 가능하다. 하지만 수정 하지 않는 것을 추천한다.

5.27 타일 지도

데모 예시: HERE Maps 길; HERE Maps 위성; 작은 차트 있는 HERE Maps

라이트닝차트는 다음과 같은 온라인 타일 데이터 서비스를 지원한다:

- Here: 거리 지도, 위성 이미지



보기 5-115. TileLayer 의 속성.

ViewXY.Maps.TileLayers 컬렉션에 **TileLayer** 객체를 추가하라. 여러 레이어를 삽입 가능하고 **AlphaLevel** 속성으로 반투명하게 만들 수 있다. **TileLayer** 객체들은 **TileLayers** 컬렉션에 나타나는 순서대로 렌더링 된다. 첫 레이어는 배경에 있는 레이어다. **AboveVectorMap** = **False** 를 설정함으로 레이어가 벡터 지도 보다 먼저 렌더링 된다. (5.26 장에 명시된 대로) 정의 되었을 때다. 기본적으로 **TileLayer** 가 벡터 지도 이후에 렌더링 된다.

TileLayer 은 온라인 서비스 제공자에서 http 프로토콜을 통해 정보를 작은 이미지로 받아 차트 구역에 보여준다. 지도 뷰를 줌 또는 패닝할 때 새로 고침 된다.

새로운 타일 세트를 로딩하는 데에는 몇 초까지 걸리는 시간이 소요된다.

타일 캐시

차트는 타일을 캐시 폴더에 저장한다. 이는 같은 구역에 자주 패닝 또는 줌할 때에 로딩 시간을 상당히 감소한다. 차트가 타일을 보여줘야 할 때 이는 캐시 폴더에서 찾을 수 있는지 먼저 확인한다. 찾을 수 없을 때 웹 서비스에서 불러 온다. 팀 사용 환경에서 여러 워크 스테이션이 타일 지도를 접근 해야 할 때 공유 로컬 네트워크 서버 폴더를 선택하는 것을 추천한다. 기본적으로 캐시 폴더 경로는 **c:\Users\[Current user]\AppData\Local\Temp** 이다.

ViewXY.Maps.TileCacheFolder 내에 캐시 폴더를 설정 하라.

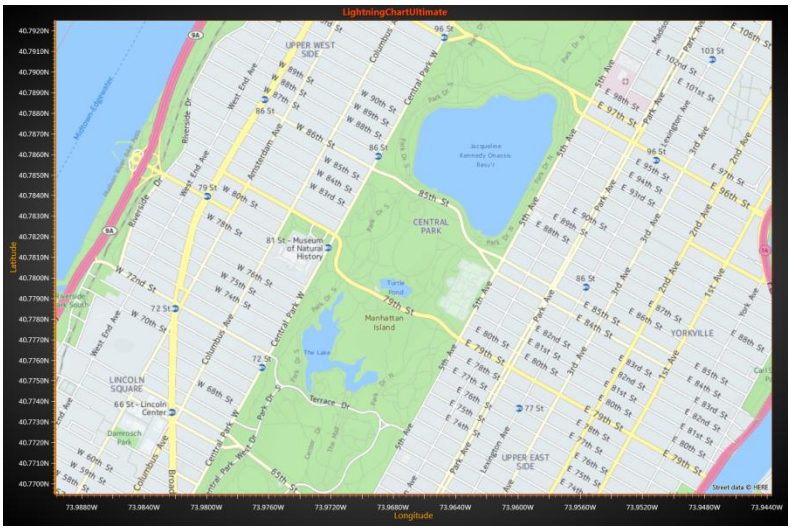
ViewXY.Maps.ClearTileCacheFolder() 메소드를 불러 캐시 폴더를 비워라.

5.27.1 HERE

라이트닝차트는 Here 타일 데이터 서비스를 지원한다. 개발자 또는 끝 사용자가 Here 와 계약을 만들어 Here 서버를 사용해야 한다. https://developer.here.com/plans/api/consumer-mapping 에서 공짜 트라이얼 키를 얻을 수 있다.

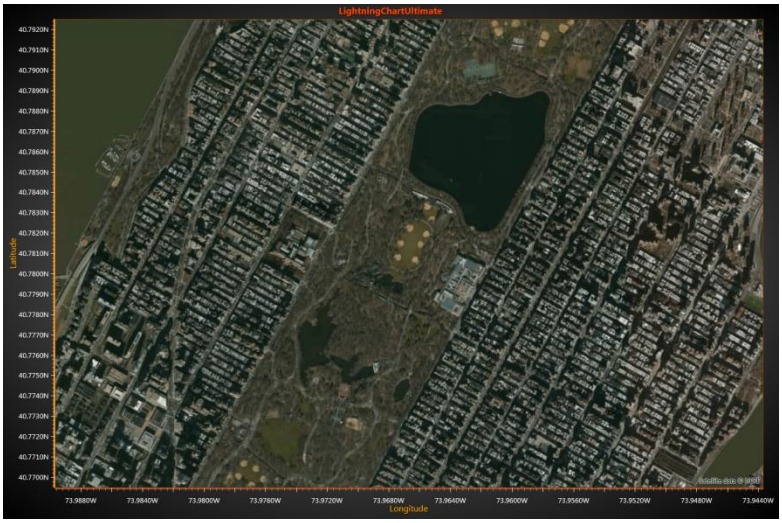
종류 선택

거리 지도 사용하기 위해 **TileLayer.Type = Street** 로 설정하라. 거리 지도는 아주 가깝게 확대 가능하다.



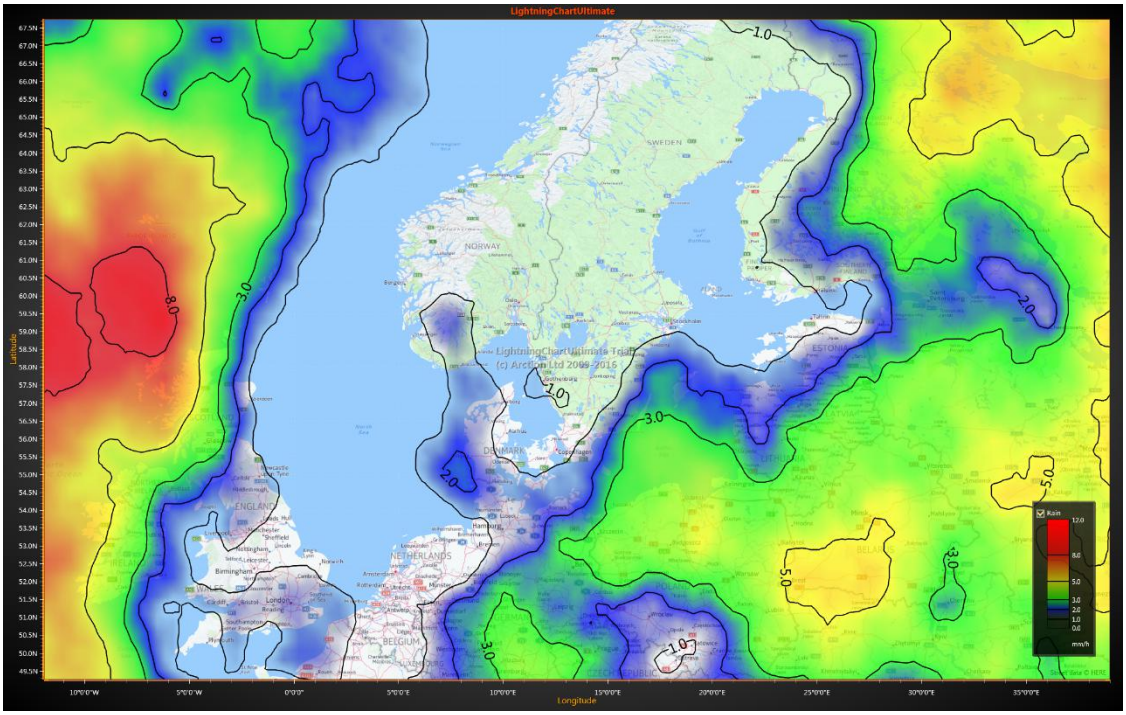
보기 5- 116. TileLayer.Type = Street.

위성 이미지 사용 하기 위해 **TileLayer.Type = Satellite** 설정하라.



보기 5- 117. TileLayer.Type = Satellite.

시리즈 및 주석 등 기타 차트 요소 표기 가능하다.



보기 5- 118. 날씨 데이터를 표시하는 IntensityGridSeries 가 있는 거리 지도.

5.28 StencilAreas

데모 예시: 강도 시리즈 스텐실 있는 지도; 색도도, 실리콘 웨이퍼 지도 분석 (윈폼만 해당)

IntensityGridSeries, **IntensityMeshSeries** 및 **Maps** 는 **StencilArea** 특징이 있어 그려진 데이터 구역들을 마스킹 할 수가 있다. 예를 들어 데이터가 지도 위 보여진다면 스텐실을 사용해 보여지는 데이터를 나라 등 특정 지도 구역들로 제한할 수 있다. **StencilArea** 객체를 생성 후 **AddPolygon()** 를 통해 크기를 PointDouble2D -배열로 정의하거나 **AddMapLayerIndex()** 를 통해 지도 레이어로 설정 후 가릴 시리즈에 추가하여 **StencilArea** 를 적용할 수 있다.

두 종류의 **StencilAreas** 가 있다:

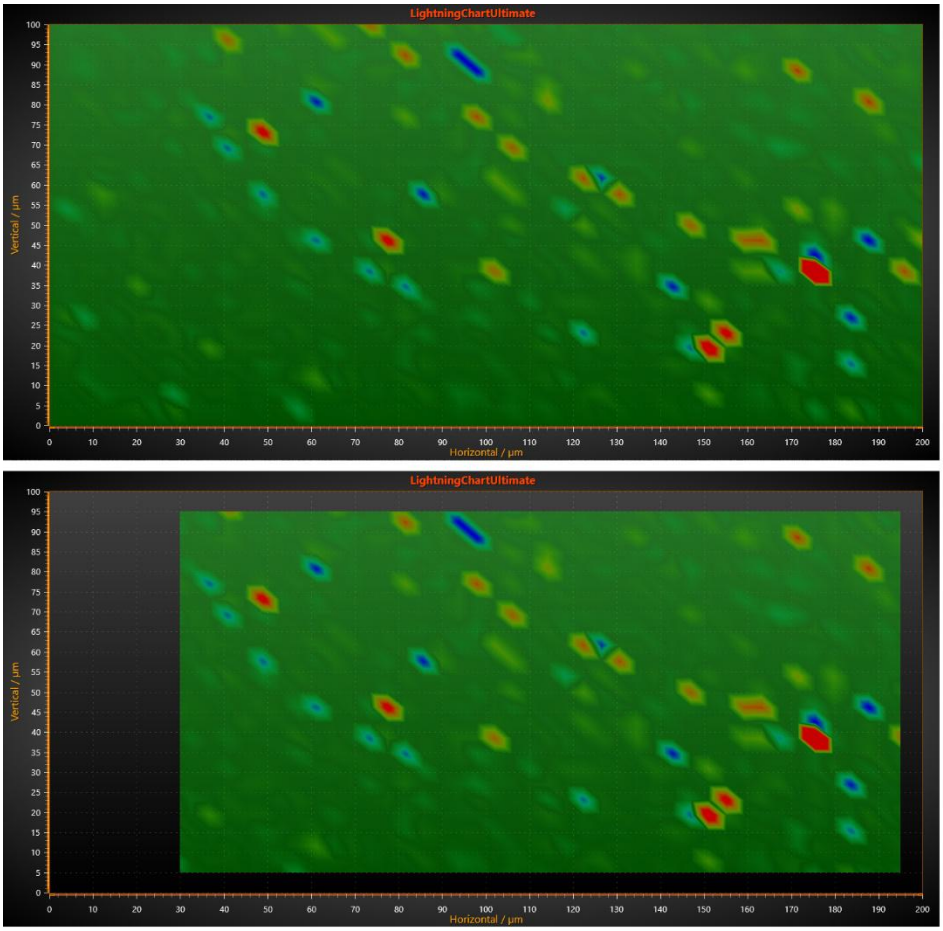
- **AdditiveAreas** 는 양성의 스텐실 마스크를 생성한다. 오로지 구역 안의 데이터만 그려지고 구역 밖의 데이터는 클리핑 된다.
- **SubtractiveAreas** 는 음성의 스텐실 마스크를 생성한다. 구역 내 데이터는 클리핑 되고 구역 외 데이터는 그려진다. **SubtractiveAreas** 는 **AdditiveAreas** 와 함께 적용 되게 설정 되어있다. 같이 사용하지 않을 시 클리핑이 적용되지 않는다.

(**AdditiveAreas** 또는 **SubtractiveAreas**) 목록에 **StencilArea** 객체가 추가 될 때마다 해당 시리즈에 **InvalidateStencil()** 또는 **InvalidateData()** 를 불러야 한다. 시계방향으로 스텐실을 정의하는 포인트 배열을 설정하는 것을 추천한다.

5.28.1 AdditiveAreas

AdditiveAreas 를 사용해 그려질 구역을 정의하라. 그 외 구역들은 모두 클리핑이 된다.

```
// Defining an additive StencilArea to an IntensityGrid

PointDouble2D[] stencilPoints = new PointDouble2D[] {
     new PointDouble2D(30, 5),
     new PointDouble2D(30, 95),
     new PointDouble2D(195, 95),
     new PointDouble2D(195, 5)
};
StencilArea stencilArea = new StencilArea(_intensityGrid.Stencil);
stencilArea.AddPolygon(StencilPoints);
_intensityGrid.Stencil.AdditiveAreas.Add(stencilArea);
```



보기 5- 119. 위에는 스텐실 없는 IntensityGrid. 아래에는 같은 그리드지만 코드로 AdditiveArea 생성됨.

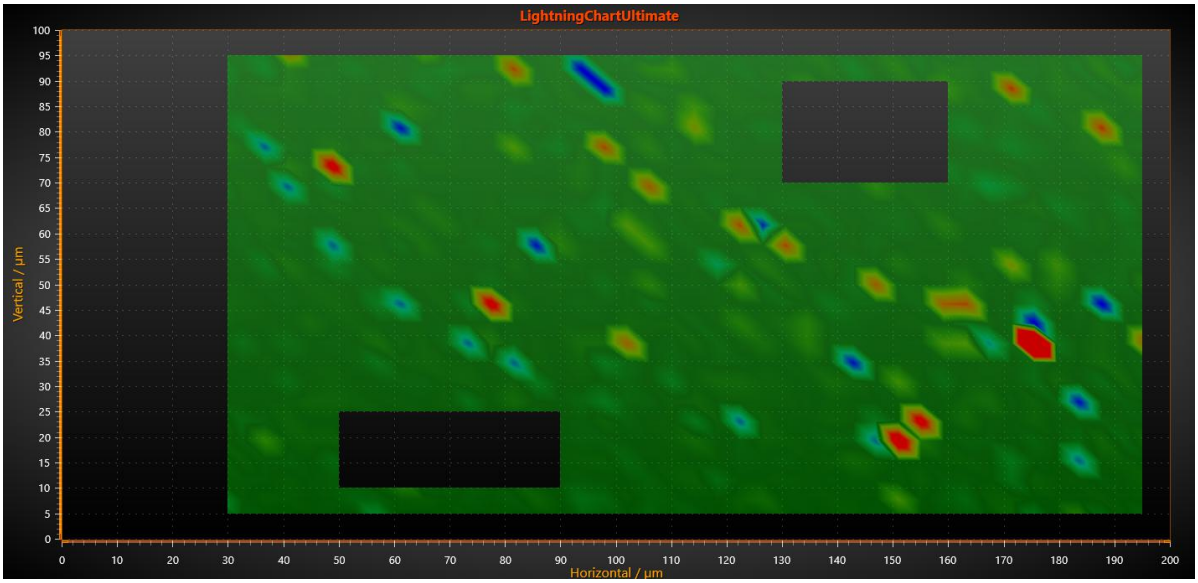
```
_intensityGrid.InvalidateStencil();
```

5.28.2 SubstractiveAreas

SubstractiveAreas 를 이용해 **AdditiveArea** 안에 그려지지 않아야 할 구역들을 정의하라.

```
// Defining two substractive StencilAreas to an IntensityGrid

PointDouble2D[] pnt2 = new PointDouble2D[] {
     new PointDouble2D(130, 70),
     new PointDouble2D(130, 90),
     new PointDouble2D(160, 90),
     new PointDouble2D(160, 70),
};
StencilArea stencilArea2 = new StencilArea(_heatMap.Stencil);
stencilArea2.AddPolygon(pnt2);
_heatMap.Stencil.SubtractiveAreas.Add(stencilArea2);
_heatMap.InvalidateStencil();

PointDouble2D[] pnt3 = new PointDouble2D[] {
     new PointDouble2D(50, 10),
     new PointDouble2D(50, 25),
```

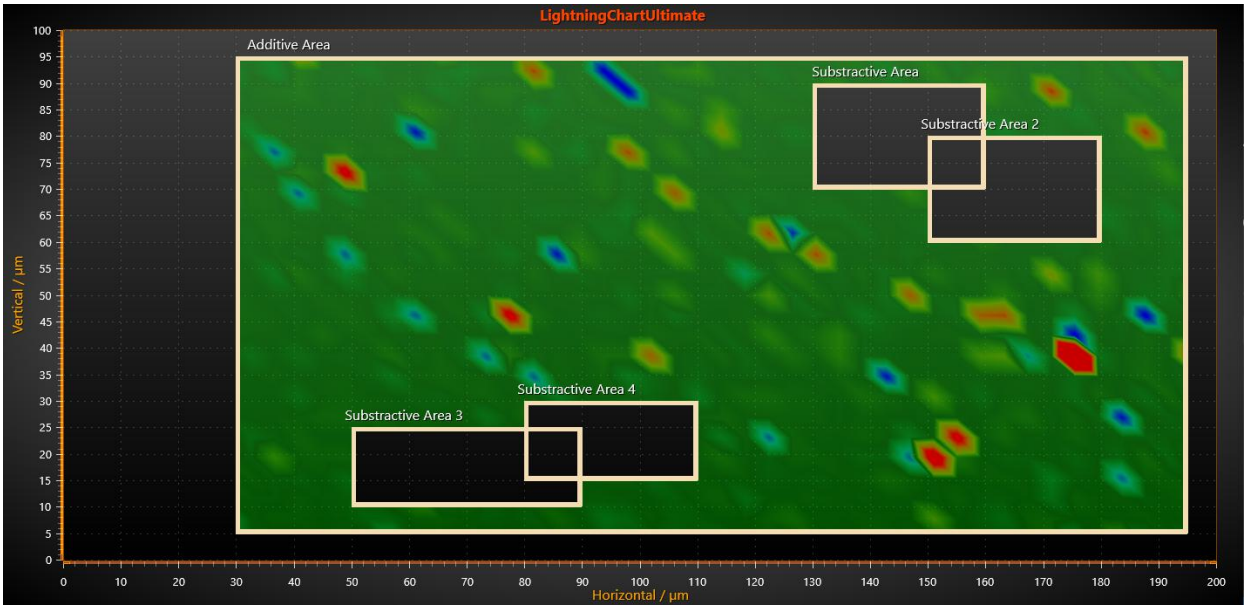
```
        new PointDouble2D(90, 25),
        new PointDouble2D(90, 10),
};
StencilArea stencilArea3 = new StencilArea(_heatMap.Stencil);
stencilArea3.AddPolygon(pnt3);
_heatMap.Stencil.SubtractiveAreas.Add(stencilArea3);
_heatMap.InvalidateStencil();
```



보기 5- 120. 두개의 SubstractiveAreas 가 설정 된 IntensityGrid. SubstractiveAreas 사용 전 AdditiveArea 먼저 설정해야 한다.

5.28.3 여러 개의 StencilAreas

여러 additive 및 substractive **StencilAreas** 설정 가능하다. 두개 이상의 구역들이 겹치는 경우에는 구역들이 이어진다.



보기 5- 121. 여러 StencilAreas 가 사용 됐다. 몇 SubstractiveAreas 가 겹쳐 합쳐졌다. 보이는 태두리가 있는 투명 다각형도 그려져 스텐실의 위치가 표시 된다.

5.29 LineSeriesCursors

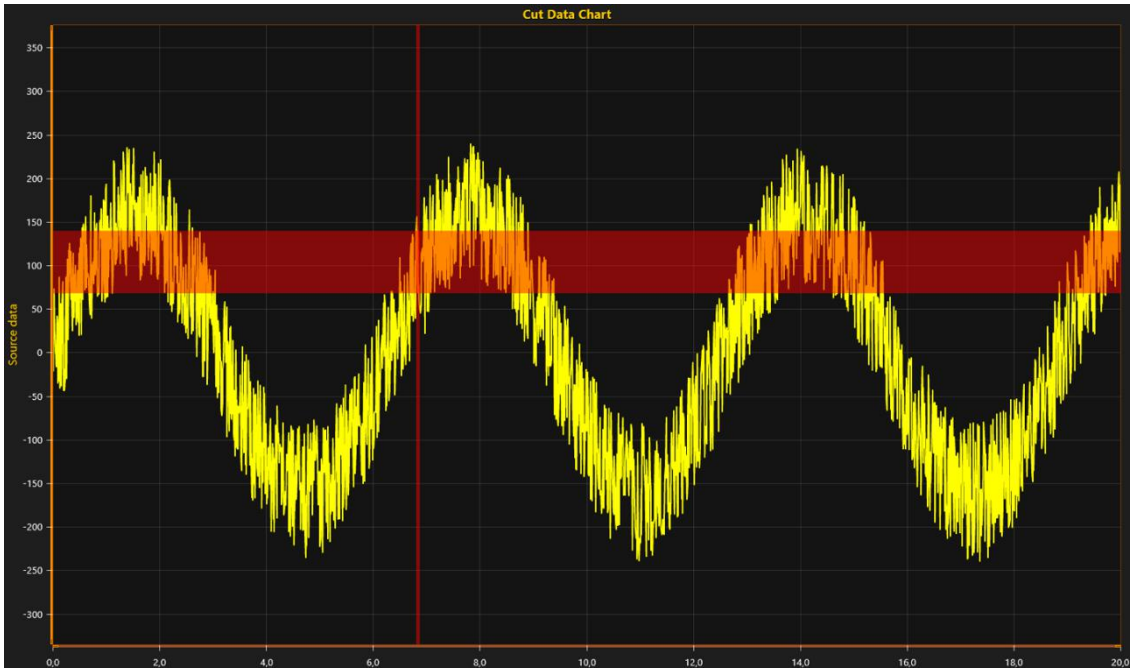
데모 예시: 포인트 라인; 멀티 채널 커서 트래킹; 스플리터 있는 세그먼트; 로그 축

라인 시리즈 커서는 X 좌표의 값을 트래킹 해서 라인 시리즈 데이터의 시각화 분석을 허용한다. 시리즈 값은 **ITrackable** 인터페이스를 적용해서만 해결 가능하다 (**SampleDataSeries**, **PointLineSeries, AreaSeries, HighLowSeries**). 다른 시리즈 종류에는 Y 좌표가 커서 자동 트래킹이 되지 않는다.

LineSeriesCursor 컬렉션에 **LineSeriesCursors** 객체를 추가하라. **SnapToPoints** 를 활성화 해 커서를 포인트에서 포인트로 바로 가게 설정하라. 커서 트래킹 스타일을 **Style** 속성으로 설정하라. **Style** 이 **PointTracking** 으로 설정 되었을 때 아무 트래킹 포인트 스타일을 사용할 수 있다. 비트맵 이미지도 사용 가능하다. **HairCrossTracking** 사용할 때 라인 시리즈 Y 값 포인트에 수평선이 그려진다. 커서 위치에 같은 시리즈에 여러 포인트가 히트 되면 최소 및 최대 포인트에 라인이 그려진다.

보기 5-121. 라인 시리즈 커서: 수직 포인트 트래킹 커서 (청록 선 및 x 표), 십자선 트래킹 커서 (빨간 선) 및 트래킹 없는 수직 런체 길이 커서 (노랑 선)

IndicateTrackingYRange 을 활성화 함으로 최소 포인트에서 최대 포인트까지 커서 가운데를 맞추며 수평 선이 그려진다.



5.29.1 LineSeriesCursor 위치에 있는 데이터 값 풀기

데모 예시: 멀티 채널 커서 트래킹

ITrackable 인터페이스를 적용하는 시리즈는 X 화면 좌표 또는 X 축 값으로 풀 수 있다.

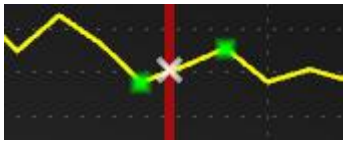
트래킹 가능한 시리즈는 정확하고 조잡한 값 풀 수 있는 메소드를 갖고 있다. 정확한 **SolveYValueAtXValue** 메소드는 필요하면 데이터 포인트를 루핑해 제일 가까운 일치 데이터 포인트를 찾는다. 조잡한 **SolveYCoordAtXCoord** 메소드는 시리즈의 캐시 렌더링 데이터를 사용해 일치하는 Y 화면 좌표를 푼다.

5.29.1.1 정확한 방식, 데이터 포인트 배열 X 값으로 Y 값 풀기

```
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;
}
```

주의! cursor.SnapToPoints 가 비활성화 되었을 때 **SolveYValueAtXValue** 는 양 옆 포인트의 보간 값을 돌려준다 (커서 선 및 시리즈 라인의 교차점).



보기 5-123. SolveYValueAtXValue 는 SnapToPoints 가 비활성화 되었을때 양 옆 데이터 포인트 사이 값을 보간한다.

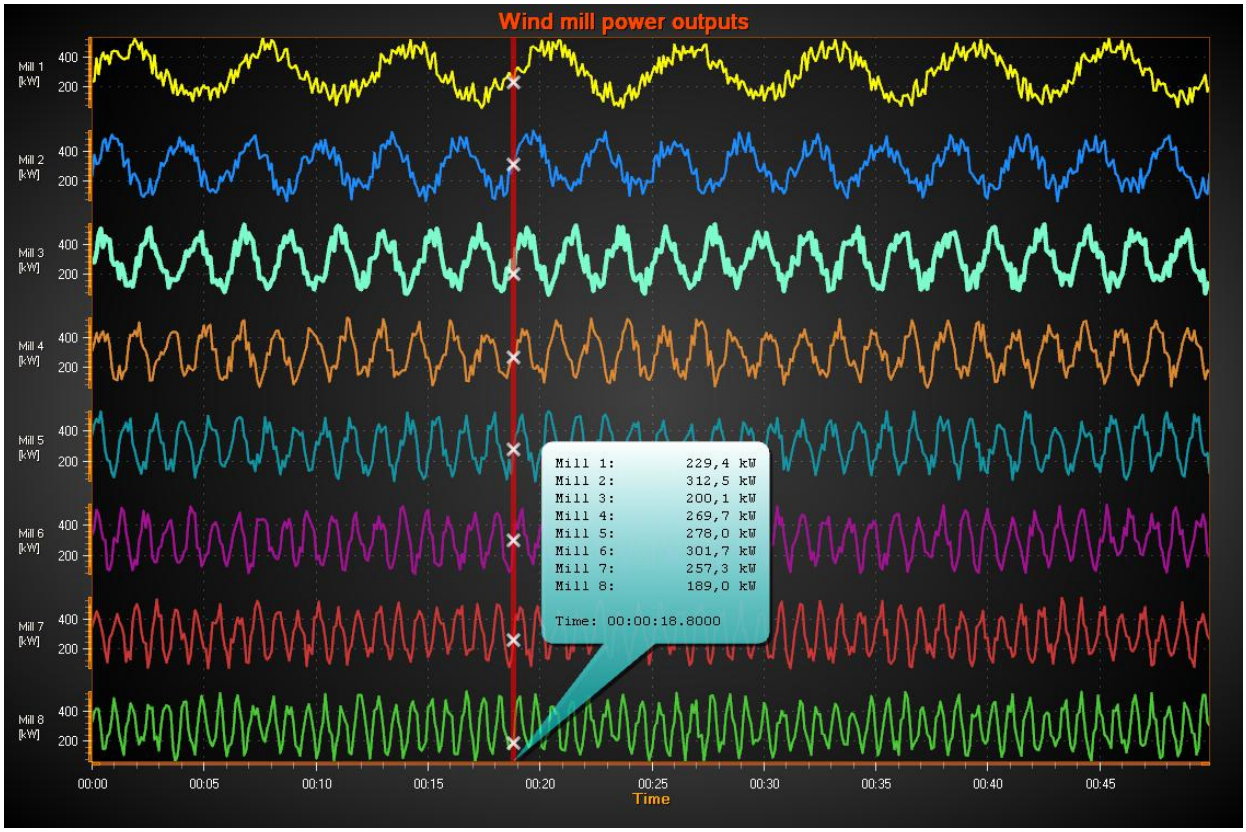
5.29.1.2 조잡한 방식, 데이터 포인트 배열 사용한 X 좌표로 Y 화면 좌표 풀기

```
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;
      }
}
```

시리즈가 100.000 개 이상의 데이터 포인트를 갖고 있을 때 조잡 방식을 사용 하는 것이 주로 훨씬 빠르다. 조잡 방식은 차트 크기가 또는 Y 세그먼트 높이의 픽셀 수가 낮으면 매우 부정확 할 수 있다.

조잡 방식의 화면 좌표는 축 값으로 변경 가능하다. X 및 Y 축의 **CoordToValue** 를 불러 가능하다.

AnnotationXY 객체를 이용해 커서 옆에 값을 표시하는 것이 좋다, 5.20 을 보아라.



보기 5-124. PointLineSeries 를 트래킹 하는 데에 사용 된 LineSeriesCursor. 값은 AnnotationXY 객체를 통해 보여졌다.

5.29.2 FreeformPointLineSeries 에서 데이터 값 풀기

FreeFormPointLineSeries 은 위와 같은 메소드를 사용 한 데이터 값 풀기 위한 정확 및 조잡 값 풀기 메소드를 갖고 있다.

주의! FreeformPointLineSeries 는 **ITrackable** 인터페이스를 적용하지 않기에 **LineSeriesCursor** 로 트래킹 할 수가 없다.

정확한 **SolveYValuesAtXValue** 메소드는 주어진 X 축 값의 모든 Y 값을 풀고 반복 가능한 **LineSeriesValueSolveResult** 구성을 돌려준다.

```
IList<LineSeriesValueSolveResult> results = series.SolveYValuesAtXValue(value);
foreach (LineSeriesValueSolveResult result in results)
{
    if (result.SolveStatus == LineSeriesSolveStatus.OK)
    {
        // Do.
    }
    else if (result.SolveStatus == LineSeriesSolveStatus.NoPointsFound)
    {
        // Do.
    }
}
```

조잡한 **SolveYCoordsAtXCoord** 메소드는 주어진 X 화면 좌표에 의한 모든 Y 화면 좌표를 풀고 반복 가능한 **LineSeriesCoordinateSolveResult** 구성 목록을 돌려준다.

```
IList<LineSeriesCoordinateSolveResult> results = series.SolveYCoordsAtXCoord(fCoordX);
foreach(LineSeriesCoordinateSolveResult result in results)
{
    if(result.SolveStatus == LineSeriesSolveStatus.OK)
    {
        // Do.
    }
    else if(result.SolveStatus == LineSeriesSolveStatus.NoPointsFound)
    {
        // Do.
    }
}
```

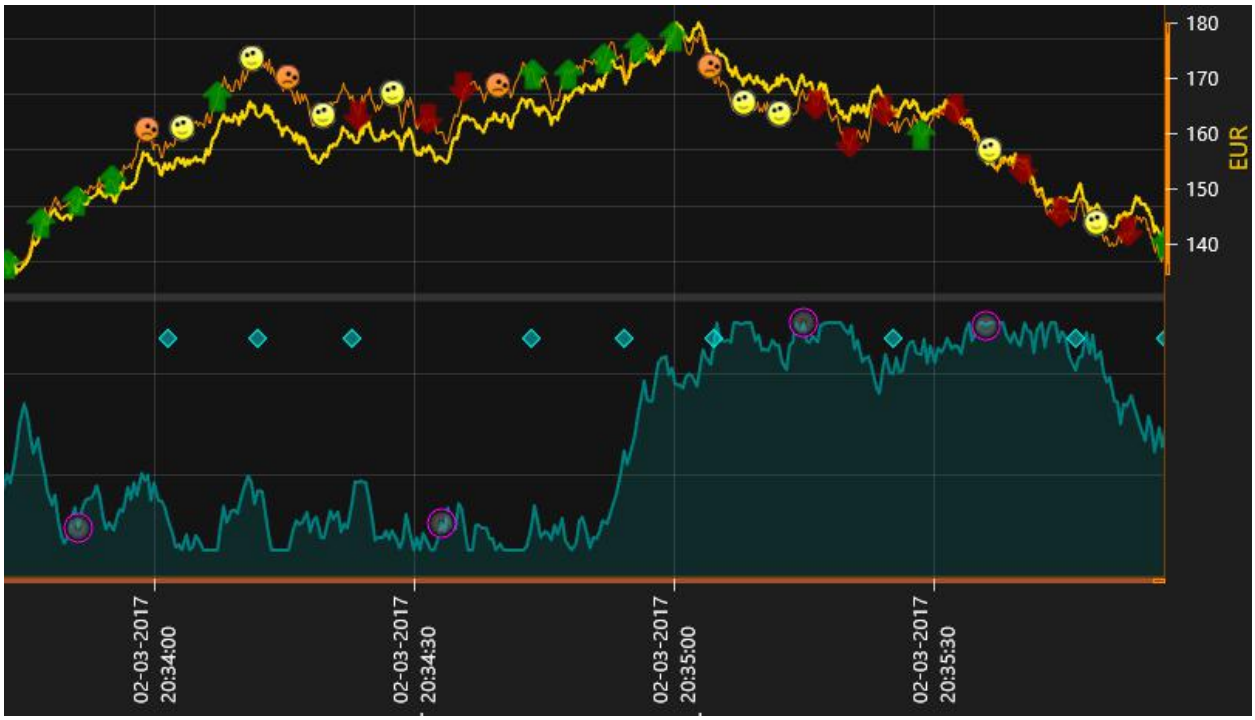
주어진 X 축 값 또는 x 화면 좌표가 포인트 사이 히트 치면 결과는 값 / 좌표 사이에서 보간 된다.

5.30 EventMarkers

데모 예시: 트래킹 마터; 지도 경로; 열지도 컬러 스프레드; 스플리터 있는 세그먼트 ; 버블 표; 캠벨 도표; 커브 노드 수정

EventMarkers 관심 포인트의 마킹을 허용한다. 실시간 모니터링 도중 특별한 이벤트가 될 수도 있고 단지 특별 주석으로 데이터를 표시 하고 싶을 때 사용 가능하다. 표시 기호를 **Symbol** 속성으로 정의하고 **Label** 속성으로 텍스트 라벨을 정의하라. 수직 위치를 **VerticalPosition** 속성으로 설정하고 **Offset** 을 이용해 필요 대로 객체를 움직여라. 모든 표시는 **XValue** 로 할당 되어야 하고 이는 X 에 마커의 위치를 설정한다.

마커의 모양을 선택하기 위해 **Symbol.Shape** 를 설정하라. 선택 가능한 모양은 **직사각형, 원, 세모, 긱발, FlagLightning, 십자, 십자수, 비트맵, HollowBasic**, **HollowBasicActive**, **HollowHarmonic**, **HollowActiveSideband, HollowSideband**, **HollowTailedActive** 및 **HollowTailed.**



보기 5-125. 트레이딩 예시에 마커 사용.

5.30.1 차트 이벤트 마커

ChartEventMarker 컬렉션은 차트 마커 추가를 허용한다. 차트 마커는 관심 포인트를 표시할 때 사용 가능하다. 예를 들어 "실험 대상이 일어 섰다", "커패시터 우회" 등이 있다. 시리즈 이벤트 마커와는 다르게 차트 이벤트 마커는 특별 시리즈에 묶여 있지 않다. 마커를 마우스로 다른 위치에 드래그 가능하다.

차트 마커의 위치를 **VerticalPosition**, **XValue** 및 **Offset** 속성들로 설정 가능하다. **BindToXAxis** 를 참으로 설정은 마커는 특별 x 축으로 묶기 가능하다. 이는 마커가 현 x 값 위치에 머물게 하며 x 축과 함께 움직이게 한다. **BindToXAxis** 가 비활성화 되었을 때 마커는 축이 아무리 움직여도 같은 차트 위치에 유지 되게 한다. 여러 x 축이 있으면 **AssignXAxisIndex** 를 사용해 마커가 어느 축에 묶여 있는지 설정 가능하다.

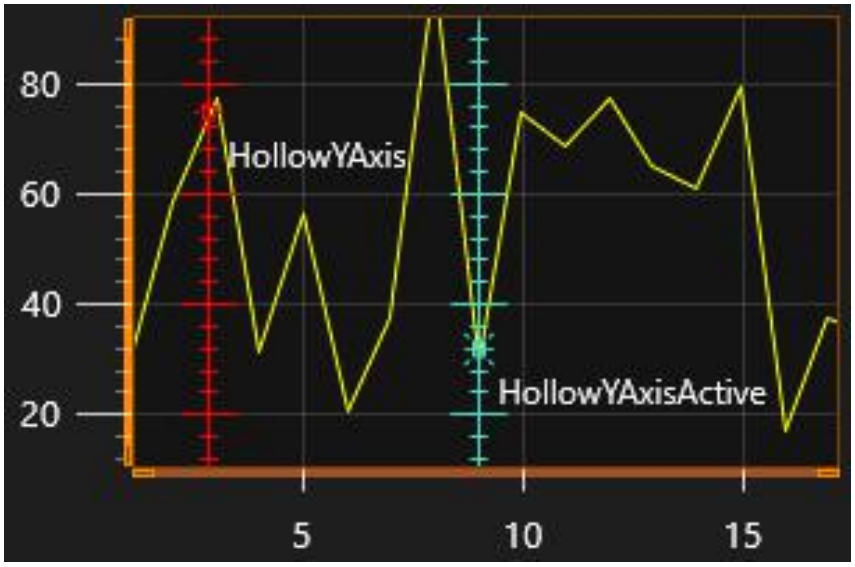
ClipInsideXRange 를 **BindToXAxis** 활성화 된 채로 같이 사용 가능하다. 이는 마커가 보이지 않는 x 축 범위에 특수 x 축 값에 묶여 있을 때 클립 되게 만든다. 이 값은 **XValue** 속성을 통해 설정 가능하다. 마커를 마우스로 수동적으로 움직이는 것은 이 설정을 비활성화 한다. **ClipInsideGraph** 속성도 있다. 이는 차트 마커가 그래프 구역 밖에 그려질 수 있는지를 결정한다.

5.30.2 라인 시리즈 이벤트 마커

라인 시리즈는 **SeriesEventMarkers** 컬렉션 속성을 갖고 있다. 이는 시리즈 별로 이벤트 마커를 할당 하는데에 사용할 수 있다. 시리즈 이벤트 마커는 시리즈 값에 이벤트 마커를 묶은 채 다른 위치로 마우스 드래그 가능하다. 이를 활성화 하기 위해 마커의 **VerticalPosition** 이 **TrackSeries** 로 설정해야 한다. **ITrackable** 인터페이스를 적용하는 모든 시리즈에 사용 가능하다.

HorizontalPosition 를 **SnapToPoints** 로 설정 함으로 마쿠가 제일 가까운 데이터 포인트로 수평 정렬 하게 된다. **HorizontalPosition** = **AtXValue** 은 마커를 아무 x 값에 놓는 것을 허용한다. 비슷하게 **VerticalPosition** = **AtYValue** 은 마커가 아무 y 레벨에 수직 설정이 가능하게 한다.

보통 모양 세트에 이어 **SeriesEventMarker** 는 두 특별 **Symbol.Shape** 설정을 지원한다: **HollowYAxis** 및 **HollowYAxisActive.** 이는 Y 축 티크 프로젝션 있는 수직 선을 허용한다. 이들은 픽셀 넓이의 수직 선이 있어 묶인 시리즈이 Y 축 **MajorTicks** 및 **MinorTicks** 에서 위치를 고른다. 티크 길이를 수정하기 위해 **YAxis.MajorDivTickStyle.LineLength** 및 **YAxis.MinorDivTickStyle.LineLength** 속성들을 수정하라.



보기 5-126. 두 특별 SeriesEventMarkers 모양: HollowYAxis 및 HollowYAxisActive. 시리즈 별 데이터 커서를 만드는 데 유용하다.

5.31 지속 시리즈 렌더링 레이어

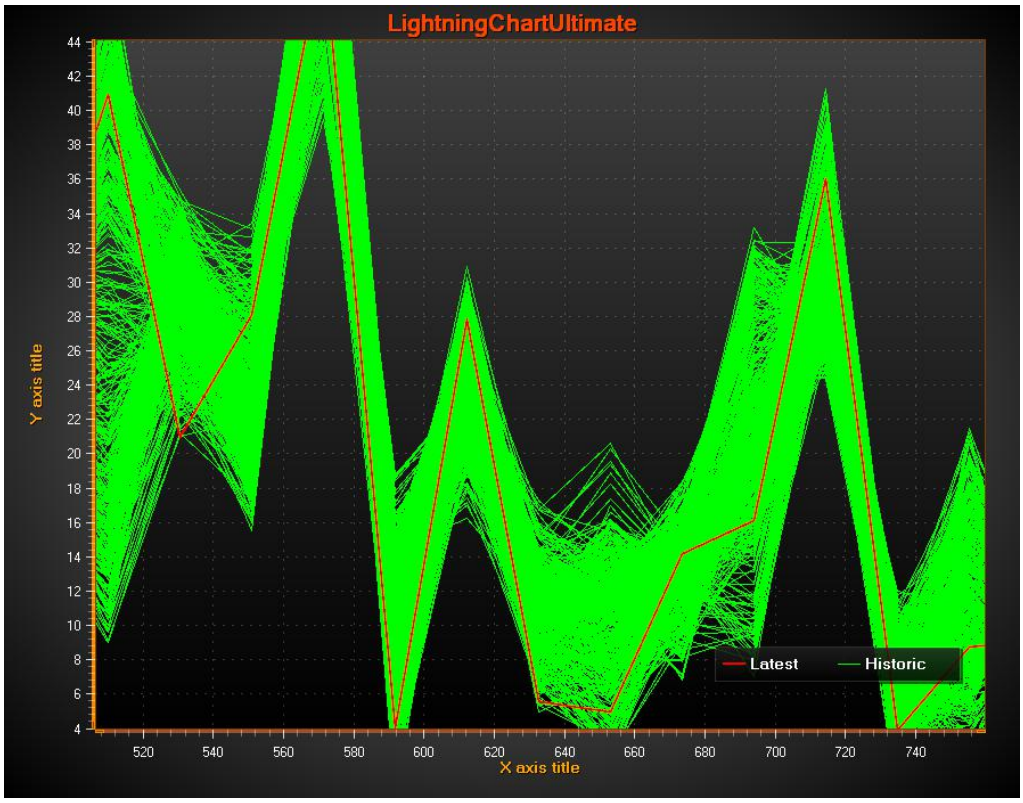
데모 예시: 라인 / 포인트; 구역 / 하이-로우

PersistentSeriesRenderingLayer 는 같은 X 및 Y 범위에 계속되어 반복 되는 라인/포인트 데이터 또는 라인/포인트/하이-로우/구역 필 데이터를 아주 빠르게 렌더링할 때 사용 가능하다.

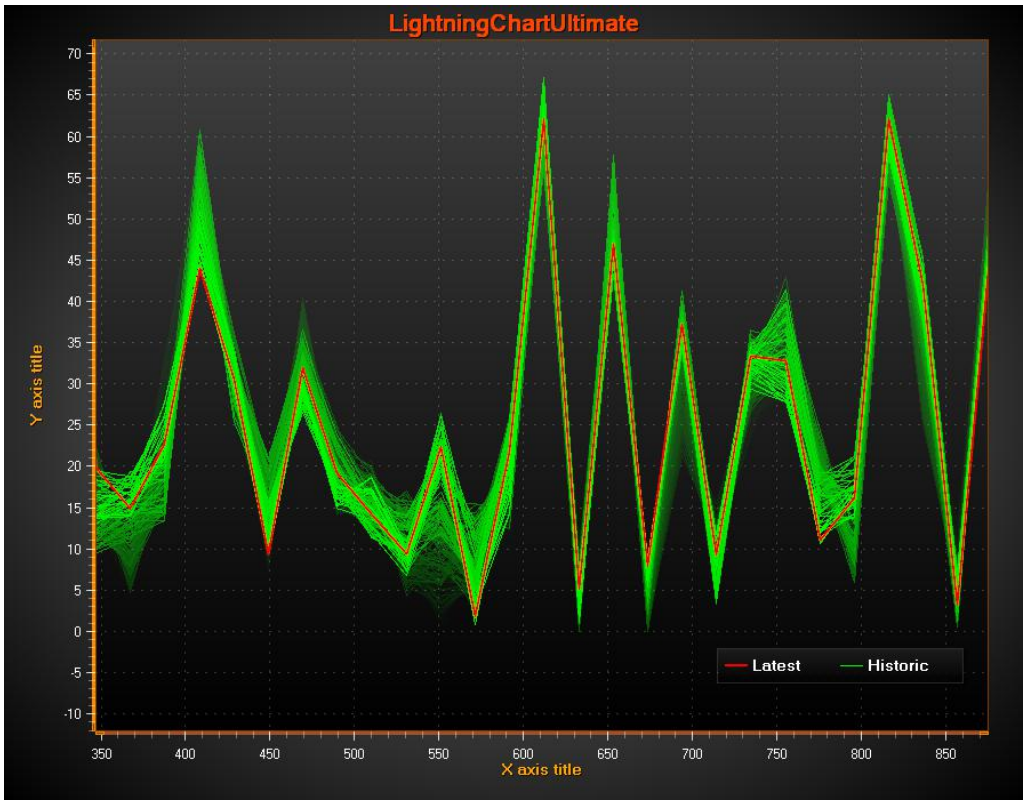
예를 들어 FFT 모니터링 사유를 보아라. 초당 20 개의 새로운 데이터 스트립을 받는다. 이전 데이터와 제일 최신 데이터가 같이 보여져야 한다. 하지만 모니터링은 몇 시간동안 지속 된다. 일반 렌더링으로 이런 데이터를 렌더링 돌리면 시간당 20 * 60 * 60 = 72000 개의 새로운 라인 시리즈가 필요하다. 컴퓨터가 1 시간의 모니터링이 끝나기 전 저장 공간이 부족할 것이다. 렌더링이 사용 못할 정도로 느리게 만들 것이다.

PersistentSeriesRenderingLayer 는 비트맵의 종류이다. 렌더링 데이터를 조금씩 추가한다. 이는 커맨드로 지우기 전까지 그래픽을 유지한다. 이렇게 해 매 업데이트마다 화면에 있는 레이어 렌더링이 레이어에 한 시리즈만이 렌더링 따라 된다. CPU 로드 또는 메모리 발자국이 오르지 않는다. 존재하는 데이터가 천천히 사라져야 하면 비트맵 픽셀의 앞라를 곱하여 이룰 수 있다.

필요한 대로 **PersistentSeriesRenderingLayer** 객체를 계속 생성할 수 있다. 각 객체에 아무 시리즈 수 만큼 아무 업데이트 라운드 마다 렌더링 가능하다.



보기 5-127. 지속 레이어가 이전 데이터를 초록색으로 보여주고 있다. 위에 일반 PointLineSeries 가 빨간색으로 보여지고 있다.



보기 5-128. 지속 레이어가 이전 데이터의 흔적을 초록색으로 보여주고 있다. 새로운 데이터를 렌더링 레이어에 업데이트 전 MultiplyAlpha 메소드를 불러 제일 오래 된 흔적이 사라지게 하고 있다.

5.31.1 레이어 생성

PersistentSeriesRenderingLayer 는 ViewXY 의 하위 속성이 아니어서 비주어 스튜디의 속성 그리드로 추가할 수가 없다. **PersistentSeriesRenderingLayer** 객체들은 코드로 생성해야 한다. 다음과 같이 생성하라:

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

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

ViewXY 객체를 매개 변수로 공급함으로 레이어를 ViewXY 에 묶는다. 시리즈가 렌더링 된 같은 XAxis 객체를 공급하라.

여러 레이어가 차트와 함께 생성 된 순서 대로 렌더링 될 것이다.

5.31.2 레이어 비우기

layer.Clear() 는 레이어를 비우고 색을 ARGB=(0,255,255,255) 로 초기화 한다.

layer.Clear(Color color) 는 주어진 색으로 레이어를 비운다. 대부분의 경우에는 배경에 쓰여진 색과 같은 색으로 설정하는 것이 제일 유용하지만 A = 0 으로 설정하라. 검정 배경에는 layer.Clear(Color.FromArgb(0,0,0,0)) 을 사용하라.

5.31.3 레이어 알파 수정

MultiplyAlpha(value) 를 이용해 레이어를 더욱 투명 또는 불투명하게 만들수 있다. 곱하는 것은 레이어의 모든 픽셀에 따로 적용 된다.

<1 의 값을 제공 함으로 더욱 투명해 진다 (레이어를 부패).

>1 의 값을 제공 함으로 더욱 불투명해 진다.

1 의 값이 주어지면 아무 효과 없다.

예를 들어, **MultiplyAlpha(0.8)** 은 알파를 존재 알파의 80%로 설정한다. **MultiplyAlpha(2)** 는 200%로 수정한다.

5.31.4 레이어에 데이터 렌더링

PointLineSeries, **SampleDataSeries**, **FreeformPointLineSeries, HighLowSeries 또는 AreaSeries 객체로** 데이터를 레이어 안으로 렌더링하라. **ViewXY.PointLineSeries**, **ViewXY.SampleDataSeries**, **ViewXY.FreeformPointLineSeries, ViewXY.HighLowSeries** 또는 **ViewXY.AreaSeries** 컬렉션에 추가 된 시리즈여도 된다. 이 컬렉션에 추가 되지 않은 일시적인 시리즈도 사용 가능하다. 원래대로 이 시리즈에 데이터를 채워라 (**PointLineSeries** 를 위해서는 6.6.4 를 보고 **SampleDataSeries** 를 위해서는 6.7.2, **FreeformPointLineSeries** 를 위해서는 6.8, **HighLowSeries** 를 위해서는 6.10.4 및 **AreaSeries** 위해서는 6.11.1 을 보아라).

layer.RenderSeries(PointLineSeriesBase series): 레이어에 한 시리즈 렌더링.

layer.RenderSeries(List<PointLineSeriesBase> seriesList): 주어진 시리즈 모두 레이어에 렌더링. 각 시리즈 따로 **layer.RenderSeries(PointLineSeriesBase series)** 를 부르는 것 보다 더욱 효율적이다.

주의! 모든 주어진 시리즈가 레이어에 렌더링 된다. Visible 이 False 로 설정 되어도 된다.

주의! 시리즈와 사용 된 x 축은 PersistentSeriesRenderingLayer 컨스트럭터와 제공된 것과 일치해야 한다. 아닐 시 시리즈 객체를 건너 뛴다.

주의! RenderSeries 레이어 안으로 렌더링할 때 위함이다. 레이어 자체가 일반 시리즈 (PointLineSeries, SampleDataSeries, FreeformPointLineSeries, HighLowSeries, AreaSeries) 직전에 렌더링 된다.

5.31.5 레이어 제거

레이어를 제거하고 차트와 함께 렌더링 되는 것을 방지 하기 위해 **layer.Dispose()** 를 불러라.

5.31.6 레이어 내 데이터 앤티 앨리어싱

데이터를 차트 렌더링 단계에 앤티 앨리어싱 하기 위해 **layer.AntiAliasing** 를 **True** 로 설정 하라. 이는 하드웨어가 지원하지 않아도 앤티 앨리어싱을 활성화 한다.

5.31.7 레이어 목록 불러오기

ViewXY.GetPersistentSeriesRenderingLayers() 은 **PersistentSeriesRenderingIntensityLayers** 를 포함한 생성된 모든 레이어의 목록을 불러온다.

5.31.8 주의 해야할 레이어 제한

특별한 렌더링 기술 덕에 다음과 같은 제한들을 주의하라:

- X 축 **ScrollMode** 가 **None** 으로 설정 되어야 한다. X 축의 실시간 스크롤링은 이 방식으로 불가능하다.

- 줌, 패닝, 축 수정 및 차트 크기 재설정은 이미지가 축 범위들과 동기화를 해제 시킨다. 이 특징들은 지속 플롯 사용할 때 비활성화 하던가 새로운 레이어 렌더링을 위한 응용 프로그램 로직을 레이어를 비우고 구 라인 시리즈를 잠시 재구현 하도록 설정해야 한다 (축 범위 변경 및 크기 재설정을 위한 이벤트 핸들러가 존재한다).

- 차트 크기 재설정은 레이어를 비운다. 윈도우 데스크탑 잠금 상태에서 다시 재생하는 것도 비운다.

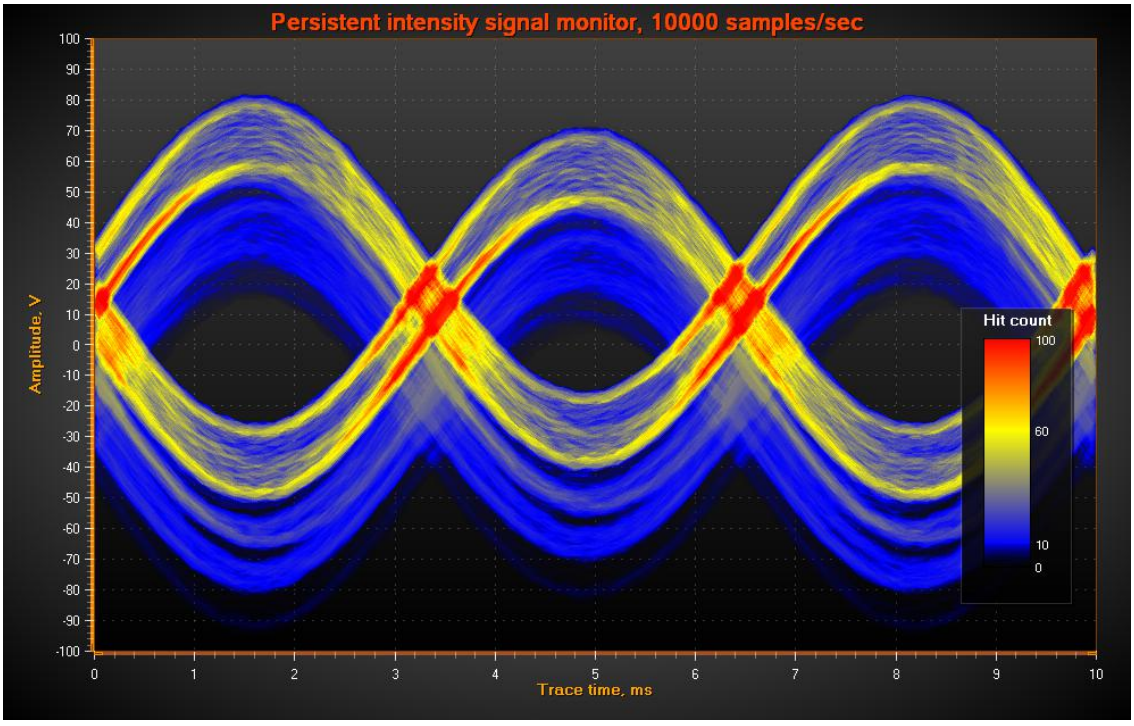
- 마우스 대화식은 레이어에만 렌더링 된 시리즈에 지원 되지 않는다.

- EMF/WMF/SVG 수출, 벡터 형식으로 클립보드에 복사, 및 벡터 형식으로 프린트는 레이어를 지원하지 않느다. 라스터 형식만이 지원된다.

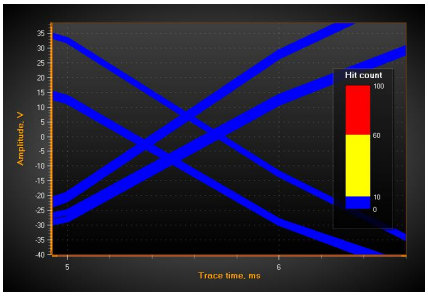
5.32 강도 레이어 지속 시리즈 렌더링

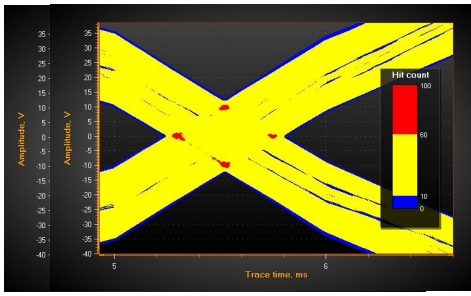
데모 예시: 강도 지속 레이어, 시그널

PersistentSeriesRenderingIntensityLayer 흔적을 레이어로 모아 픽셀 당 힛 카운트 대로 색 칠한다. 색칠은 값 범위 팔레뜨는 이용한다. 흔적들은 **PersistentSeriesRenderingLayer** 와 같은 시리즈 종류와 같이 사용 가능하다 (6.31 장을 보아라). 둘은 아주 비슷하며 색칠이 가장 큰 차이점이다. 두번째 렌더링 콜로 다시 픽셀 위치에 흔적을 렌더링 할때 이의 강도는 증가하여 값 범위 팔레뜨의 값 또한 증가 시킨다.



보기 5-129. 지속 강도 레이어가 활동이 집중된 구역을 하이라이트 한다. 이 경우에는 노랑색과 빨간색으로 칠한다.





보기 5-130. 반복적 시그널 흔적이 같은 구역에 렌더링 된다. 왼쪽에는 몇개만의 흔적이 레이어에 렌더링 되어 모든 색을 파랑으로 보인다. 가운데에는 많은 흔적들이 렌더링 되었지만 다 다른 좌표에 렌더링 되었다 흔적의 교집합에는 힛 수가 10 이상일 때 팔레뜨에서 노란색으로 정의 되어 있다. 맨 오른쪽 이미지에는 총 몇 백개의 흔적이 렌더링 되었다. 교집합이 빨간색으로 정의된 범위에 접근하기 시작한다.

5.32.1 레이어 생성

PersistentSeriesRenderingIntensityLayer 은 ViewXY 의 하위 속성이 아니라 비주얼 스튜디오의 속성 그리드에 추가할 수 없다. **PersistentSeriesRenderingIntensityLayer** 객체들은 코드로 생성해야 한다.

다음과 같이 생성하라:

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

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

5.32.2 레이어 비우기

layer.Clear() 는 레이어를 비우고 카운터를 리셋한다.

5.32.3 팔레뜨 색 변경

팔레뜨 종류 및 단계를 레이어의 **ValueRangePalette** 속성에 정의하라. **ValueRangePalette.Type = Gradient** 는 그라디언트 색을 만들고 **ValueRangePalette.Type = Uniform** 은 단색의 레이어 렌더링을 생성한다.

5.32.4 새로운 흔적의 강도 효과 변경 및 이전 흔적의 부패

NewTraceIntensity 속성을 이용해 RenderSeries 로 불러진 새로 렌더링 된 흔적의 강도 효과 정도를 제어하라. 보통 값은 1…100 사이다. 얼마나 빠르게 색 범위가 흔적으로 채워지게 설정 되어있는지에 따랐다.

HistoryIntensityFactor 를 사용해 이전 흔적의 부패 속도를 수정하라. 보통 값 범위는 0.5 – 0.99 사이다.

HistoryIntensityFactor 자체를 설정하는 것은 다음 **RenderSeries** 의 콜 전에 레이어를 업데이트 하지 않는다.

5.32.5 레이어 안으로 데이터 렌더링

PointLineSeries, **FreeformPointLineSeries**, **SampleDataSeries, HighLowSeries 또는 AreaSeries** 를 **RenderSeries** 메소드로 레이어 안으로 렌더링 하라.

layer.RenderSeries(PointLineSeriesBase series): 레이어 안으로 한 시리즈 렌더링

layer.RenderSeries(List<PointLineSeriesBase> seriesList): 주어진 모든 시리즈를 레이어에 렌더링. **layer.RenderSeries(PointLineSeriesBase series)** 보다 성능이 낫진 않다.

데이터가 레이어에 업데이트 되었을 때 새로운 흔적에 **NewTraceIntensity** 가 사용된다. 옛날 흔적 데이터는 동시에 **HistoryIntensityFactor** 로 부패된다. **layer.RenderSeries(List<PointLineSeriesBase> seriesList)** 는 매 시리즈 객체 후 옛날 흔적을 부패 시킨다.

5.32.6 레이어 제거

레이어를 제거하고 차트와 렌더링을 방지하기 위해 **layer.Dispose()** 를 불러 사용하라.

5.32.7 레이어 내 앤티 앨리어싱

차트 렌더링 단계에 데이터를 앤티 앨리어싱 하기 위해 **layer.AntiAliasing** 를 **True** 로 설정하라. 이는 하드웨어가 지원하지 않더라도 앤티 앨리어싱을 활성화 한다.

5.32.8 레이어 목록 불러오기

ViewXY.GetPersistentSeriesRenderingLayers() 은 **PersistentSeriesRenderingLayers** 를 포함한 생성된 레이어의 목록을 불러온다.

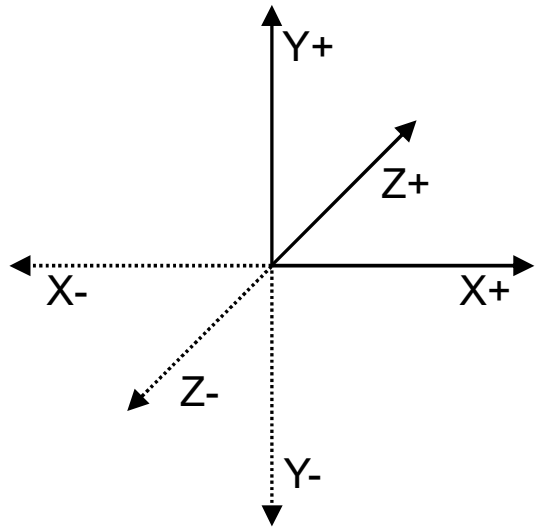
6. View3D

View3D 3 차원적인 데이터 시각화를 허용한다. 3D 모델은 확대, 회전 및 여러 방식대로 밝힐 수 있다. **합쳐진 시각화를 만들기 위해 같은 3D 뷰에 다른 시리즈 유형을 놓을 수 있다.**

View3D	**3D chart view**
Annotations	**(Collection)**
AutoSizeMargins	False
BarSeries3D	**(Collection)**
BarViewOptions	
Border	**Border**
Camera	
ClipContents	False
Dimensions	
FrameBox	**FrameBox - col -1**
LegendBox	**LegendBox3D**
Lights	**(Collection)**
Margins	**0, 0, 0, 0**
MeshModels	**(Collection)**
OrientationArrows	
PointLineSeries3D	**(Collection)**
Polygons	**(Collection)**
Rectangles	**(Collection)**
SurfaceGridSeries3D	**(Collection)**
SurfaceMeshSeries3D	**(Collection)**
VolumeModels	**(Collection)**
WallOnBack	**WallXY**
WallOnBottom	**WallXZ**
WallOnFront	**WallXY**
WallOnLeft	**WallYZ**
WallOnRight	**WallYZ**
WallOnTop	**WallXZ**
WaterfallSeries3D	**(Collection)**
XAxisPrimary3D	**HighlightingItemBase**
XAxisSecondary3D	**HighlightingItemBase**
YAxisPrimary3D	**HighlightingItemBase**
YAxisSecondary3D	**HighlightingItemBase**
ZAxisPrimary3D	**HighlightingItemBase**
ZAxisSecondary3D	**HighlightingItemBase**
ZoomPanOptions	

보기 7- 1. View3D 객체 메인 트리.

6.1 3D 모델 및 차원



보기 7-2. 3D 모델 양 및 음 방향

3D 세계 중심에 3D 모델이 설계된다. 차원 크기가 3D 공간 모델 상자의 크기를 정의한다. 벽 및 축 크기가 이 차원 상자와 함께 정의 된다. **Dimensions** 속성을 이용해 각 차원 크기를 설정하라.

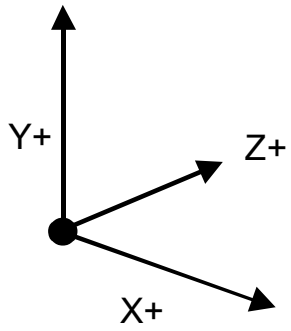
카메라 회전이 정의 되지 않았을 때 양 X 방향은 오른쪽이고 양 Y 차원은 위, 양 Z 방향은 화면 안쪽이다.

6.1.1 세계 좌표

어느 3D 객체들은 "세게 좌표"를 축 값 대신 사용한다. 예를 들어 불이 축 범위와 개별적으로 하기 위해 이런 식으로 위치 되어 있다. 세계 좌표는 "3D 모델 공간 좌표"로도 불릴 수 있다.

원점 [0,0,0] 은 모델의 중심에 있다. 실제 3D 모델 공간은 [-Dimensions.X/2 to Dimensions.X/2], [-Dimensions.Y/2 to Dimensions.Y/2] 및 [-Dimensions.Z/2 to Dimensions.Z/2] 이 있다.

라이트닝차트는 값을 시리즈 값, 축 값, 세계 좌표 및 화면 좌표 사이 변경할 수 있는 메소드를 제공한다. 데모 어플리케이션 예시 및 사용 설명서를 보아라.



보기 7-3. 3D 뷰 설정 예시. 차원이 X=100, Y=40, Z=80 로 설정 되었다. 보이는 벽은 좌, 하, 뒤. 투시 카메라가 사용 되었다.

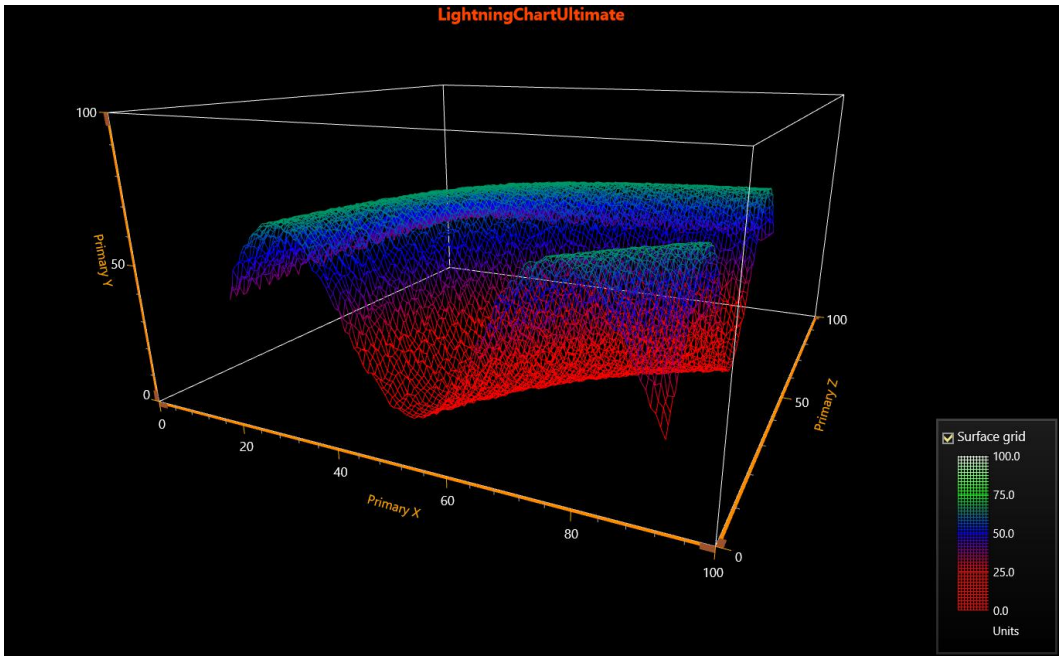
6.2 벽

벽 (**WallOnFront**, **WallOnBack**, **WallOnTop**, **WallOnBottom**, **WallOnLeft**, **WallOnRight**) 은 축 그리드 및 그리드 스트립 및 축에 베이스를 추기 위해 사용 됐다. 기본적으로 아래, 좌, 우, 뒤 및 앞 벽들이 보인다. **AutoHide** 속성이 참으로 설정 되었다. 뷰를 회전할 때에 뷰를 방해하는 벽들이 임시적으로 숨겨져 차트 내용을 막지 않게 된다. 벽을 강제로 보이게 설정하기 위해 **Visible = true** 및 **AutoHide = false** 로 설정하라.

XGridAxis, **YGridAxis**, **ZGridAxis**, **GridStripColorX**, **GridStripColorY**, **GridStripColorZ** 및 **GridStrips** 속성들을 사용해 어느 축에서 그리드가 적용 되는지를 선택하고 그리드 스트립의 색을 변경하라. 벽 정위 대로 사용 가능한 속성들이 다르다.

6.3 FrameBox

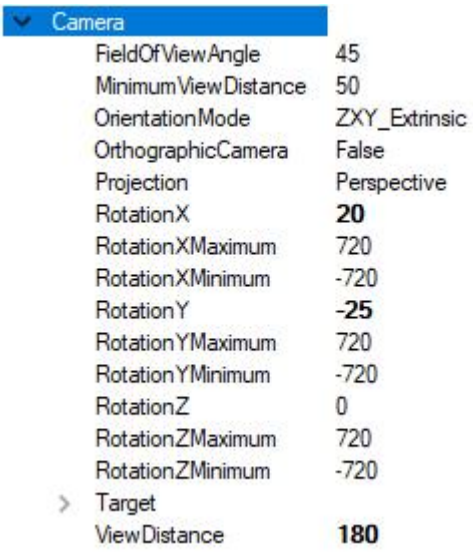
단순화 된 3D 상자 표현을 벽 대신 사용할 수 있다. 모든 벽에 Visible = false 설정하고



FrameBox.Style = AllEdges 설정하라. 색 또는 프레임을 **FrameBox.LineColor** 로 설정하라.

보기 7-4. FrameBox 가 보이고 벽이 숨겨졌다.

6.4 카메라



보기 7- 5. 카메라 속성.

카메라 유형, 위치, 거리 및 목표가 함께 3D 뷰포인트를 결정한다. **RotationX**, **RotationY, RotationZ** 및 **ViewDistance** 을 이용해 3D 모델 공간에 카메라 위치를 설정 하라. **Target** 속성을 설정해 카메라 방향을 원하는 대로 설정하라.

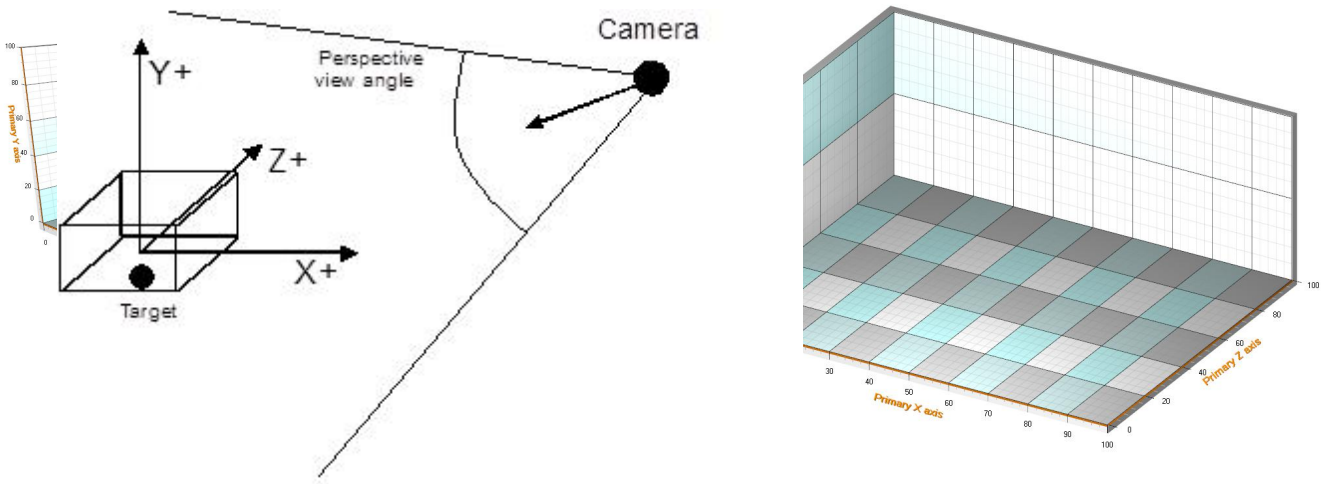
Projection 속성으로 투사 유형을 선택하라.

- **Perspective,** 현실적인 투사.
- **Orthographic**, 과학 및 공학 응용 프로그램에 사용되는 투사 유형. **OrthographicLegacy** 보다 이 선택을 추천한다.
- **OrthographicLegacy**, (라이트닝차트 v.8.3 이전에 OrthoGraphicCamera = True 와 동일함). 이것은 Orthographic 에 비해 확대 후 렌더링이 느리다. 이는 3D 객체의 크기를 (축 값이 아닌) 3D 세계 좌표로 정의 되었어도 유지한다. 또한 벽의 두께가 확대 및 축소 중 같다. 확대하는 것은 차원을 변경하지만 **ViewDistance** 에는 효과 없다.

RotationX, RotationY 및 **RotationY** 은 경계를 설정하여 제한 할 수 있다. **RotationXMinimum**, **RotationXMaximum**, **RotationYMinimum**, **RotationYMaximum**, **RotationZMinimum** 및

RotationZMaximum 속성들을 설정하여 가능하다.

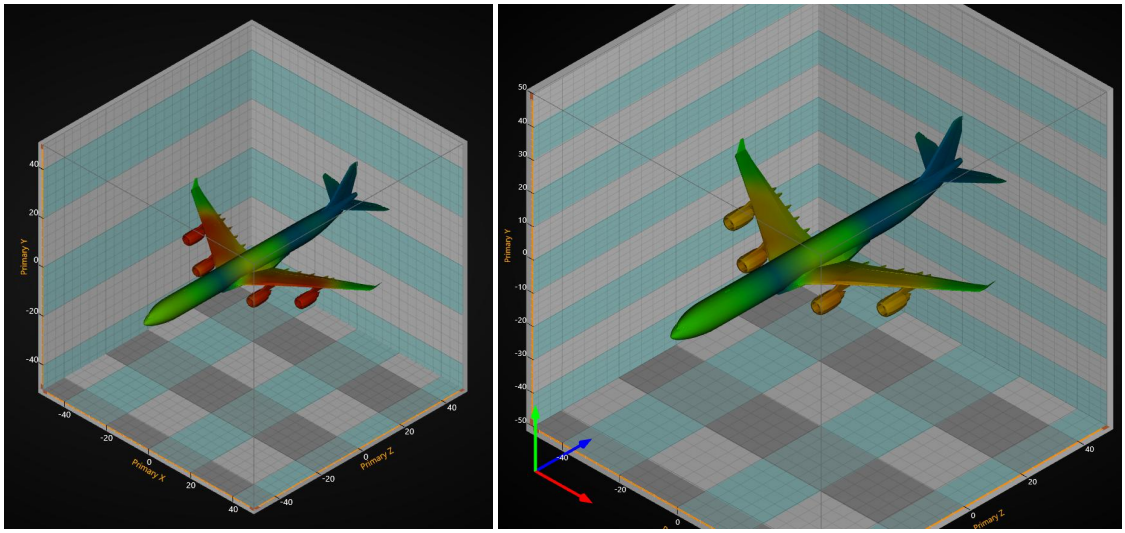
보기 7-6. 3D 공간에서의 투시 카메라 표현.



보기 7-7. 3D 공간에 투시 및 직교 카메라 뷰.

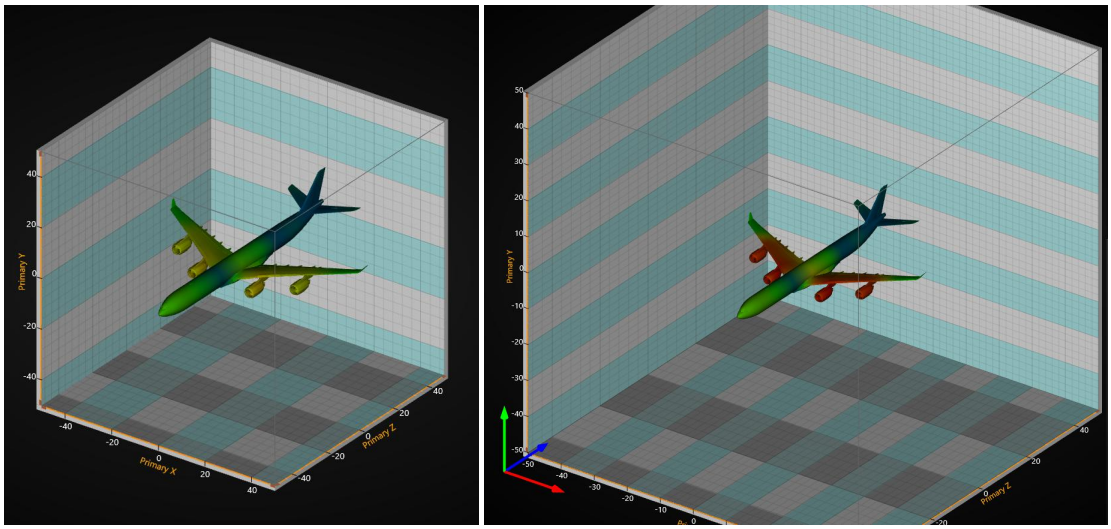
Orthographic 및 OrthographicLegacy 의 줌 뷰는 다음과 같이 다르다:

Orthographic



보기 7-8. 직교 투사 유형. 왼족에는 줌 없음. 오른쪽에는 줌 인. 비행기 (MeshModel3D) 객체가 화면에 다른 객체와 같이 커진다.

OrthographicLegacy



보기 7-9. OrthographicLegacy. 왼쪽에는 줌 없음. 오른쪽에는 줌 인. 비행기 (MeshModel3D, 6.14 를 보아라) 객체는 변하지 않지만 3D 차원이 달라진다.

사전 정의 카메라

데모 예시: 카메라 및 빛

View3D.Camera 의 **SetPredefinedCamera** 메소드를 이용해 사전 정의 된 카메라 중 하나를 사용하라.

```
// Setting predefined camera orientation
chart.View3D.Camera.SetPredefinedCamera(PredefinedCamera.BackOrthographic);
```

6.1.1 카메라 정위 모드

라이트닝차트 v8.4 는 더욱 나아진 카메라 정위 정의의 새로운 카메라 정위 모드를 추가했다. 새로운 모드의 이름은 **ZXY_Extrinsic (**이름이 계산되는 차원의 순서를 정의한다) 이다. 이 모드가 기본 정위 모드로 설정 되어있다. 이는 회전 기반 문제들을 고쳤다. 특히 차트의 극 근처 회전으로 인해 나타난 문제들을 많이 고쳤다 (예를 들어 차트 위에 위치한 카메라). 옛날 정위 모드, **XYZ_Mixed** 는 아직 사용 가능하지만 미래에 없어질 계획이다. **View3D.Camera.OrientationMode** 를 통해 정위 옵션에 접근 가능하다.

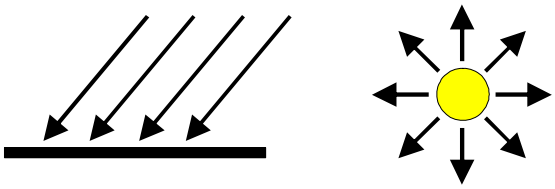
이 변화와 같이 회전 또한 변경 되었다. 새로운 카메라 정위 모드에는 축 방향 (세계 유닛 벡터)가 수평 마우스 회전 축과 같이 사용된다. 이것이 카메라가 회전 되는 축이다. 축 결정은 **RotationX**, **RotationY** **또는**

RotationZ 속성들이 변경 될 때 자동으로 되어진다. 카메라의 윗 방향과 가장 가까운 축이 회전 축으로 선택되어 모든 상황에 회전이 자연스럽게 느껴진다.

새로운 정위 및 회전 모델은 전에 불가능했던 3D 뷰를 가능하게 만든다.

6.5 빛

빛은 3D 모델 공간 아무데에나 자유롭게 위치 가능하다. **Lights** 컬렉션 속성에 여러 빛을 추가 가능하다. 빛 종류에 두가지가 있다: **Directional 및 PointOfLight.**



보기 7- 10. Directional 빛 및 포인트 오브 라이트.

주의! 어느 시리즈 유형은 **SuppressLighting** 속성을 통해 표면에서 빛을 완전히 막는 것을 허용한다. 시리즈가 제대로 밝혀져야 한다면 활설화가 되었나 확인하라. 표면 시리즈는 **LightedSurface** 속성을 갖고 있어 제대로 밝혀진 표면을 선택한다.

주의! 3D 모델 상자 안에 모든 빛을 넣으면 벽 면을 매우 어둡게 만들 수 있다. 축 티크가 잘 보이지 않을 수 있다. 이런 경우에는 축 티크 색을 수정하라.

방향적 빛

Directional 빛에는 빛 선이 평행선이다. 거리가 증가 할 수록 빛의 강도가 감소되지 않는다. 빛 플럭스가 **Location** 및 **Target** 속성에서 방향을 얻는다. **LocationFromCamera** 속성은 카메라 위치를 빛의 원천지로 사용하게 해준다.

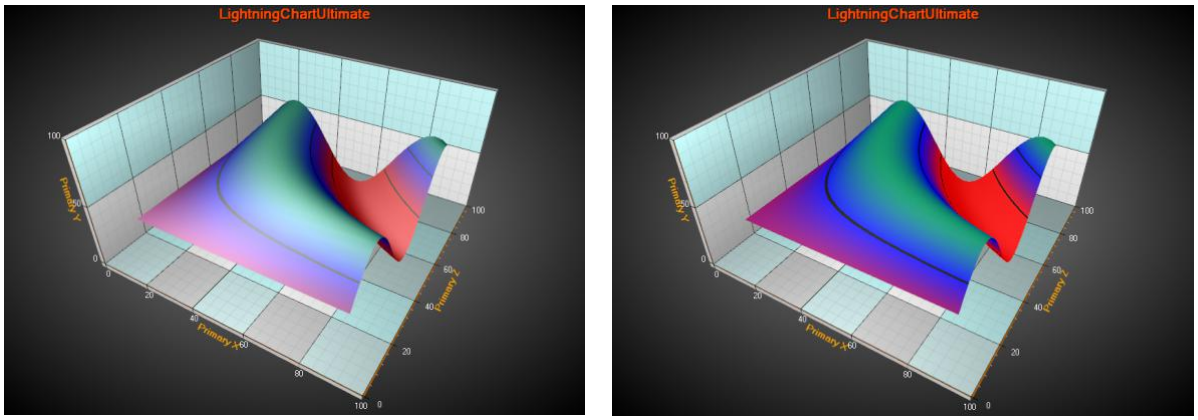
빛의 원천지

PointOfLight 에서는 거리가 멀어질 수로 빛의 강도가 감소 된다. **AttenuationConstant**, **AttenuationLinear** 및 **AttenuationQuadratic** 속성들을 사용해 거리 따라 강도의 감소를 제어하라. **Target** 은 이 빛의 종류와는 상관이 없다. 이는 빛이 모든 방향으로 똑같이 분배 되기 때문이다.

빛과 기재

모든 3D 객체들은 **Material** 속성이 있다. 기재는 빛에 어떻게 반응하는지 말해준다. 기재의 **DiffuseColor** 가 빛의 **DiffuseColor** 에 반응을 한다. 기재의 **SpecularColor** 가 빛의 **SpecularColor** 와 반응을 한다. 색 퍼짐은 매트 베이스 색으로 이해해도 되고 색 반사광은 비춰진 표면에 반사되는 색이다. 높은 **SpecularPower** 을 사용하면 객체에게 메탈과 비슷한 외관을 준다.

표면 시리즈는 **ColorSaturation** 속성을 갖고 있다. 사용 가능한 범위는 0 에서 100%이다. 높은 값은 표면 채움 색을 부스트하며 쉐이딩 효과를 줄인다.

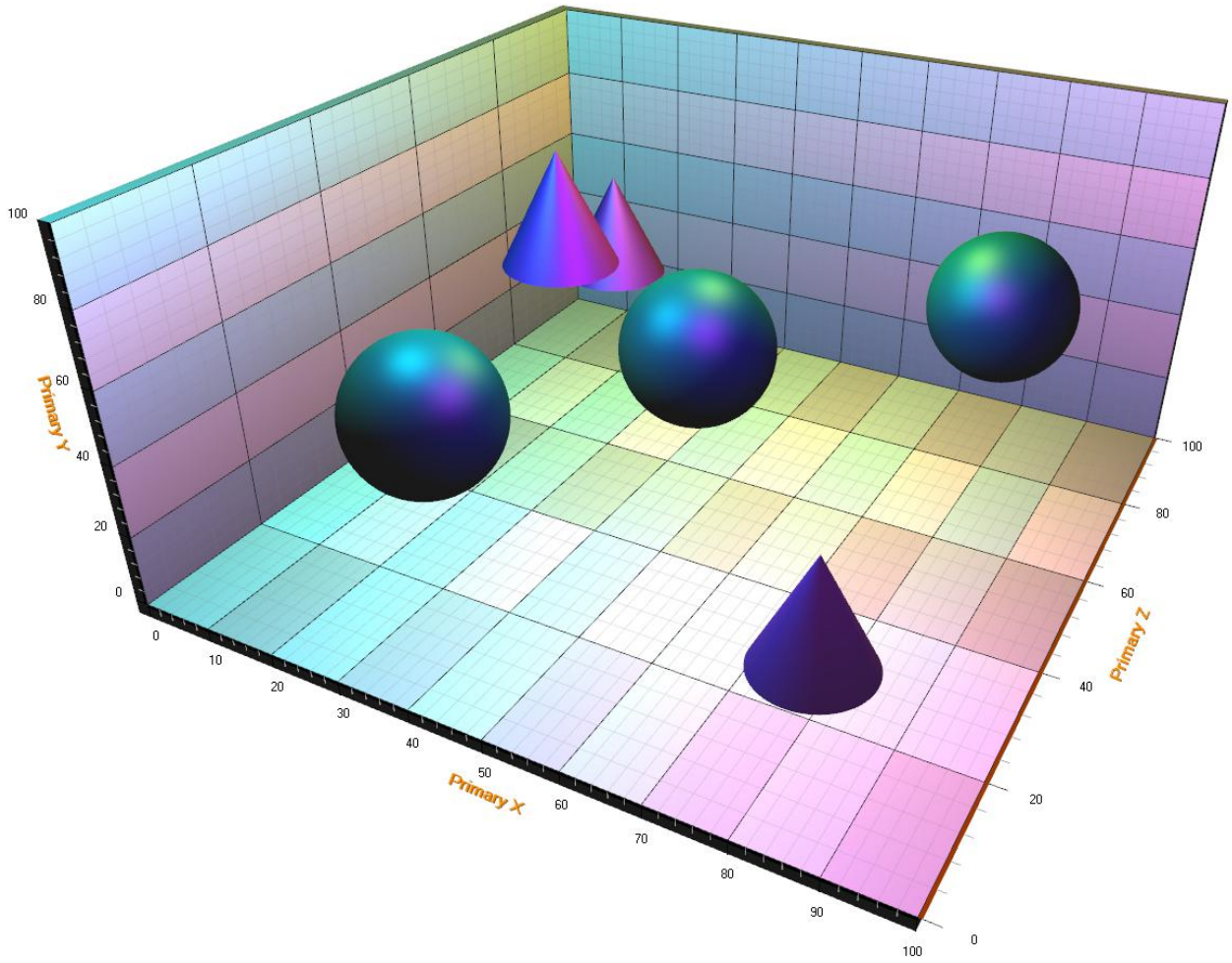


보기 7-11. 왼쪽의 표면 시리즈는 ColorSaturation = 50%이다. 오른쪽에는 ColorSaturation = 85%이다.

사전 정의된 조명 계획

데모 예시: 카메라 및 빛

View3D 의 **SetPredefinedLightingScheme** 메소드를 이용해 빌트인 사전 정의된 조명 계획을 사용하라.



보기 7-12. 사전 정의된 'DiscoCMY' 사용. 이 계획은 천장 근처 세가지 다른 색의 PointOfLights 을 사용했다. 원과 원뿔은 PointLineSeries3D 로 만들어 졌다.

6.6 축

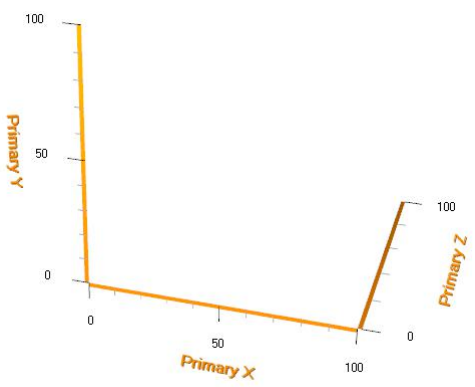
각 차원에는 두 축이 있다: 기본 및 보조. 다른 말로는 View3D 는 다음과 같은 속성 사용 가능하다: **XAxisPrimary3D**, **XAxisSecondary3D**, **YAxisPrimary3D**, **YAxisSecondary3D**, **ZAxisPrimary3D** 및 **ZAxisSecondary3D.**

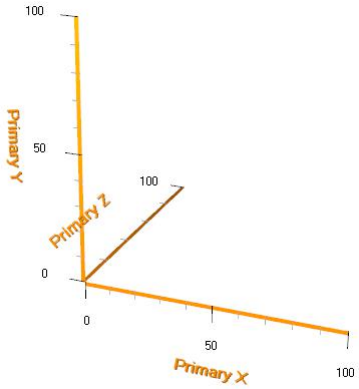
기본적으로 3D 축이 ViewXY 의 축과 비슷하게 행동한다. 많은 속성과 메소드가 비슷하다.

6.1.1 위치

3D 모델 상자 모서리에 축을 위치할 수 있다. 축의 **Location** 속성을 이용해 위치를 수정하라.

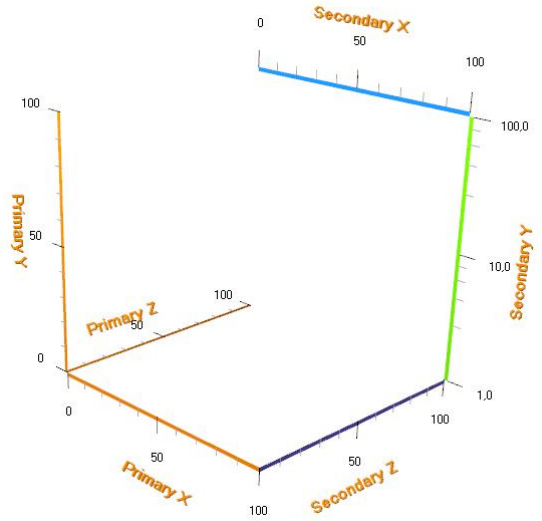
- X 축에는 **Location** 옵션이: **BottomFront**, **BottomBack**, **TopFront** 및 **TopBack** 이 있다.
- Y 축에는 **Location** 옵션이: **FrontLeft**, **FrontRight**, **BackLeft** 및 **BackRight** 이 있다.
- Z 축에는 **Location** 옵션이: **BottomLeft**, **BottomRight**, **TopLeft** 및 **TopRight** 이 있다.





보기 7- 13. ZAxisPrimary 위치는 BottomLeft 에 설정.

은 BottomRight

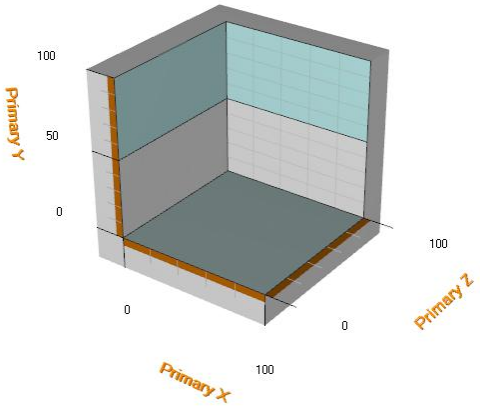


보기 7- 15. 보조 축 보이게 설정 및 위치와 색 임의로 설정. 보조 Y 축 ScaleType Logarithmic 로 설정.

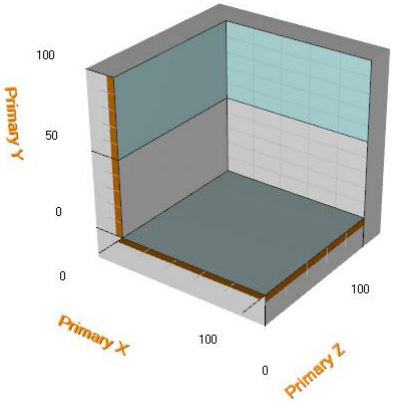
6.1.1 정위

각 축은 두 평면에 정위 가능하다. 이것은 두 축 티크 및 값 라벨의 위치 및 정위에 적용된다.

- X 축: XY 및 XZ 평면
- Y 축: XY 및 YZ 평면
- Z 축: XZ 및 YZ 평면



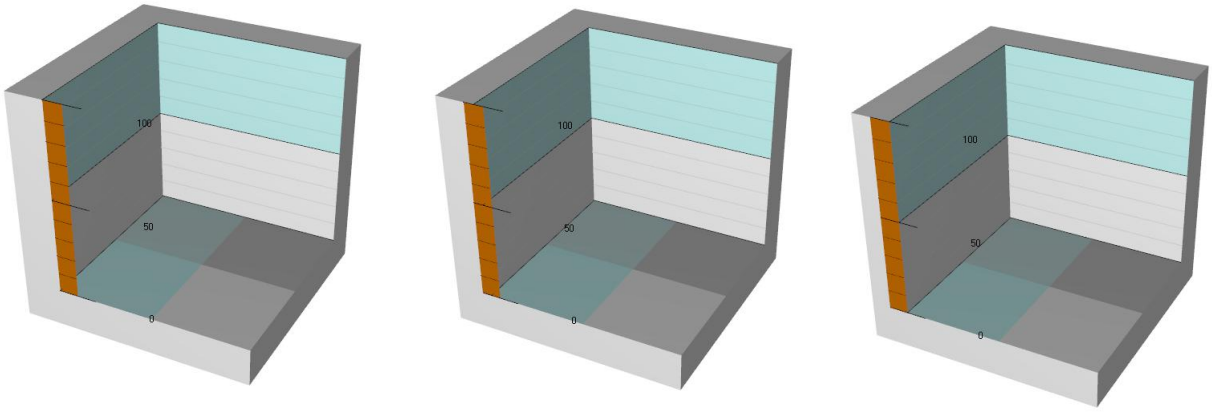
보기 7- 16. X 축 정위는 XY 및 Z 축 정위는 XZ 평면.



보기 7- 17. Y 축 정위는 그대로 이지만, X 축 정위는 ZY 평면으로 변경.

6.1.1 CornerAlignment

3D 모델 상자 모서리에 축 정렬은 **CornerAlignment** 속성으로 변경 가능하다. **MajorDivTickStyle, MinorDivTickStyle 및 Alignment** 속성들을 사용해 텍스트 정렬을 제어하라.



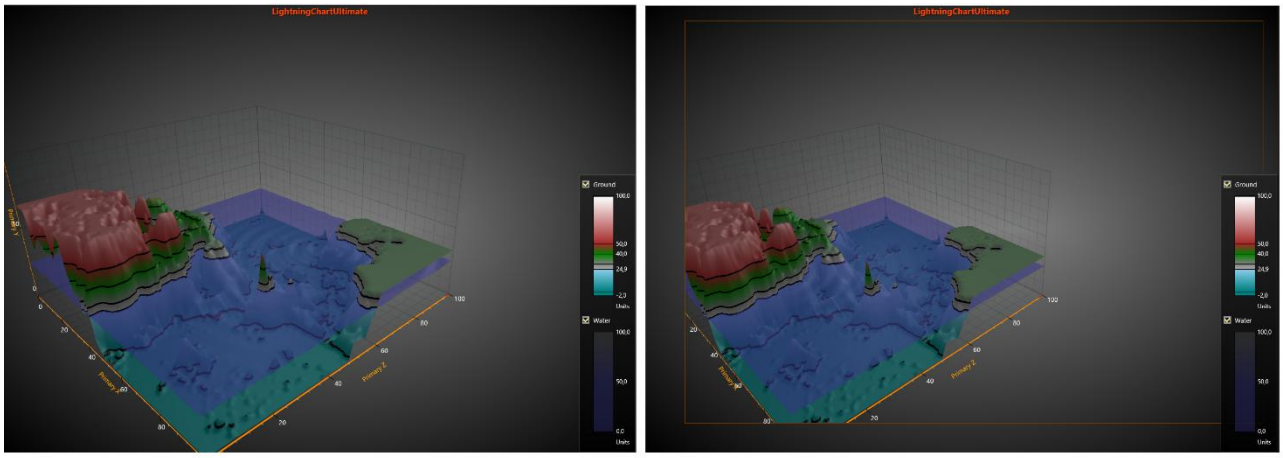
보기 7-18. 이 예시에는 Y 축만 보인다. 보기 1: 축 CornerAlignment 가 Inside 로 설정. 정렬 속성 MajorDivTickStyle 및 MinorDivTickStyle 은 Near 로 설정 되었다. 보기 2: CornerAlignment 가 AtCorner 로 설정. 보기 3: CornerAlignment 가 Outside 로 설정.

6.7 마진

라이트닝차트 v8.4 이후에는 View3D 가 마진을 지원한다. ViewXY 와 비슷하게 **AutoAdjustMargins** 가 **true** 으로 설정 되었을 때 그래프 크기가 모든 축과 차트 제목을 위한 공간을 확보하기 위해 수정 된다. **비활성화** 되어 있으면 **View3D.Margins** 속성이 활성화 되어 수동 마진 설정을 허용한다. **AutoAdjustMargins** 의 기본 설정은 **거짓으로** 되어 있다.

View3D.MarginsChanged 이벤트는 마진의 크기가 재 설정 등 마진에 변경이 일어났을 때 트리거 되게끔 설정 가능하다.

마진 밖의 뷰 내용은 자동으로 잘린다. 차트 제목, 주석 및 레전드 상자와 화면 좌표로 정의 된 그들의 위치 외에는 모든 내용이 잘린다. 이 목록은 마진에도 자유롭게 위치 선정 가능하다. 1 픽셀 넓이의 태두리를 가진 직사각형 **Border** 는 마진이 어디에 위치 되어있는지 보여주기 위해 그릴 수 있다. 기본적으로 View3D 에는 태두리가 보이지 않게 설정 되어있다. 직사각형의 색은 **Border.Color** 를 통해 변경 가능하다.



보기 6-19. 왼쪽 그래프는 마진이 없다 (모든 마진 0 으로 설정). 오른쪽에는 마진이 설정되고 마진 밖의 내용물은 잘렸다. Border.Visible 은 True 로 설정 되어 뷰의 마진이 어디에 있는지 보여준다.

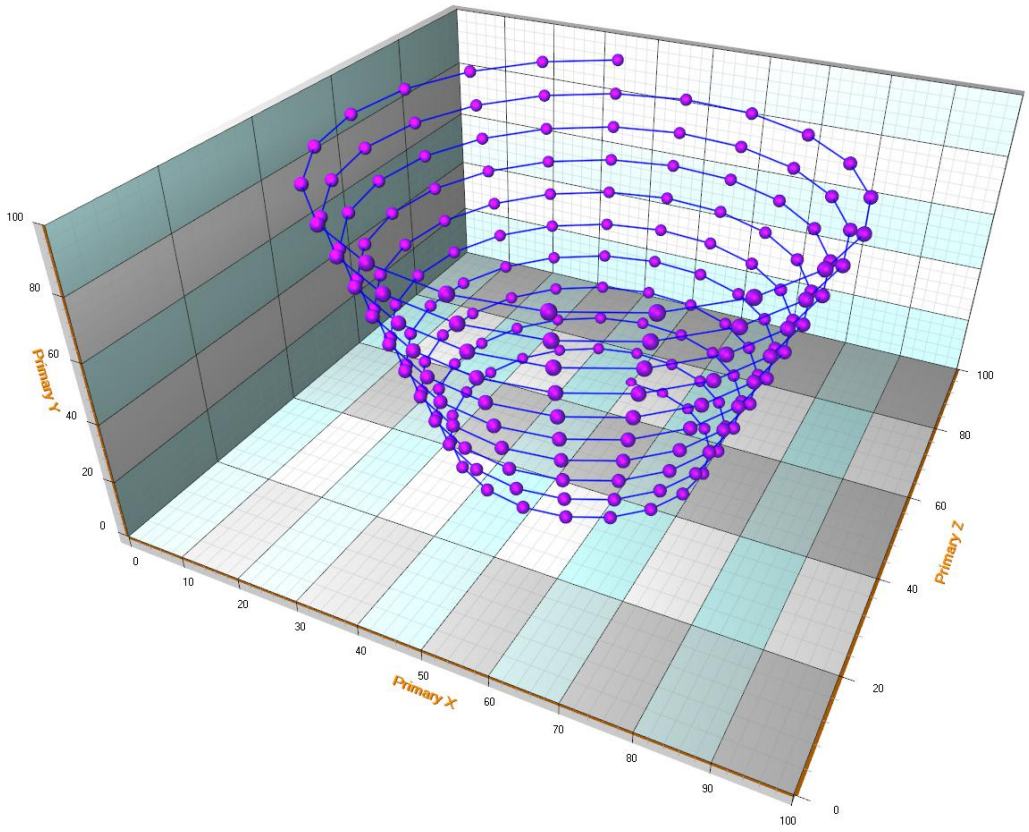
6.8 3D 시리즈, 기본

View3D 의 시리즈는 여러 다른 방식 및 형식으로 데이터 시각화가 가능하다. 모든 시리즈는 축 값 범위에 묶여 있다. 각 차원의 시리즈는 기본 또는 보조 축에 선택 되어 묶일 수 있다. **XAxisBinding**, **YAxisBinding** 및 **ZAxisBinding** 속성들을 선택해 이 기능을 제어하라.

6.9 PointLineSeries3D

데모 예시: 스캐터 포인트; 포인트 라인; 포인트 트래킹; 포인트 클라우드; 평행 좌표 표; 여러 색 3D 포인트 라인

PointLineSeries3D 는 3D 공간에 포인트 및 선의 표현을 허용한다. 포인트에는 여러 기본 3D 모양들을 사용 가능하다. **LineVisible 가** 참으로 설정 되어 있으면 선으로 포인트 들이 연결 된다.



보기 6- 20. PointLineSeries3D 의 예시. PointStyle 의 모양은 원으로 설정 됨.

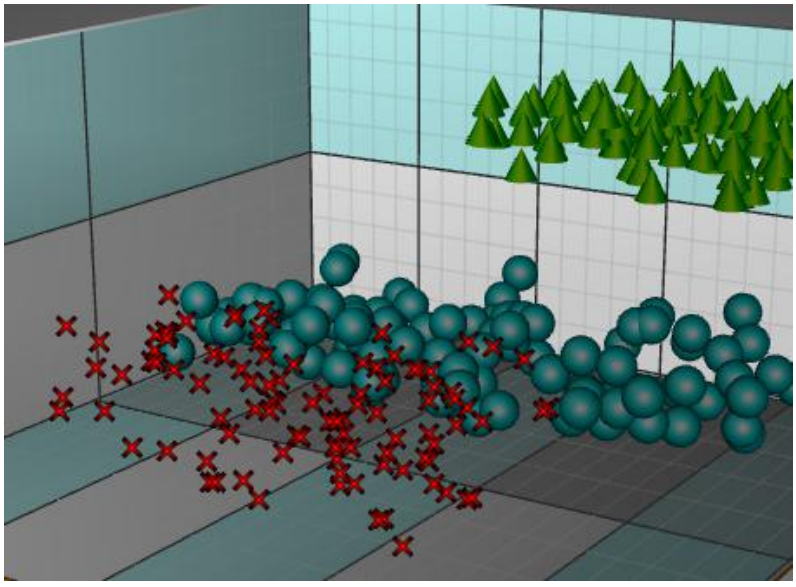
6.1.1 포인트 스타일

포인트는 2D 모양이 아닌 실제 3D 포인트로 표현 가능하다.

Points can be shown as real points, or as 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	

보기 6-21. PointStyle 속성 트리. ShapeType 을 이용해 2D 및 3D 모양 사이 변경 가능하다.



보기 6-22. 빨간색 x 모양은 ShapeType = Shape2D 를 갖고있다. 청록과 초록색 객체는 ShapeType = Shape3D 를 갖고 있다.

주의! 2D 모양들은 모든 3D 객체들 위로 렌더링 되고 다른 객체의 가시성에 의해 숨기는 기능이 지원 되지 않는다.

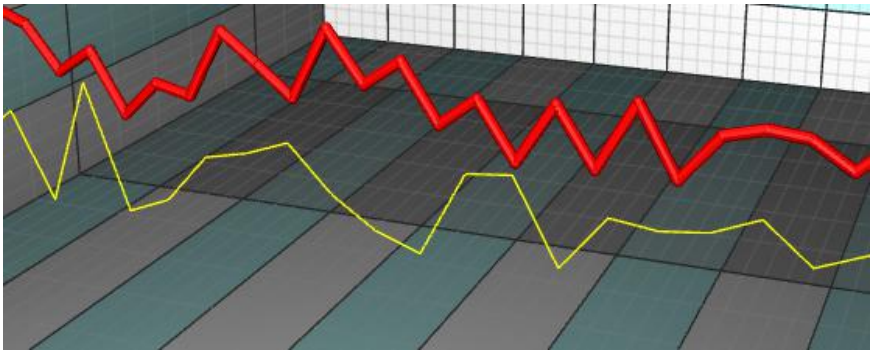
라인 스타일

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

보기 6-23. LineStyle 속성.

선들이 쉐이딩 된 3D 선으로 또는 1 픽셀 넓이의 얇은 선으로 렌더링 가능하다.

시리즈에 많은 데이터가 있을 때에는 **LineOptimization = Hairline** 로 설정하는 것을 추천한다. 이는 성능을 최적화 하기 위해서다.



보기 6-24. 노란 선: LineStyle.LineOptimization = Hairline. 빨간선: LineStyle.LineOptimization = NormalShaded.

6.1.1 포인트 추가

PointLineSeries3D 는 세가지의 다른 포인트 형식을 지원한다:

- Points 속성 (SeriesPoint3D 배열)
- PointsCompact 속성 (SeriesPointCompact3D 배열)
- PointsCompactColored 속성 (SeriesPointCompactColored3D 배열)

PointsCompact 및 **PointsCompactColored** 구형물은 매우 메모리 효율적이다. 간단한 포인트 스타일로 1 억 개의 데이터 포인트 까지 시각화를 허용한다.

PointsType 속성을 통해 포인트 형식을 설정하라.

6.9.3.1 포인트

Points 속성을 이용함으로 포인트 고급 색 옵션이 지원 된다. SeriesPoint3D 구형은 다음과 같은 필드로 구성 되었다:

더블 X:	X 축 값
더블 Y:	Y 축 값
더블 Z:	Z 축 값
Color 색:	각 데이터 포인트 색. **IndividualPointColors** 이 활성화 되었거나 **MultiColorLine** 가 활성화 되었을 때만 적용 된다.
플로트 sizeFactor:	크기 팩터가 **PointStyle.Size** 에 정의 된 크기를 곱한다. **IndividualPointSizes** 가 활성화 되었을 때만 적용 된다.
객체 Tag:	자유롭게 할당 가능한 보고 객체. 예를 들어 세부 정보를 첨부하기 위한

시리즈 포인트는 코드로 추가해야 한다. **AddPoints(…)** 메소드를 이용해 존재하는 포인트 끝에 포인트를 추가하라.

```
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
```

시리즈 데이터 전체를 설정 도중 옛날 포인트를 덮어 쓰기 위해 새로운 포인트 배열을 직접 할당하라:

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

6.9.3.2 PointsCompact

PointsCompact 속성은 저 메모리 소비를 활성화 한다. 이는 많은 데이터 포인트가 있을 때 중요하다. SeriesPointCompact3D 구성은 다음과 같은 필드로 구성 되었다:

플로트 X: X 축 값

플로트 Y: Y 축 값

플로트 Z: Z 축 값

```
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
```

시리즈 데이터 전체를 설정 도중 옛날 포인트를 덮어 쓰기 위해 새로운 포인트 배열을 직접 할당하라:

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

6.9.3.3 PointsCompactColored

PointsCompactColored 속성은 저 메모리 소비를 활성화한다. 이는 많은 데이터 포인트가 있을 때 중요하다. 개인 색으로 포인트를 칠하는 것을 허용한다.

SeriesPointCompactColoured3D 구조는 다음과 같은 필드로 구성 되었다:

플로트 X: X 축 값

플로트 Y: Y 축 값

플로트 Z: Z 축 값

int Color: 포인트 색

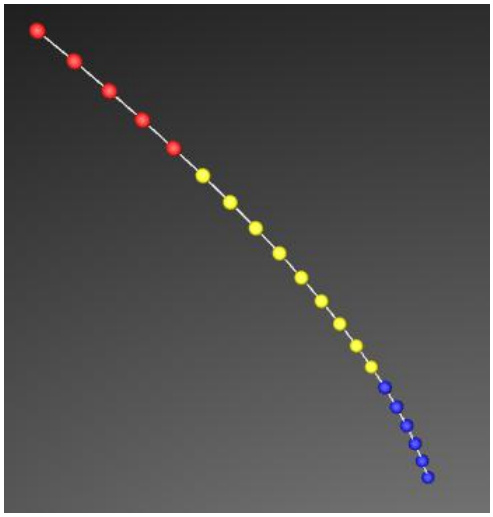
```
SeriesPointCompactColored3D[] pointsArray = new
SeriesPointCompactColored3D[3];
pointsArray    [0]    =    new    SeriesPointCompactColored3D(50,    50,    50,
Color.Blue.ToArgb());

pointsArray    [1]    =    new    SeriesPointCompactColored3D(30,    50,    20,
Color.Red.ToArgb());

pointsArray    [2]    =    new    SeriesPointCompactColored3D(80,    50,    80,
Color.Green.ToArgb());

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

시리즈 데이터 전체를 설정 도중 옛날 포인트를 덮어 쓰기 위해 새로운 포인트 배열을 직접 할당하라:

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

개별적으로 포인트 색칠

IndividualPointColors = True 로 설정함으로 포인트의 색 필드가 **Material.DiffuseColor** 대신 적용 된다.

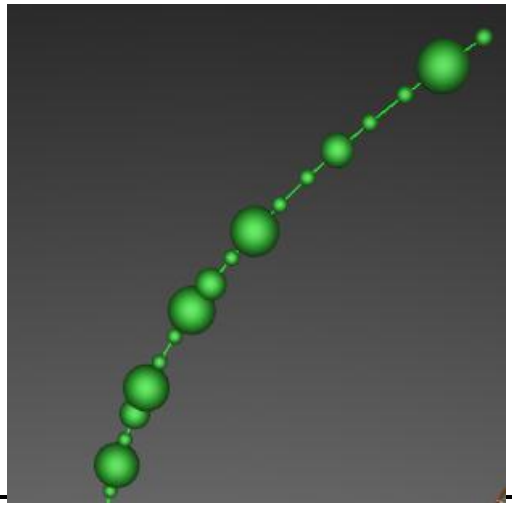


보기 6- 25. IndividualPointColors 사용.

주의! 개별적 포인트 색은 PointsType = PointsCompact 로 설정 되었을 때 지원 되지 않는다.

개별적으로 포인트 크기 설정하기

IndividualPointSizes = True 로 설정함으로 포인틔 **sizeFactor** 필드들이 적용된다. 배수는 **PointStyle.Size** 에 정의 된 크기를 곱한다.

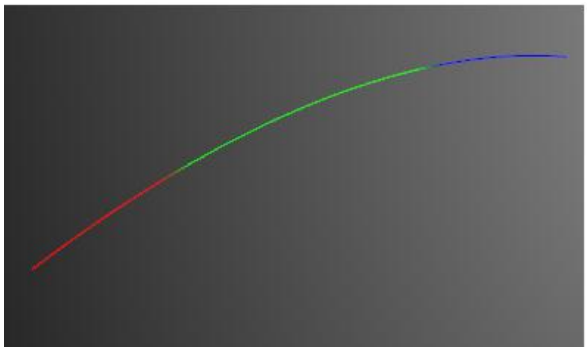


보기 6-26. IndividualPointSizes 사용.

주의! 개별적 포인트 크기는 PointsType = PointsCompact, 또는 PointsCompactColored 가 있을 때 지원 되지 않는다.

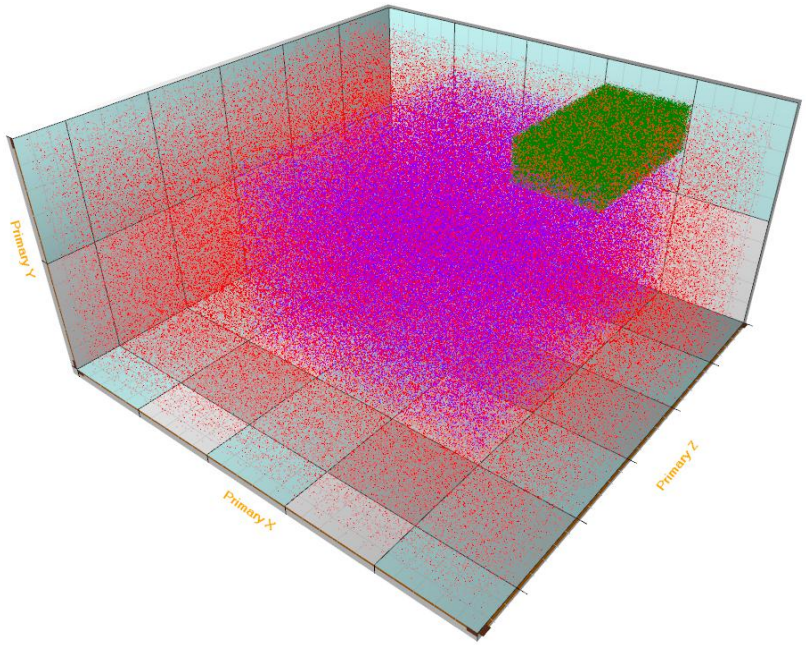
멀티 색 라인

주어진 데이터 포인트 색으로 선을 칠하기 위해서는 **MultiColorLine = True** 으로 설정하라. 차트가 양 포인트 사이 색 그라디언트를 보간 한다.

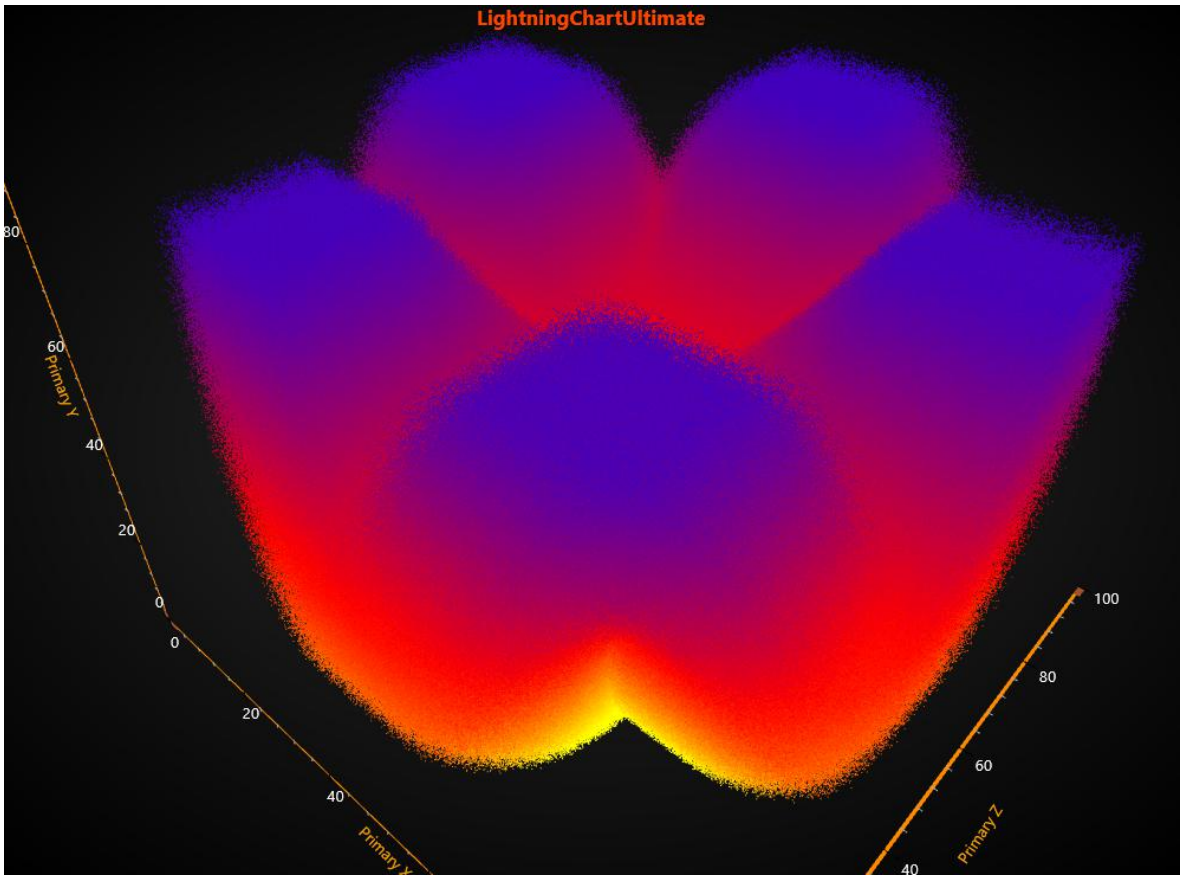


보기 6-27. MultiColorLine 활성화.

주의! MultiColorLine 은 PointsType = PointsCompact 가 있을 때 지원 되지 않는다.

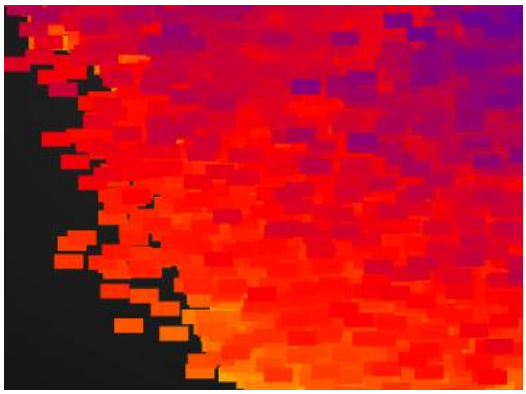


보기 6- 28. 수 백만개의 스캐터 포인트. LineVisible = False, PointsVisible = True, PointsOptimization = Pixels.



보기 6- 29. IndividualPointsColoring = True, using PointsCompactColored, LineVisible = False, PointsVisible = True. 1.2 억개의 스캐터 포인트

수 백만개의 데이터 포인트는 직사각형으로 가장 효율적으로 시각화 할 수 있다. **PointsCompactColored** 또는 **PointsCompact** 을 사용할 때 포인트의 크기는 **PointStyle.Shape2D.Width** 및 **PointStyle.Shape2D.Height** 로 제어 가능하다.

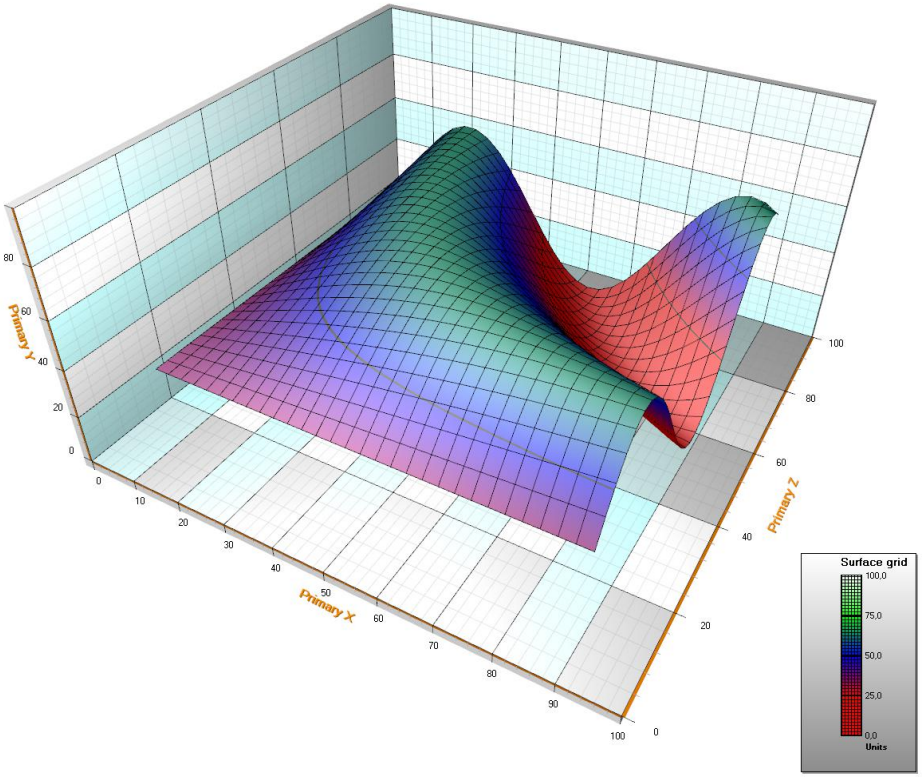


보기 6- 30. PointStyle.Shape2D.Width = 20 및 PointStyle.Shape2D.Height = 10.

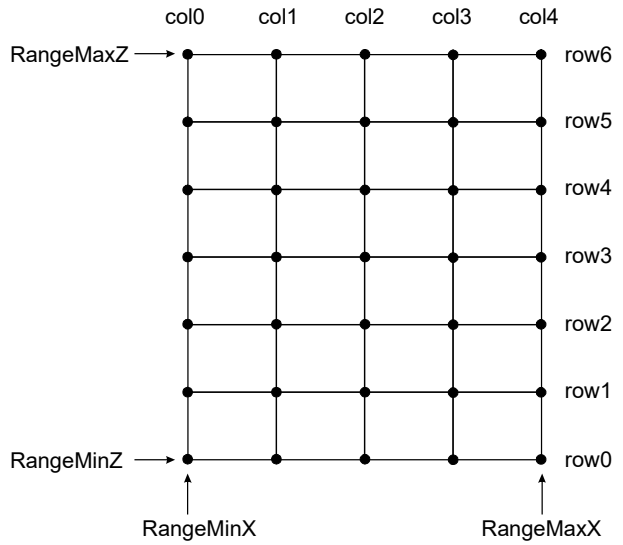
6.10 SurfaceGridSeries3D

데모 예시: 간단 3D 표면 그리드, 납작 투사; 표면 그리드, 값 색칠; 표면 그리드, 물과 육지

SurfaceGridSeries3D 는 3D 표면으로 데이터를 시각화할 수 있다. **SurfaceGridSeries3D** 에서는 노드가 x 와 z 차원에 일정한 간격으로 되어있다.



보기 6-31. 기본 스타일의 표면 그리드 시리즈. 높이 데이터가 사인 함수로 만들어졌다. 레전드 상자가 높이 색 간격을 보여준다



보기 6-32. 표면 그리드 노드. SizeX = 5, SizeZ = 7.

노드 거리는 다음과 같이 자동으로 계산 된다

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

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

표면 그리드 데이터 설정

- X 범위를 **RangeMinX** 및 **RangeMaxX** 속성들을 이용해 설정하라. 이는 할당된 X 축 기반으로 최소 및 최대 값 순서대로 나열한다.
- Z 범위를 **RangeMinZ** 및 **RangeMaxZ** 을 사용해 설정하라. 이는 할당된 Z 축 기반으로 최소 및 최대 값 순서대로 나열한다.
- **SizeX** 및 **SizeZ** 속성들을 설정해 그리드를 열 및 행 기반 크기를 주어라.
- 모든 노드에 Y 값을 설정하라:

  데이터 배열 인덱스를 사용한 방식

```
    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 to refresh when the new values are
ready
```

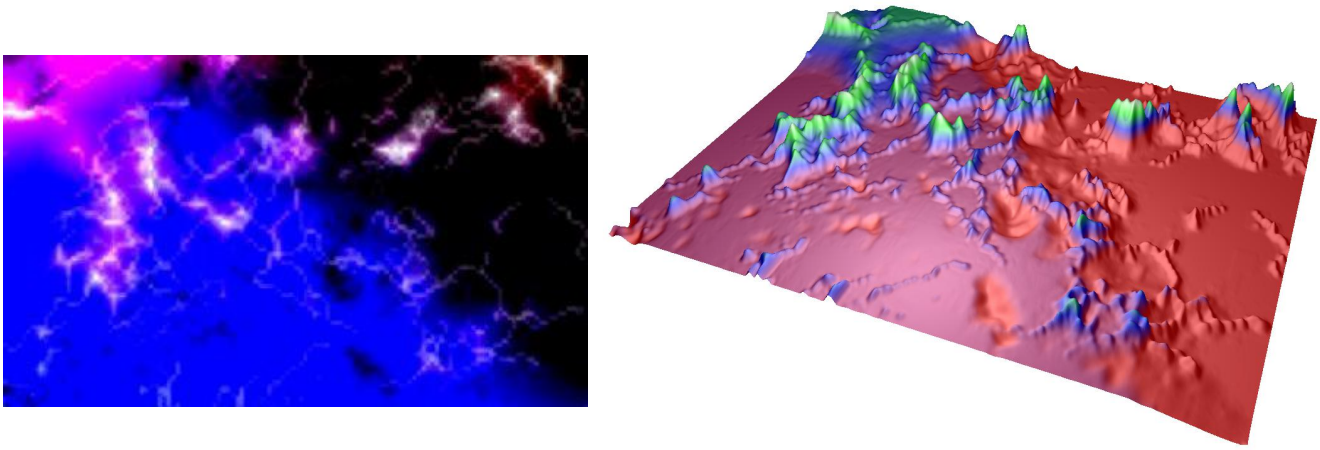
SetDataValue 를 사용한 다른 방식

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

비트맵 파일에서 표면 생성하기

데모 예시: 큰 표면

비트맵 이미지에서 **SetHeightDataFromBitmap** 메소드를 불러 표면을 생성할 수 있다. 표면은 비트맵의 크기를 갖게 된다 (앤 앨리어싱 및 재샘플링이 사용되지 않을 시). 각 비트맵 이미지의 빨강, 초록 및 파랑 값들이 합해진다. 합이 클수록 해당 노드의 높이 데이터 값이 높아진다. 검정 및 어두운 색들은 밝고 흰 색보다 낮은 값을 갖게 된다.

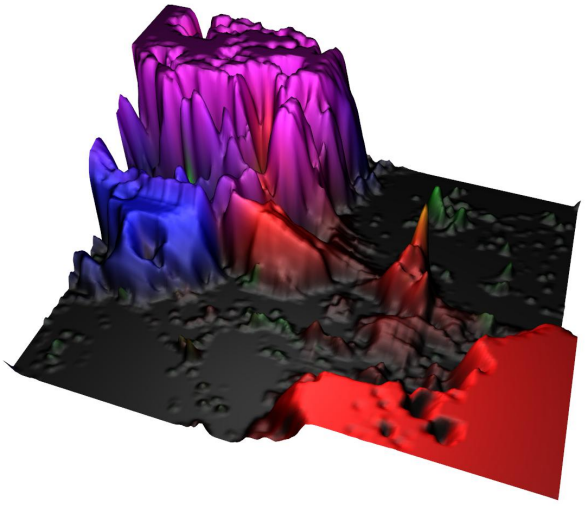


보기 6-33. 소스 비트맵 및 계산된 표면 높이 데이터. 어두운 값은 낮고 밝은 값은 표면이 높아진다.

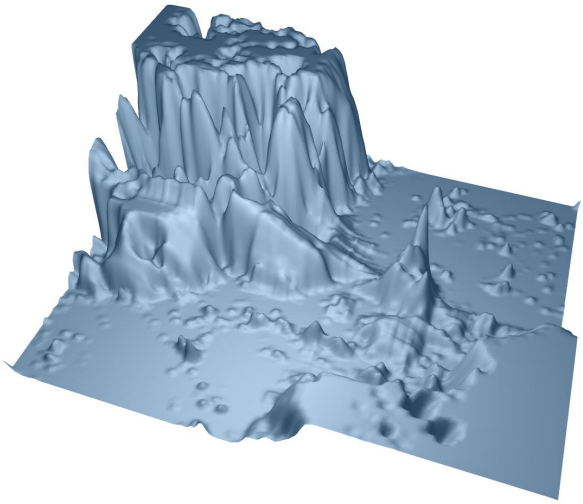
채우기 스타일

Fill 속성을 이용해 표면의 채우기 스타일을 선택하라. 다음과 같은 옵션을 선택할 수 있다:

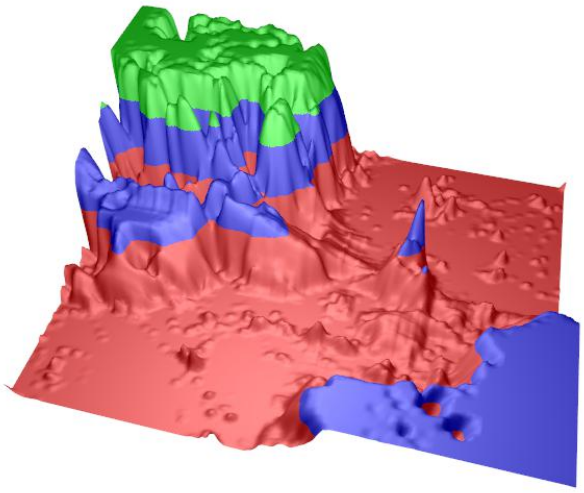
- **None**: 이 옵션을 선택하면 채우기가 적용 되지 않는다. 이 선택은 와이어프레임 메쉬와 유용하다.
- **FromSurfacePoints**: 데디터 속성 노드의 색이 사용된다.
- **Toned**: ToneColor 가 적용된다.
- **PalettedByY**: Y 값 팔레뜨대로 색칠, 7.10.4 장을 보아라.
- **PalettedByValue**: **SurfacePoint** 의 **Value** 값 필드의 팔레뜨 대로 색칠, 7.10.4 장을 보아라.
- **Bitmap**: 비트맵 이미지가 표면 전체를 덮게 늘어진다. 비트맵 이미지를 **BitmapFill** 속성에서 설정하라. **BitmapFill** 속성은 이미지를 수직 및 수평으로 대칭할 수 있는 하위 속성을 갖고 있다.

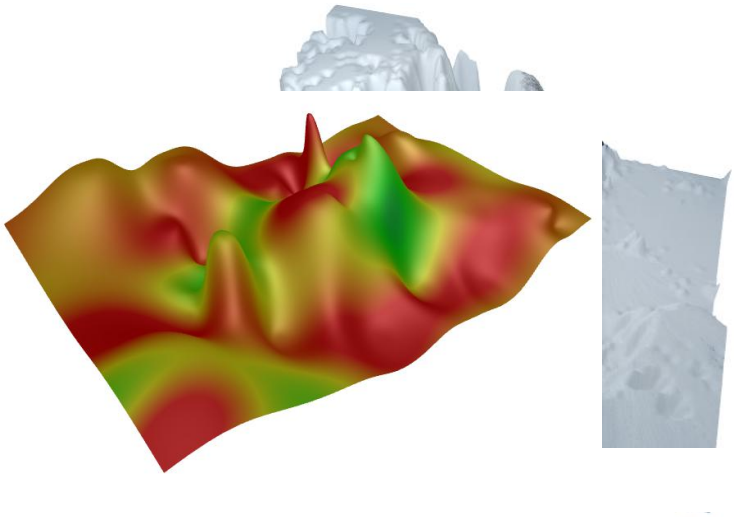


보기 6-34. FromSurfacePoints 채우기. 데이터 포인트 당 색칠.



보기 6-35. 톤 채우기.





보기 6-36. PalettedByY

윤곽 팔레뜨

ContourPalette 속성은 높이 색칠을 위한 색 단계 정의 설정을 허용한다. **ContourPalette** 다음과 같은 용도에 사용 가능하다:

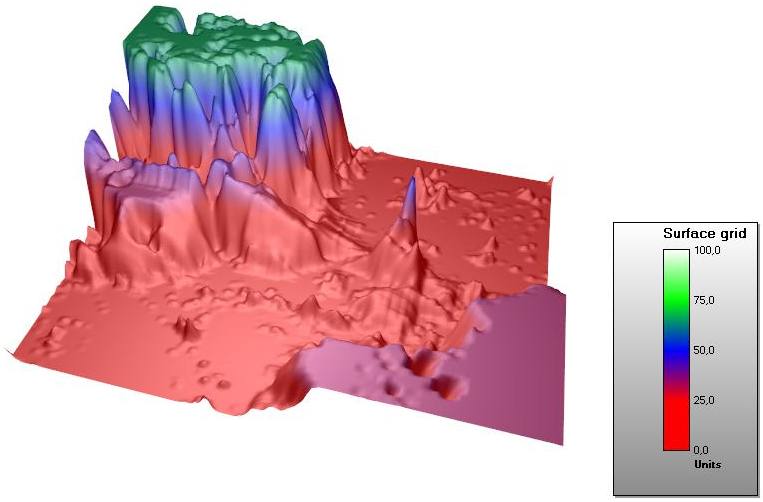
- 채우기 (7.10.3 를 보아라)
- 와이어프레임 메쉬 (7.10.5 을 보아라)
- 윤곽선 (7.10.6 을 보아라)

윤곽 팔레뜨를 위한 무한개의 단계를 정의 가능하다. 각 단계는 높이 값과 해당 색을 갖고 있다.

팔레뜨는 **MinValue**, **Type** 및 **Steps** 속성들을 포함단다. **Type** 에는 두 선택이 있다: **Uniform** 및 **Gradient.** 보기 6-39 의 윤곽 팔레뜨는 다음을 보여준다:

- **MinValue**: 0
- **Type**: 그라디언트
- **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

첫 단계 값 아래 높이 값들은 첫 단계의 색대로 칠해진다.



보기 6-39. 표면 그리드 시리즈 윤곽 팔레뜨 Type 가 Gradient 로 설정 되었다.

와이어프레임 메쉬

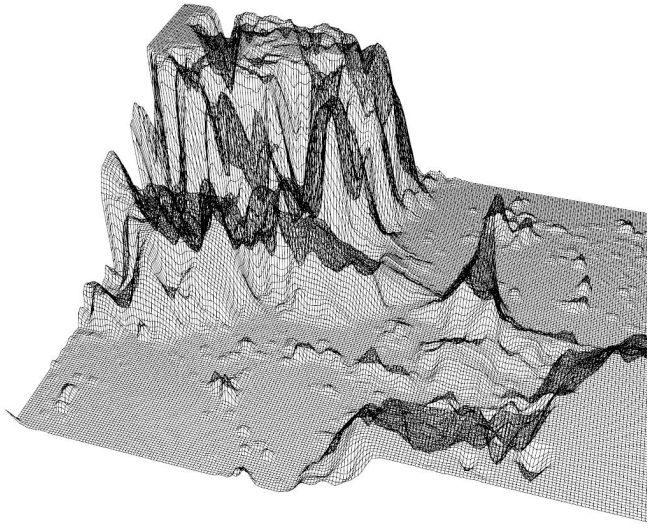
WireframeType 을 이용해 와이어프레임 스타일을 선택하라. 선택 가능 옵션은 다음과 같다:

- **None**: 와이어프레임 없음
- **Wireframe**: 단색의 와이어프레임. **WireframeLineType.Color** 으로 색을 설정하라.
- **WireframePalettedByY**: 와이어프레임 색이 SurfacePoint 의 **Y 필드 ContourPalette** 를 따른다 (7.10.4 을 보아라)
- **WireframeSourcePointColored**: 와이어프레임 색이 표면 노드의 색을 따른다.

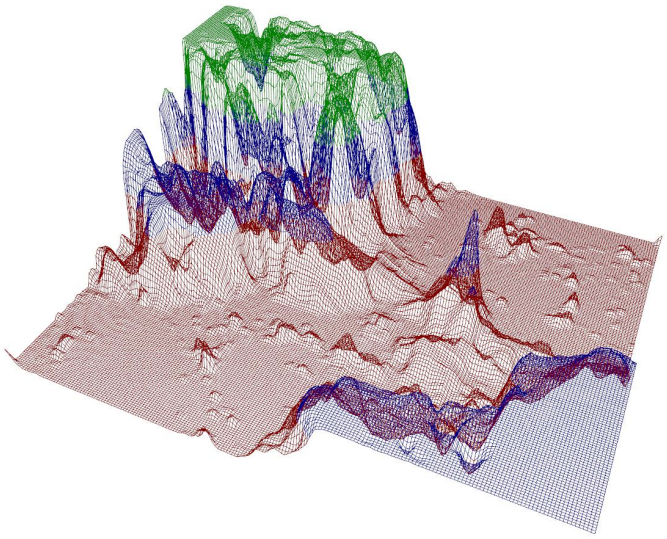
- **Dots**: 단색 점들이 노드 위치에 그려진다.
- **DotsPalettedByY**: 노드 위치에 점이 그려지고 **ContourPalette** SurfacePoints **Y** 필드 대로 색칠된다.
- **DotsPalettedByValue**: 노드 위치에 점이 그려지고 **ContourPalette** SurfacePoints 의 **Value** 필 드 대로 칠해진다.
- **DotsSourcePointColored**: 노드 위치에 점들이 그려지고 표면 노드의 색을 따른다.

와이어프레임 라인 스타일 (**색, 넓이, 패턴**) 은 **WireframeLineStyle** 을 통해 수정 가능하다.

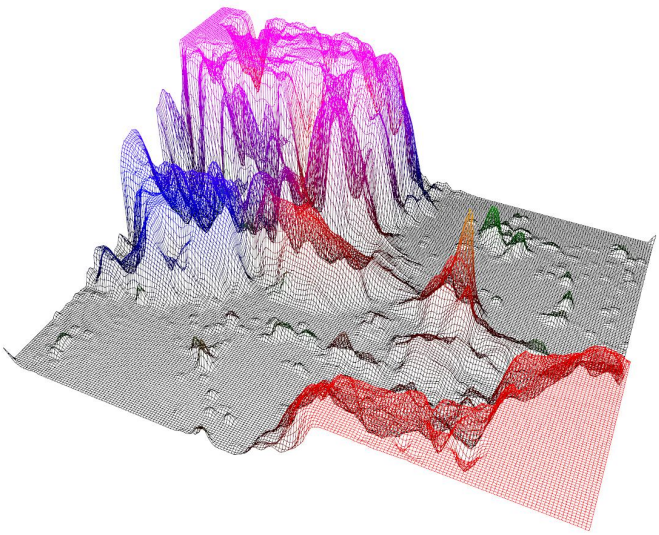
주 의 ! 팔 레 뜨 로 칠 해 진 와 이 어 프 레 인 라 인 및 점 은 **WireframeLineStyle.Width = 1** 및 **WireframeLineStyle.Pattern = Solid** 일때만 사용 가능하다.

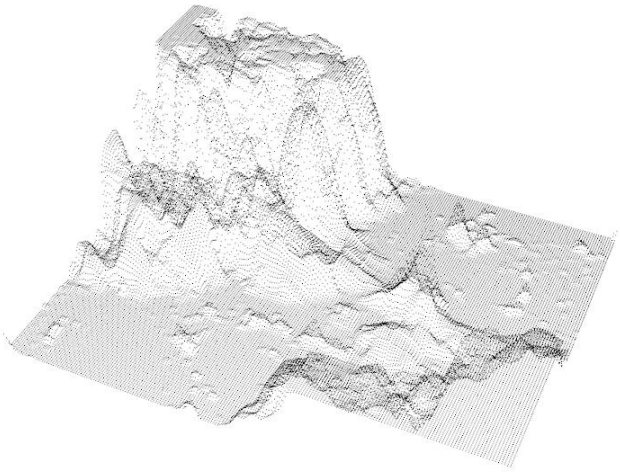


보기 6-40. WireframeType = Wireframe.



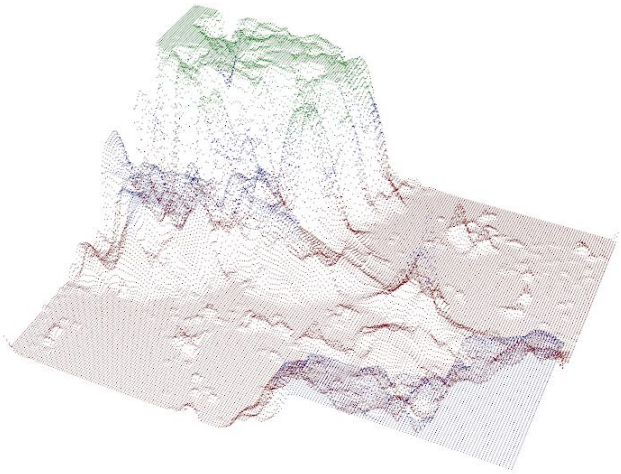
보기 6-41. WireframeType = WireframePalettedByY.





보기 6-42. WireframeType = SourcePointColored.

보기 6-43. WireframeType = Dots.

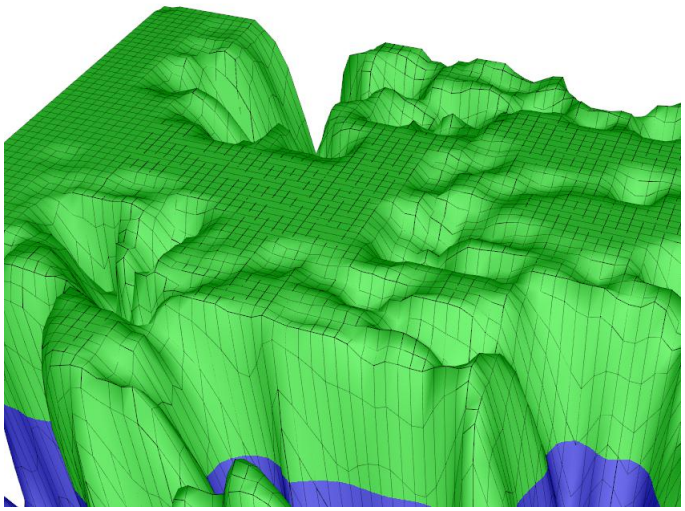


보기 6-44. WireframeType = DotsPalettedByY.

보기 6-45. WireframeType = DotsSourcePointColored.

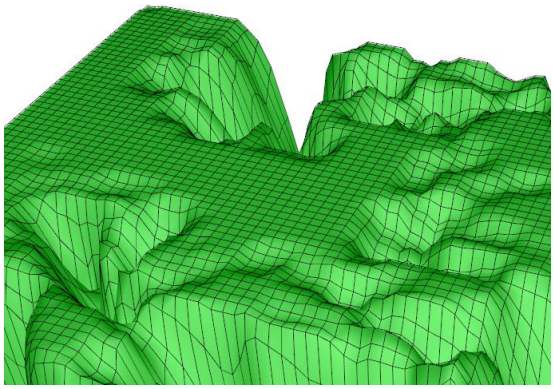
6.10.5.1 와이어프레임을 채우기와 동시에 사용할 때 주의 점

채우기 및 와이어프레임이 3D 모델의 같은 위치에 그려지면 z-싸움이 일어날 수 있다. 이는 끊어진 와이어프레임 라인으로 나타난다. 이는 GPU 가 어느 객체가 카메라에 더 가까운지 구별을 못하기 때문이다.

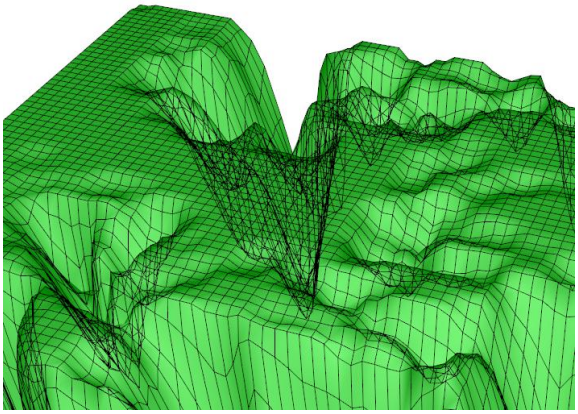


보기 6-46. 표면 그리드 와이어프레임과 채우기. z 싸움이 일어나 끊어진 와이어프레임 라인으로 나타난다.

Z-싸움이 일어나는 것을 방지하기 위해 **WireframeOffset** 또는 **DrawWireframeThrough** 속성을 이용하라. **WireframeOffset** 을 사용하면 와이어프레임이 3D 모델 공간에서 살짝 위치가 움직여 진다. **DrawWireframeThrough** 은 와이어프레임을 채우기를 뚫고 그린다. 표면이 카메라에 보이든 안 보이든 표시된다.



보기 6- 47. WireframeOffset = (X=0; Y=0.1; Z=0).

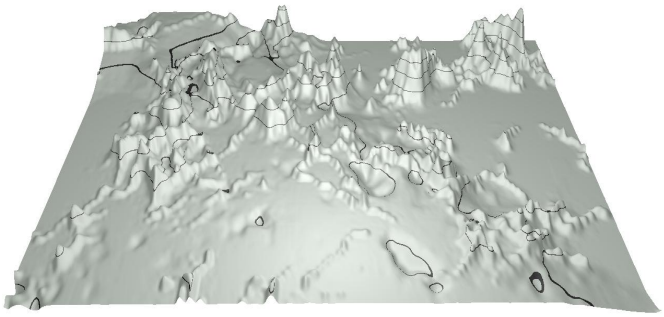


보기 6- 48. DrawWireframeThrough 가 활성화 되었다

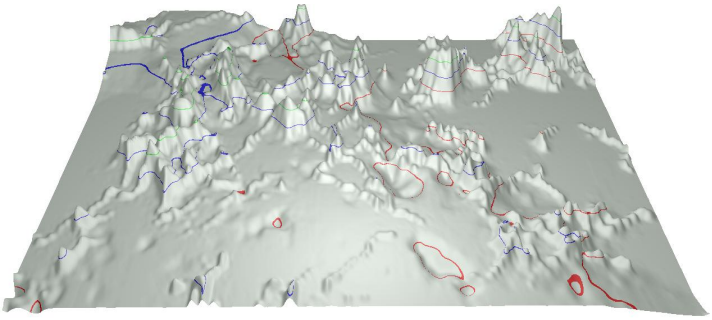
윤곽선

윤곽선은 표면을 팔레뜨 채우기로 채우지 않고 높이 데이터의 빠른 분석을 허용한다. 윤곽선은 채우기와 와이어프레임과 같이 사용 가능하다. **ContourLineType** 속성을 설정함으로 윤곽선을 여러 스타일로 그릴 수 있다:

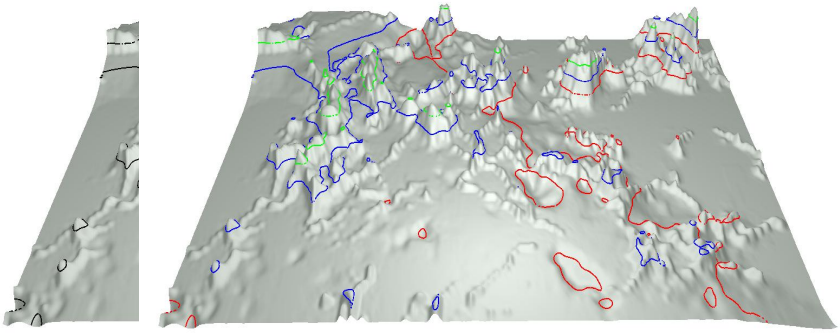
- **None**: 윤곽선 없음
- **FastColorZones**: 라인이 얇은 수직 구역으로 그려진다. 아주 강한 렌더링을 허용한다. 계속 업데이트 되거나 동기적인 표면에 유용하다. 가파른 높이 변화는 얇은 선으로 천천히 바뀌는 높이 차이는 굵은 선으로 표시 된다. 모든 선들이 **ContourLineStyle.Color** 속성에 정의된 색 대로 칠해진다. 구역 높이는 **FastContourZoneRange** 속성으로 설정 가능하다.
- **FastPalettedZones**: **FastColorZones** 와 같지만 색은 **ContourPalette** 을 따른다 (7.10.4 를 보아라).
- **ColorLineByY 및 ColorLineByValue**: 실제 선으로 윤곽선이 생성된다. **FastColorZones** 보다 렌더링이 오래 걸린다. 선 넓이는 **ContourLineStyle.Width** 속성으로 수정 가능하다. 윤곽선의 위치를 **WireframeOffset** 속성으로 움직여 채우기와 Z 싸움을 방지할 수 있다.
- **PalettedLineByY 및 PalettedLineByValue**: **ColorLineByY 및 ColorLineByValue** 과 같지만 색은 **ContourPalette** 옵션을 따른다 (7.10.4 을 보아라).



보기 6- 49. ContourLineType = FastColorZones.

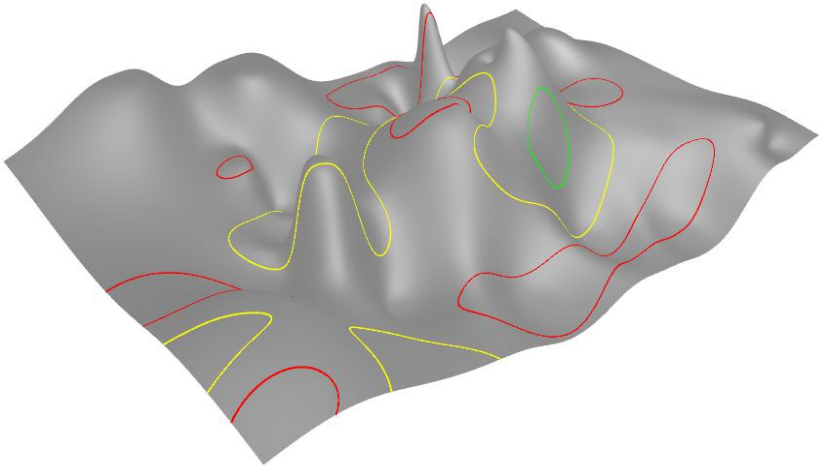


보기 6- 50. ContourLineType = FastPalettedZones.



보기 6- 51. ContourLineType = ColorLine.

보기 6- 52. ContourLineType = PalettedLine.



보기 6- 53. ContourLineType = PalettedLineByValue.

퇴색

데모 예시: 스펙트럼 3D

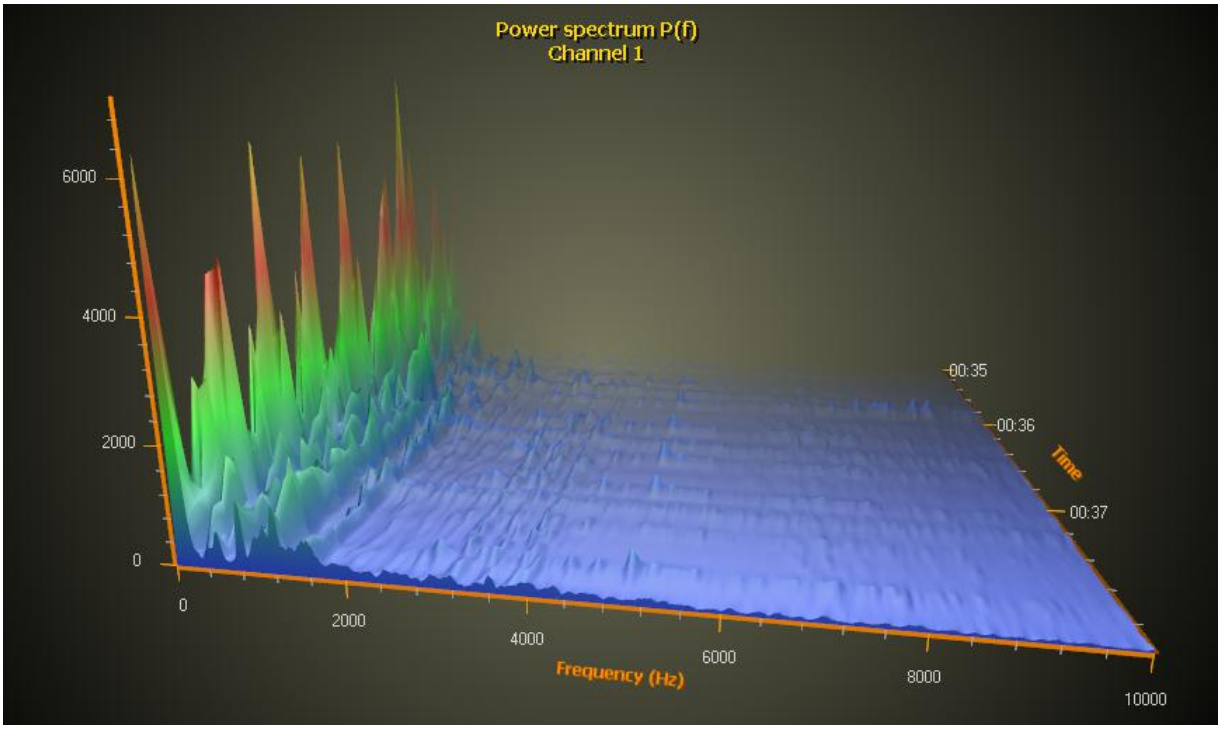
SurfaceGridSeries3D, **SurfaceMeshSeries3D** 및 **WaterfallSeries3D** 는 **FadeAway** 속성이 있어 차트 뒤쪽으로 갈수로 시리즈가 퇴색되는 것을 표현한다. **Fadeaway** 는 퍼센트로 계산되고 허용 범위는 0 (기본 값, 퇴색 없음)에서 100 (완전 퇴색) 이다. 값이 높을수록 높은 Z 값의 데이터가 더욱 투명해진다.

표면 데이터 스크롤링

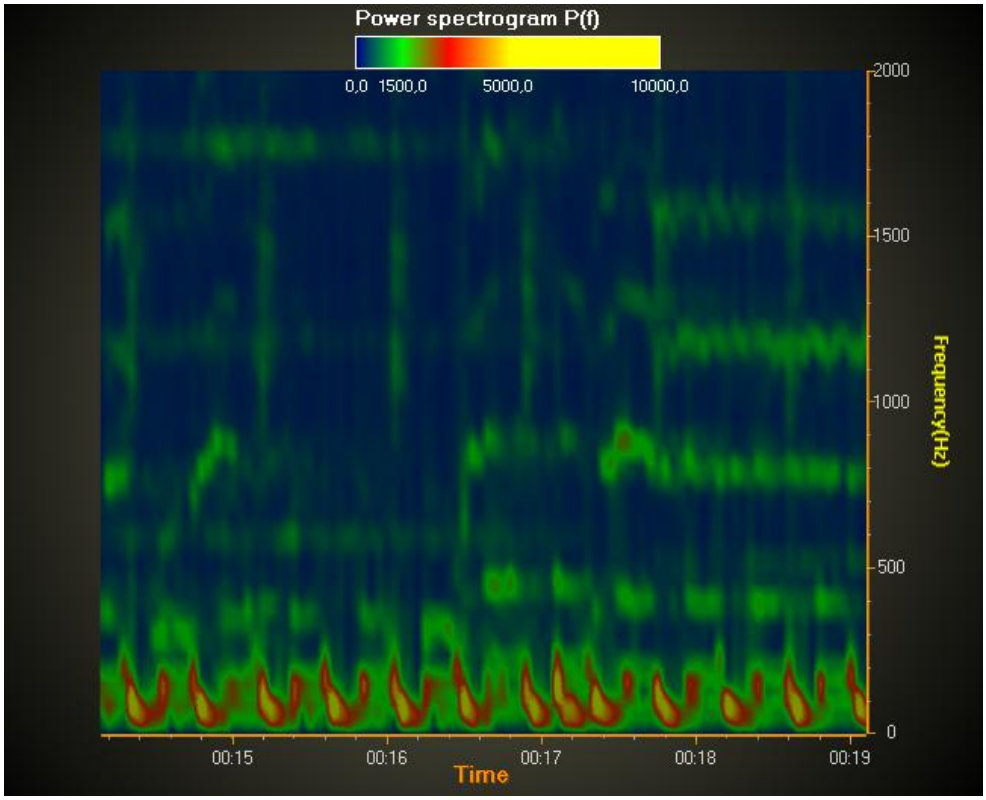
데모 예시: 스펙토그램

SurfaceGridSeries3D 및 **SurfaceMeshSeries3D** 은 **InsertRowBackAndScroll** 및 **InsertColumnBackAndScroll** 메소드로 성능 최적화 된 주기적 데이터 추가를 갖고 있다. 이 메소드들은 신규 데이터 열 또는 행을 표면 시리즈 데이터 표에 삽입하고 동시에 제일 오래 된 값을 떨군다. 다음 3D 스펙트럼 디스플레이 (보기 6-54)를 보아라. 새로운 FFT 값들이 (카메라에 가까운) 새로운 열로 추가 되었고 옛날 데이터 및 시간 축 (Z 축)을 스크롤해야 한다. 제일 오래 된 표면 값은 떨궈야 한다.

InsertRowBackAndScroll 및 **InsertColumnBackAndScroll** 은 신규 데이터를 더블 배열로 받지만 표면 시리즈와 스로롤 축 (**InsertRowBackAndScroll** 에는 Z 차원/축 및 **InsertColumnBackAndScroll** 에는 X 차원/축)을 위한 새로운 최소 및 최대 값도 필요하다. 이 지속된 시리즈 수정 및 축 범위는 스르롤링 효과를 활성화 한다.



보기 6-54. 표면 그리드 있는 3D 스펙트럼. InsertRowBackAndScroll 메소드를 성능 최적화 된 데이터 추가를 위해 사용했다. Fadeaway 속성은 100 으로 설정 되어 표면이 차트 뒤쪽으로 갈수로 자연스럽게 퇴색한다. 투시용 카메라가 사용 됐다.



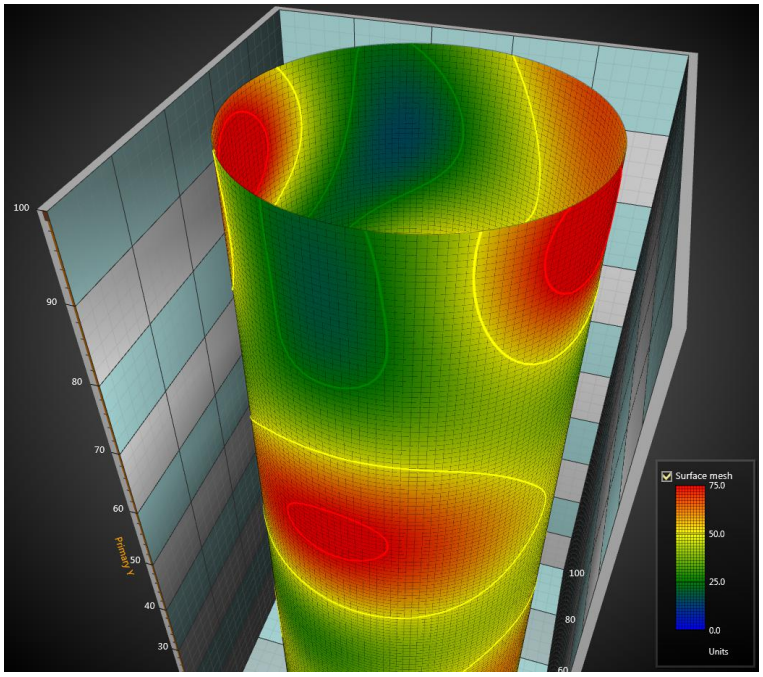
보기 6-55. 표면 그리드가 있는 스펙토그램. InsertColumnBackAndScroll 메소드가 사용 됐다. 직교 카메라가 모델 위에서 사용

되어 직선 및 수직 투사가 주어진다. SuppressLighting 가 활성화 되어 원치 않는 빛 반사를 제거 했다. Fadeaway = 0 으로 설정 되어 그리드 시리즈가 완전히 보인다.

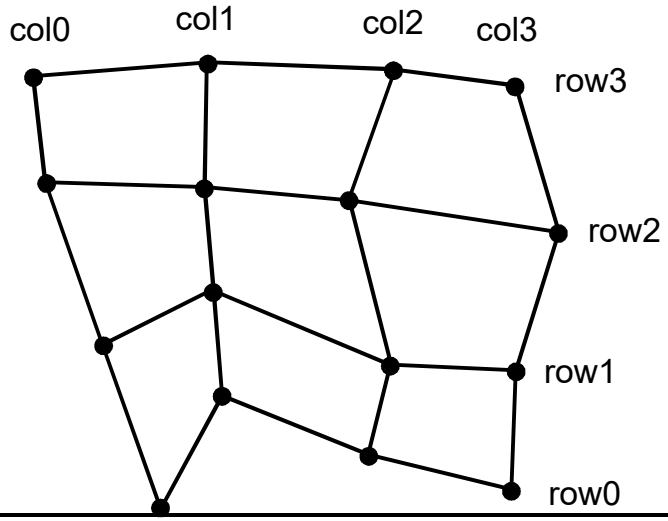
6.11 SurfaceMeshSeries3D

데모 예시: 표면 메쉬, 열 소산; 표면 메쉬; 스테핑 표면 메쉬; 비행 경로 있는 지구; 그라디언트 막대

SurfaceMeshSeries3D 는 **SurfaceGridSeries3D** 와 아주 비슷하고 같은 속성을 공유한다. 제일 큰 차이점은 표면 노드가 3 차원 공간에서 자유롭게 놓일 수 있다는 것이다. 그래서 표면이 직사각형이 아니어도 된다는 것이다. **SurfaceMeshSeries3D** 는 표면을 아무 모양대로 설정 가능하다. 원 또는 사람 머리 모양대로 설정 가능하다.



보기 6- 56. SurfaceMeshSeries3D, 파이프로 만든 형상.



보기 6-57. 표면 메쉬 노드. SizeX = 4, SizeZ =4.

표면 메쉬 데이터 설정

- **SizeX** 및 **SizeZ** 속성들을 설정해 그리드를 열과 행 기반 크기를 주어라
- 모든 노드의 X, Y, Z 값을 설정하라

데이터 배열 인덱스를 사용한 방식

```
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
```

SetDataValue 를 이용한 다른 방식

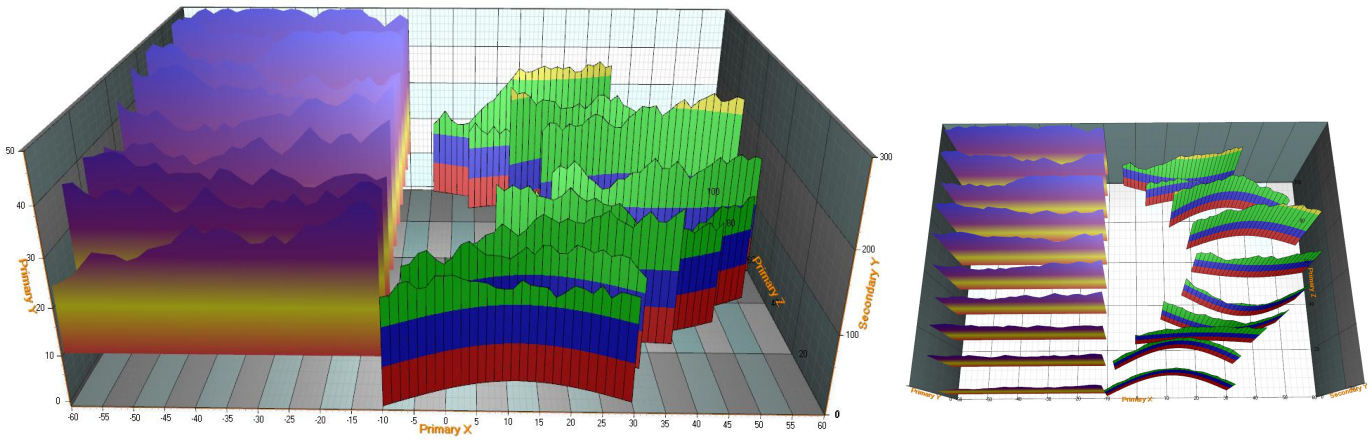
```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)
{
        for (int nodeIndexZ = 0; nodeIndexZ < rowCount; nodeIndexZ ++)
        {
          meshSeries.SetDataValue(nodeIndexX, nodeIndexZ,
                    xValue,
                    yValue,
                    zValue,
```

```
                    dataValue,
                    Color.Green); //Source point colors are not used in this
                    example, so use any color here
            }
        }
    meshSeries.InvalidateData(); //Notify new values are ready to refresh
```

6.12 WaterfallSeries3D

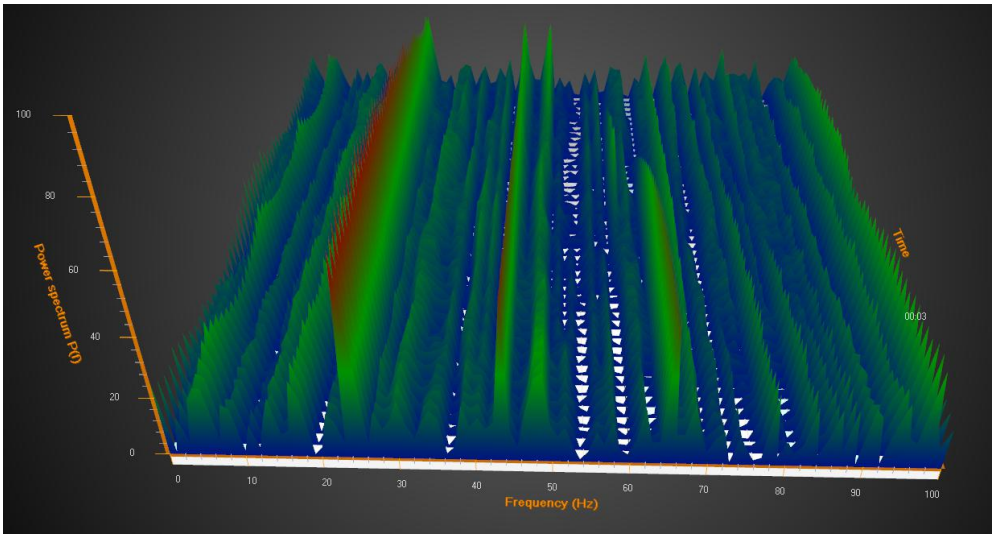
데모 예시: 폭포 3D

WaterfallSeries3D 와는 데이터가 구역 스트립으로 시각화 가능하다. 구역은 **SurfaceGridSeries3D** 와 같이 채우고 와이어프레임 생성 및 윤곽선을 만들 수 있다. 7.10 장을 보아라. Y 차원에서는 구역이 **BaseLevel** 속성 값에서 부터 시작 된다. 노드 데이터가 **SurfaceMeshSeries3D** 에서와 같이 설정 가능하다. 7.11.1 을 보아라.



보기 6-58. 두 폭포 시리즈. 왼쪽 보라색 시리즈에는 X 와 Z 가 직사각형으로 되어 있다. BaseLevel = 10. 오른쪽 빨-녹-파 시리즈에는 X 와 Z 값들이 휘었다. 각 열이 다른 수평 위치에 놓여졌다.

WaterfallSeries3D 는 정통 3 차원 스펙트럼을 표시하는 데에 유용하다.



보기 6-59. 전통 스펙트럼 표현을 위해 사용된 폭포 시리즈.

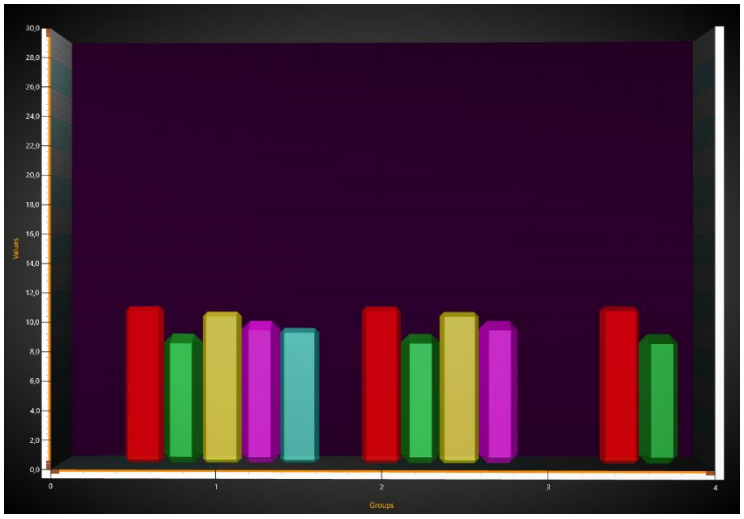
6.13 BarSeries3D

데모 예시: 수평 막대; 막대, 그룹; 막대, 맨해튼

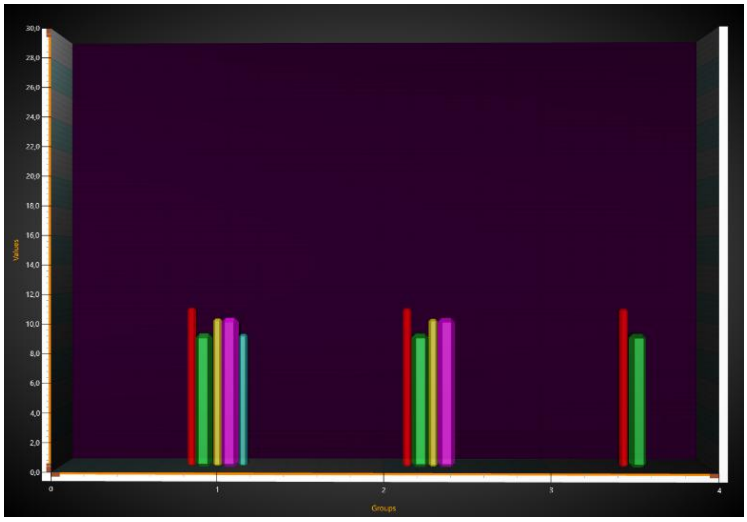
BarSeries3D 는 3 차원에서 막대 데이터 시각화를 할 수 있다.

막대 그룹화

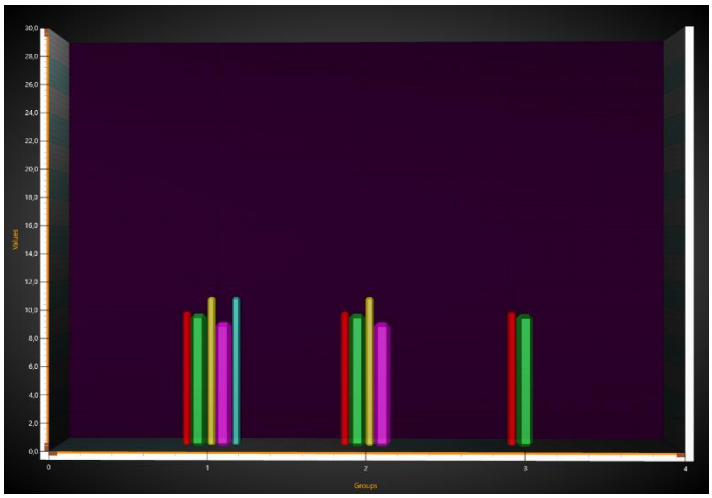
막대 시리즈는 View3D 의 여러 **BarViewOptions** 속성 옵션으로 그룹할 수가 있다. **BarViewOptions.ViewGrouping** 로 3 차원 뷰에 막대가 어떻게 묶이는지 제어할 수 있다.



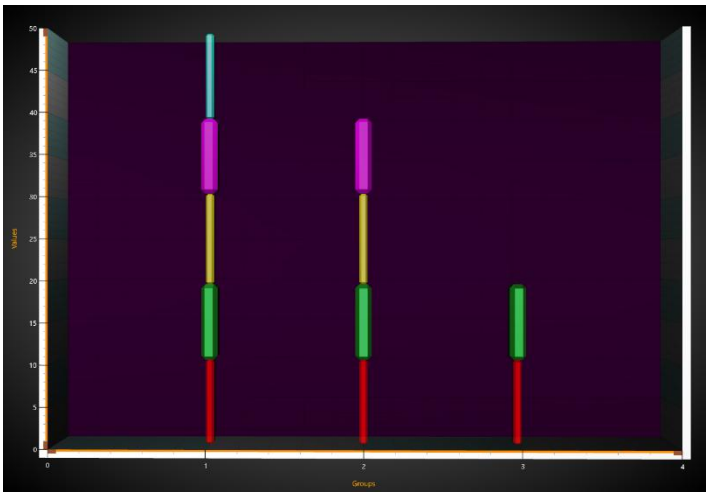
보기 6-60. BarViewOptions.ViewGrouping = GroupedIndexedFitWidth. 막대가 인덱스 대로 그룹화 되었다. 막대 넓이 및 그룹 간격은 전체 차트 넓이에 알맞게 설정 되었다.



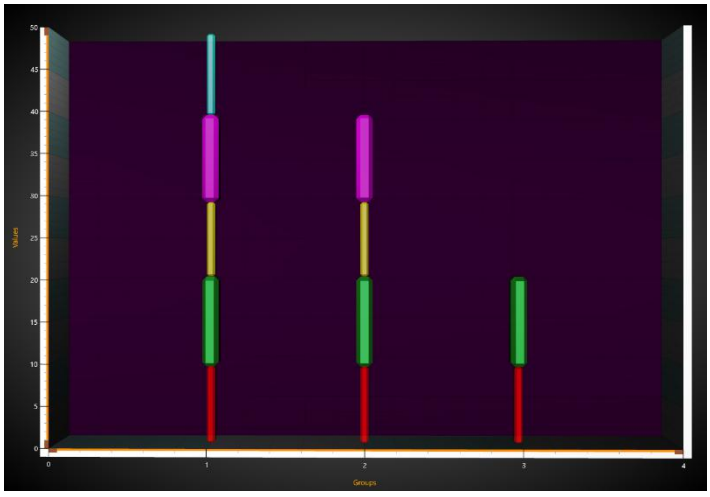
보기 6- 61. BarViewOptions.ViewGrouping = GroupedIndexed. 원래 막대 넓이가 적용되고 그룹이 차트 넓이에 맞게 나열 되었다.



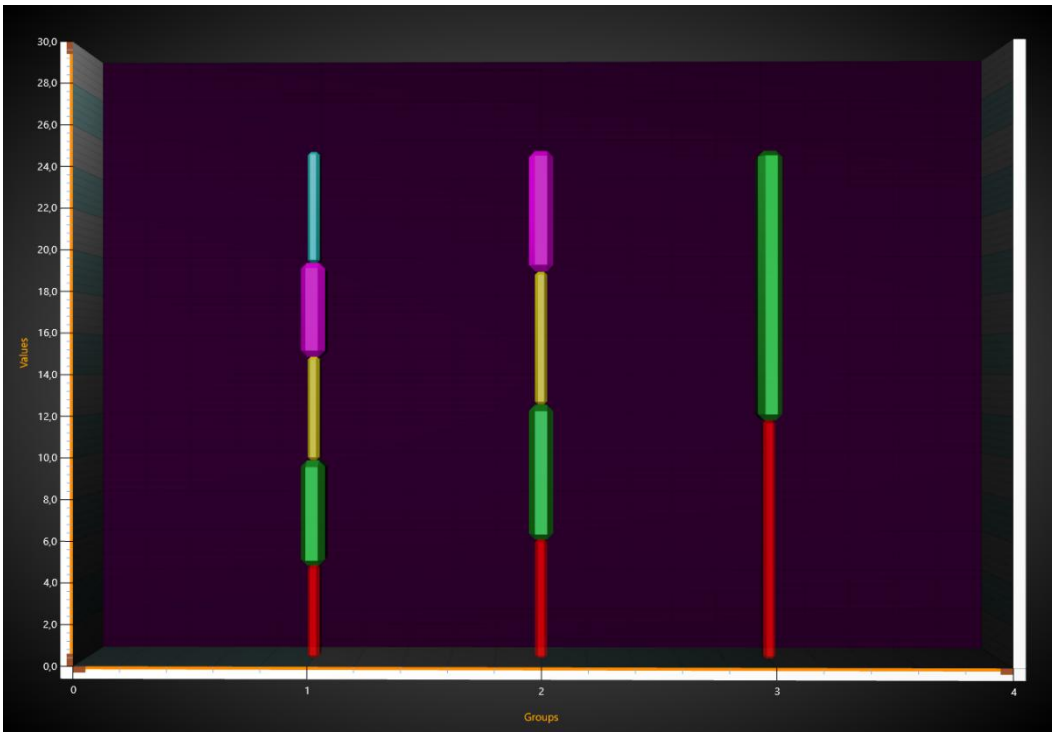
보기 6- 62. BarViewOptions.ViewGrouping = GroupedByXValue. 막대 X 값 적용.



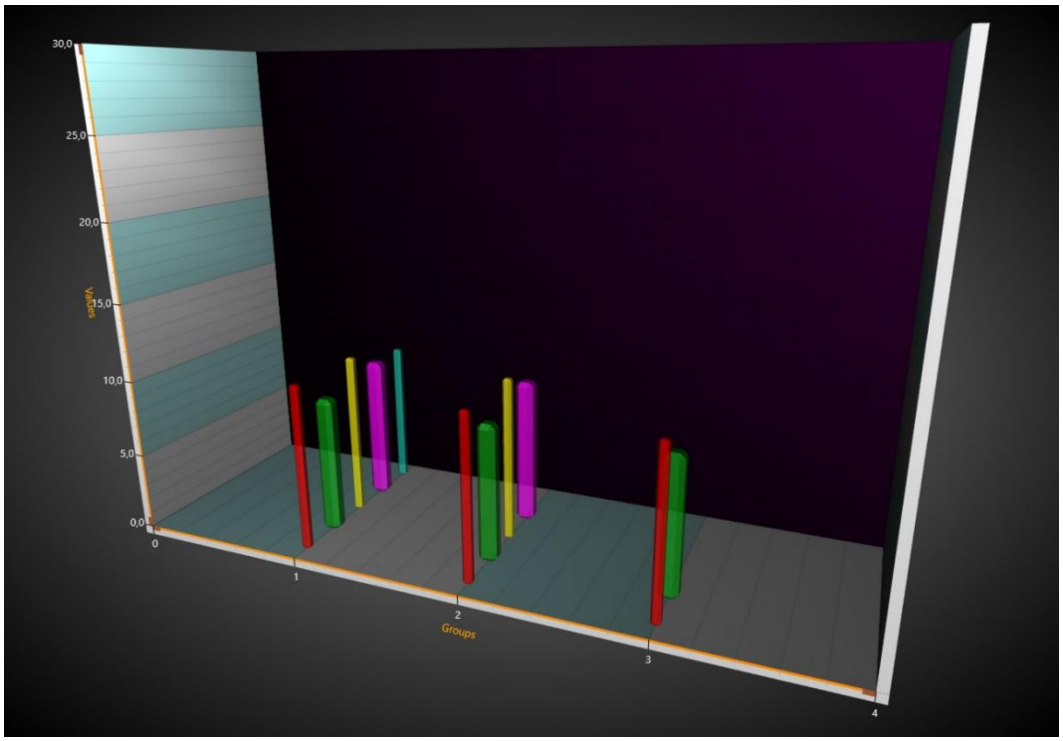
보기 6- 63. BarViewOptions.ViewGrouping = StackedIndexed. 같은 인덱스를 공유하는 모든 막대가 스택 되었다



보기 6- 64. BarViewOptions.ViewGrouping = StackedByXValue. 같은 X 값을 가진 막대 모두가 스택 되었다. 이 예시는 StackedIndexed 와 똑같이 생겼지만 이는 X 값과 지수가 같기 때문이다.



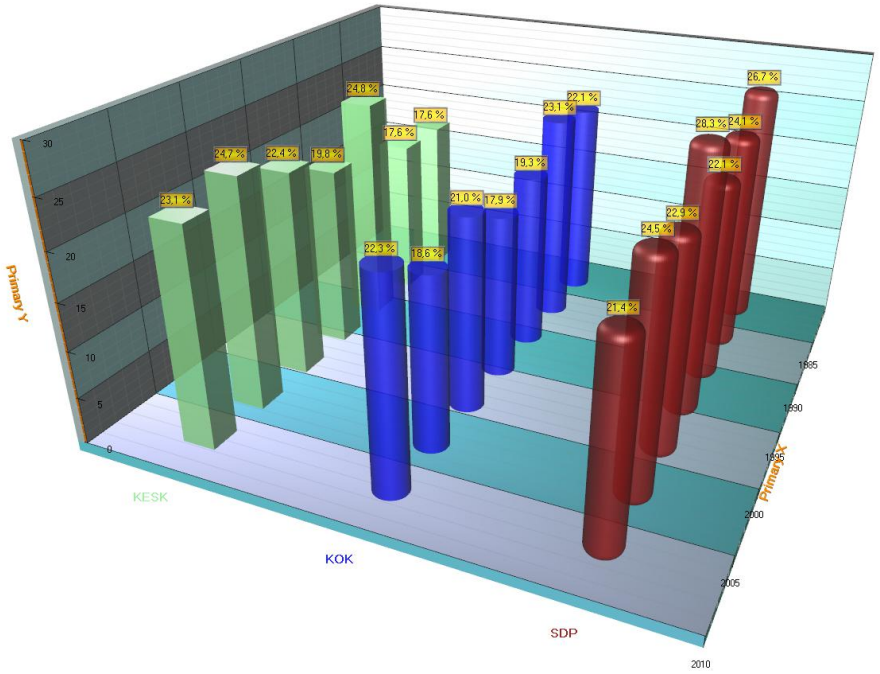
보기 6-65. BarViewOptions.ViewGrouping = StackedStretchedToSum. 같은 X 값을 가진 모든 막대가 스택 되고 StackSum 까지 늘려졌다. 이 경우에는 25 로 설정.



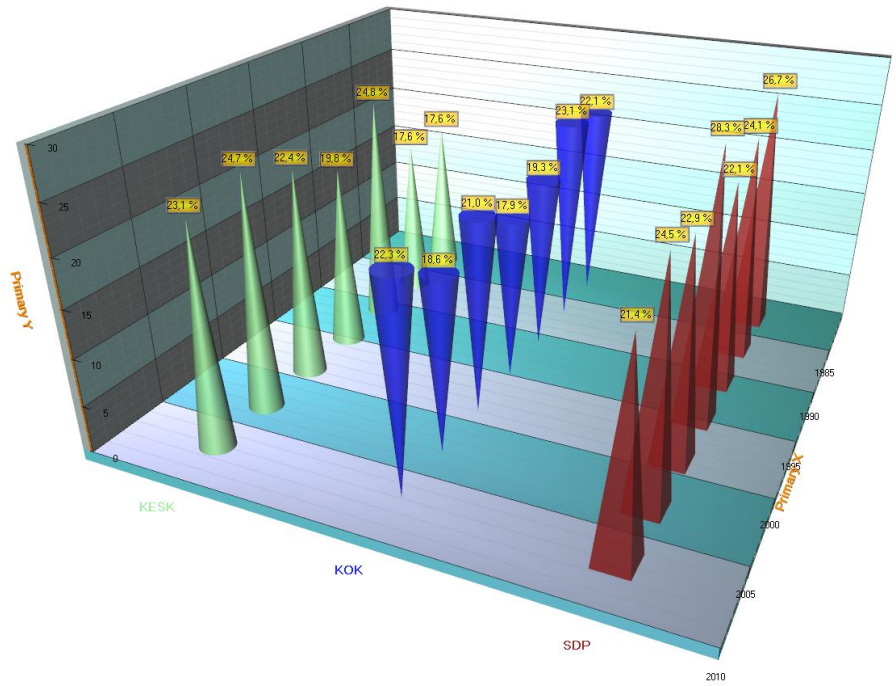
보기 6-66. BarViewOptions.ViewGrouping = Manhattan. 첫 시리즈 값은 카메라에 가장 가깝고 마지막 시리즈가 제일 멀다. 막대 x 값은 x 차원에서 막대 위치를 제어한다.

막대 스타일

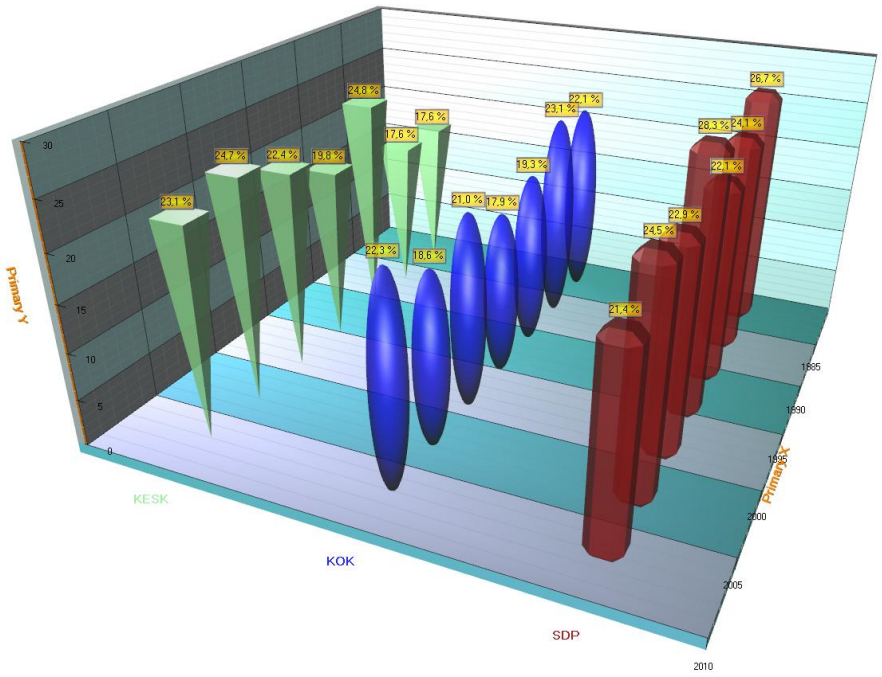
BarSeries3D 는 막대 모양을 제어하기 위해 **Shape** 속성이 있다. 어느 모양에는 **CornerPercentage** 를 변경하여 모서리를 둥글게 만들 수 있고 **DetailLevel** 로 시각적 품질을 변경할 수 있다다.



보기 6-67. 막대 모양: 간단, 원형 및 둥근 원형.



보기 6-68. 막대 모양: 원형 뿔, 역 원형 뿔 및 피라미드.



보기 6-69. 막대 모양: 역피라미드, 타원체 및 경 사진.

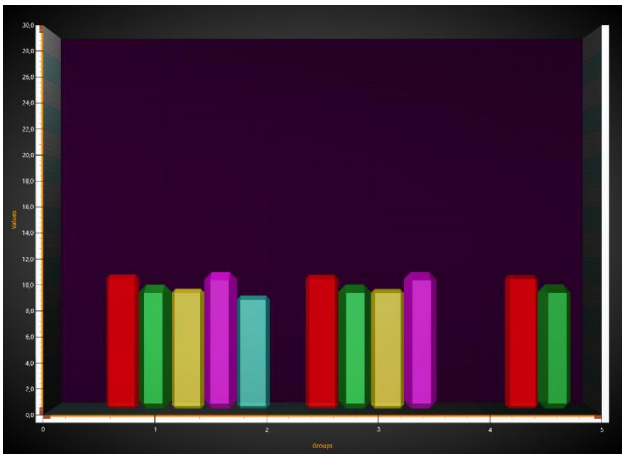
막대 시리즈 데이터 설정

막대 시리즈 데이터는 **BarSeriesValue3D** -구조로 추가 가능하다. 이는 **x**, **y**, **z** 및 **텍스트** 필드를 갖고 있다.

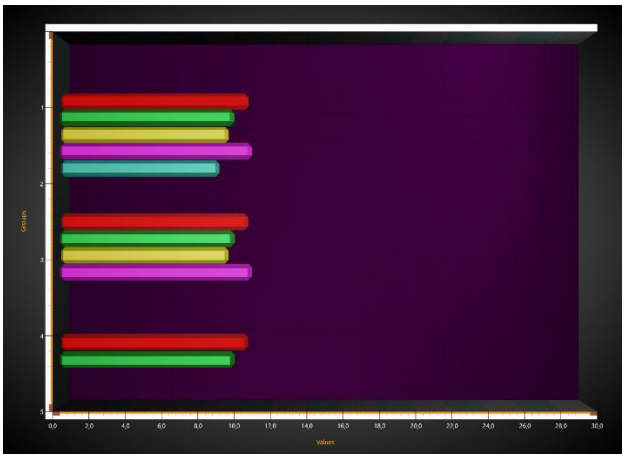
```
// create new values array
BarSeriesValue3D[] values = new BarSeriesValue3D[3];
values[0] = new BarSeriesValue3D(20, 45, 5, "");
values[1] = new BarSeriesValue3D(30, 50, 5, "");
values[2] = new BarSeriesValue3D(40, 35, 5, "");

// add values to series
chart.View3D.BarSeries3D[0].AddValues(values, false);
```

수평으로 막대 표시

막대가 Y 축 방향으로 그려졌다. 막대를 수직으로 보여주기 위해 카메라를 90 도 회전하라.



보기 6-70. 수직 막대 뷰.



보기 6-71. 수평 막대 뷰.

수직 막대 뷰를 설정하기 위해 다음 코드를 보아라:

```
chart.BeginUpdate();
chart.View3D.Dimensions.Y = 100;
chart.View3D.Dimensions.X = 150;
chart.View3D.YAxisPrimary3D.Location = AxisYLocation3D.FrontLeft;
chart.View3D.Camera.RotationX = 0;
chart.View3D.Camera.RotationY = 0;
chart.View3D.Camera.RotationZ = 0;
chart.View3D.Camera.ViewDistance = 170;
chart.EndUpdate();
```

수평 막대 뷰를 설정하기 위해 다음 코드를 보아라:

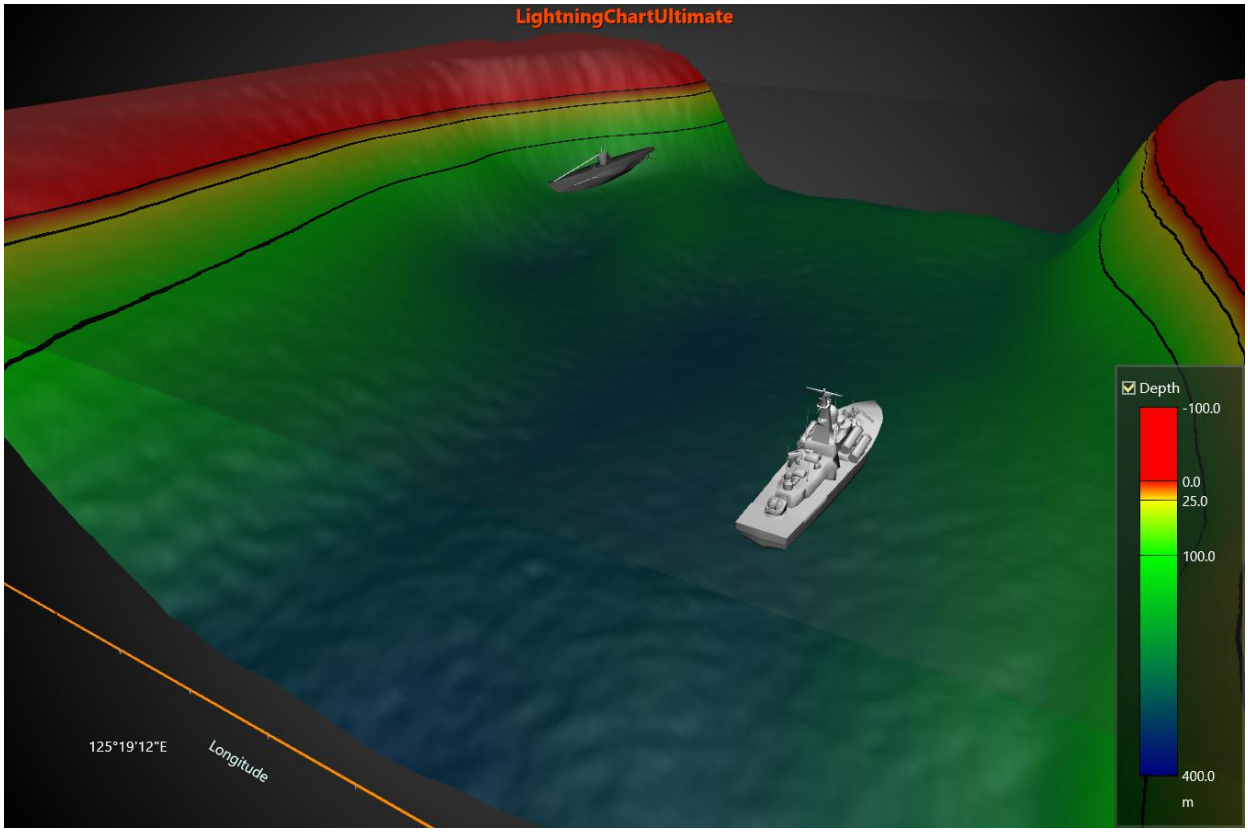
```
chart.BeginUpdate();
chart.View3D.Dimensions.Y = 150;
chart.View3D.Dimensions.X = 100;
chart.View3D.YAxisPrimary3D.Location = AxisYLocation3D.FrontRight;
chart.View3D.Camera.RotationX = 0;
chart.View3D.Camera.RotationY = 0;
chart.View3D.Camera.RotationZ = 90;
chart.View3D.Camera.ViewDistance = 170;
chart.EndUpdate();
```

6.14 MeshModels

데모 예시: 해수 있는 선박; 메쉬 모델 색칠, 와이어프레임; 메쉬 모델 식시간 색칠; 코드별 메쉬 모델

MeshModels 목록 속성으로 외부 3D 모델 에디터에서 라이트닝차트 View3D 로 3D 모델 삽입이 가능하다. 모델은 OBJ 형식으로 불러올 수 있다. 이는 3 차원 모델 어플리케이션 및 게임 엔진에 사용되는 기본 형식이다.

주의! 라이트닝차트 v.7 이후로는 Direct3D X-형식 (*.x) 을 더 이상 지원하지 않는다. DirectX 11 이 지원하지 않기 때문이다.



보기 6-72. View3D 로 로딩한 군함 및 잠수함 모델. SurfaceGridSeries3D 로 해수 깊이 시각화 위에 올렸다.

모델 로딩하기

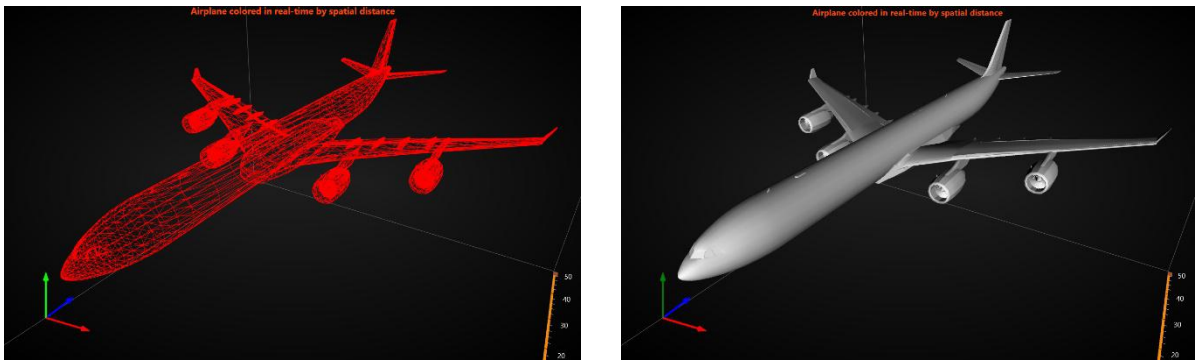
- 파일에서 모델을 로딩하기 위해 **ModelFileName** 속성에서 경로 및 파일 이름을 설정하거나 **LoadFromFile** 메소드를 사용하라. 파일에서 모델을 불러올 때 같은 경로에 존재하고 MTL 파일 및 이미지 파일들에 접근 가능하면 텍스쳐 필도 같이 로딩 된다.
- 라이트닝차트 버전 8.5 부터 **MeshModel** 생성이 .obj 파일 꼭지점에 색칠을 지원한다. 꼭지점 위치는 x,y,z 이후 빨강, 노록, 파랑 및 알파 값을 지원한다 (XYZRGBA).
- 스트림에서 모델을 로딩하기 위해 **LoadFromStream** 메소드를 이용하라. 스트림 읽기 메소드는 형상 및 기재만 읽고 텍스처는 읽지 않는다.
- 자원에서 모델을 로딩하기 위해 **LoadFromResource** 메소드를 사용하라.

모델 위치 선정, 스케일 및 회전

MeshModel 객체 **Position** 은 할당된 X, Y, Z 축을 따른다. 모델은 **Rotation** 속성을 수정함으로 회전할 수 있다. **Size** 는 **Size** 속성으로 정의 가능하다. 이는 원래 모델 크기를 위한 팩터의 컬렉션이고 축 범위 또는 3D 세계 차원을 따르지 않는다.

채우기 및 와이어프레임 활성화

- 채우기를 보여주기 위해 **Fill** = True 설정 하라.
- 와이어프레임을 보여주기 위해 **WireFrame** = True 를 설정하라. 선호 라인 색을 **WireFrameLineColor** 에서 설정하라.

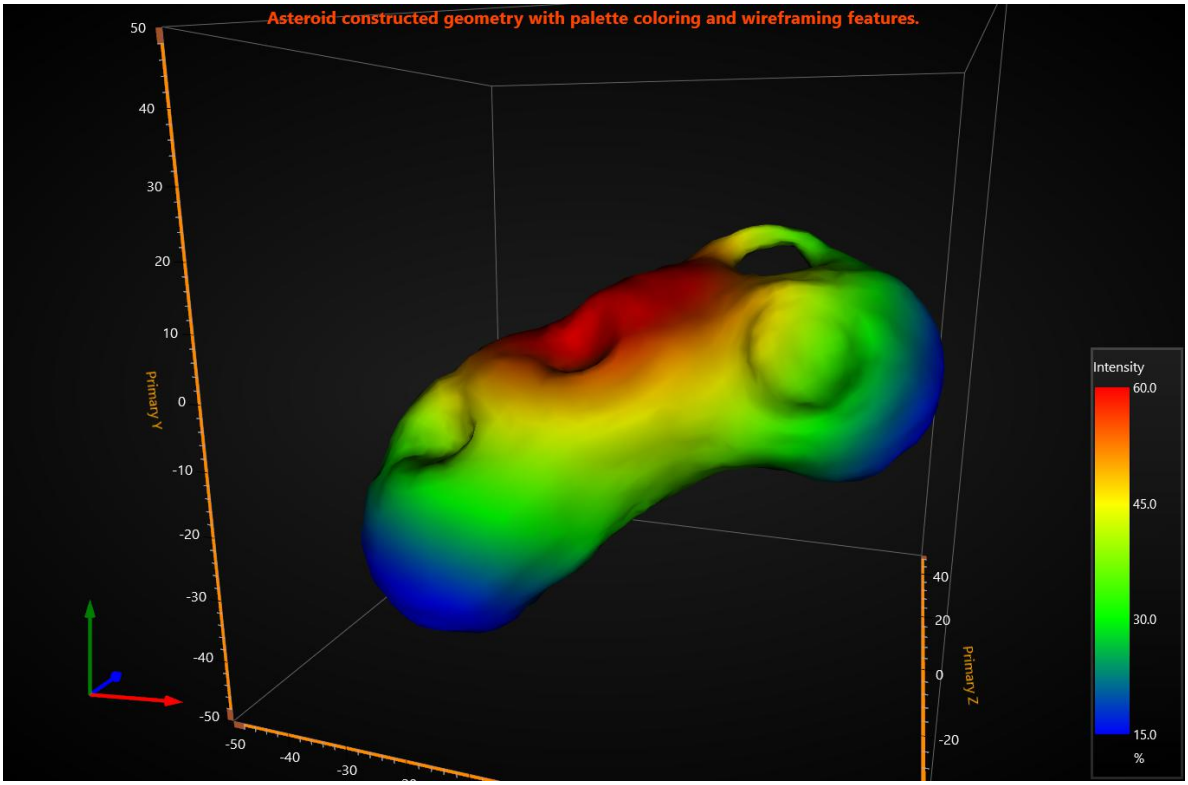


보기 6-73. 와이어프레임으로 보여지는 비행기 (WireFrameLineColor = Red). 기본 회색으로 채워졌다.

커스텀 색칠 채움

기본적으로 모델은 OBJ 색으로 렌더링 된다. 모델 꼭지점에 커스텀 색칠을 적용하기 위해 **UpdateFillColors(int[] colors)** 를 사용하라. 이 메소드는 주기적으로 불러 실시간 색 업데이트를 적용할 수 있다. **UpdateFillColors** 은 꼭지점 위치의 길이와 일치한 ARGB 색 배열이 필요하다. 한 꼭지점에는 한 색이 필요하다.

GeometryConstructed 이벤트는 축 값 공간에 꼭지점 위치를 보고한다. **X,Y,Z** 배열 등이 될 수 있다. 색을 데이터 포인트 등 기타 차트 객체의 공간적인 거리 등에 색을 적용하고 싶을 때 특히 유용하다. 초기화 단계에서 **GeometryConstructed** 이벤트 핸들러에 구독하고 더 이상 필요가 없을 때 구독 취소하라.

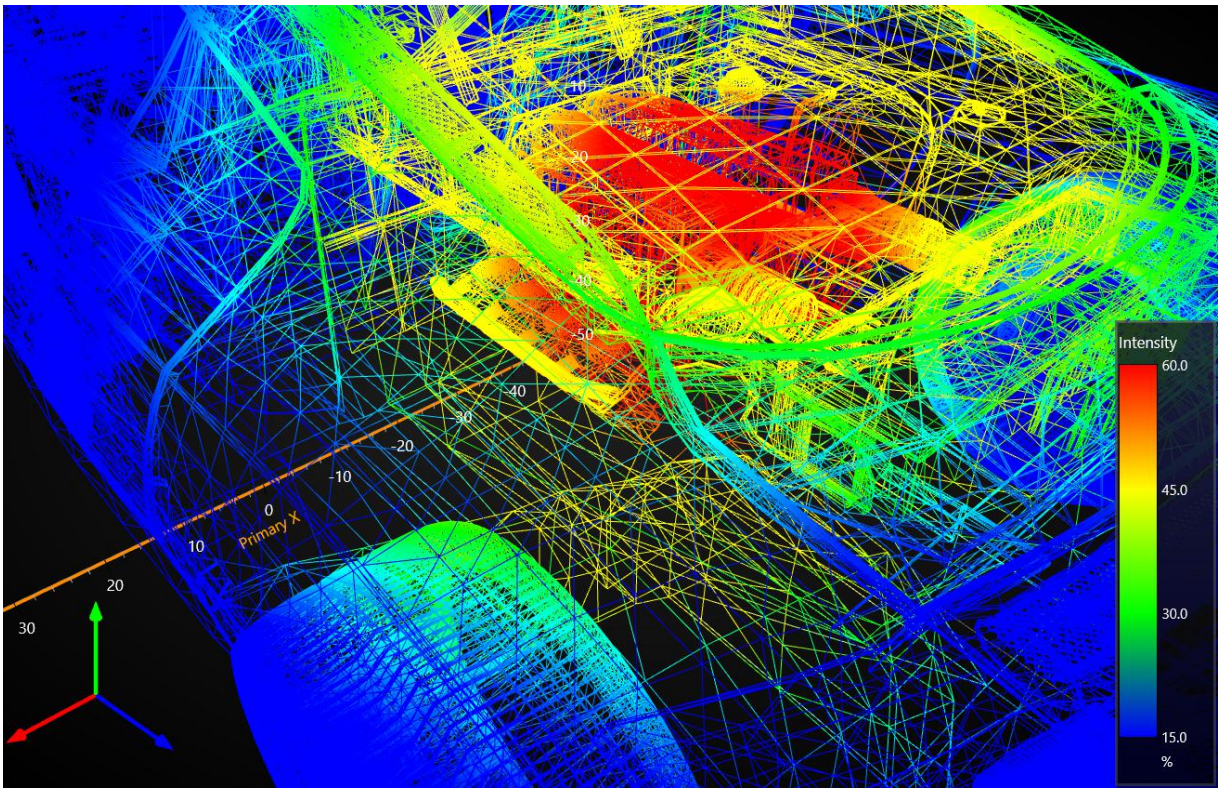


보기 6- 74. UpdateFillColors 메소드를 이용해 공간적 거리 대로 색이 적용된 MeshModel.

주의: ChartTools.ConvertDataToColorsByFixedIntervalPalette 메소드를 이용해 데이터 값을 주어진 팔레뜨 단계 별 (ARGB int) 색으로 변경할 수 있다.

와이어프레임 커스텀 색 적용

와이어프레임에도 커스텀 색을 적용할 수 있다. **GeometryConstructed** 이벤트 핸들러를 이용해 필요한 색 배열 길이를 얻어 **UpdateWireframeColors** 메소드를 이용해 새로운 색을 적용하여라.



보기 6- 75. 공간적 거리 대로 색이 적용 된 MeshModel 와이어프레임. UpdateWireframeColors 메소드를 이용해 적용 됐다.

역 꼭지점 와인딩 순서

어떤 모델들은 역 와인딩 순서로 만들어져 컬링이 보이지 않게 만든다. 모델이 제대로 나타나지 않을 경우 **Cull** 설정을 **Clockwise**, **CounterClockwise** 및 **None** 사이 변경해 보아라.

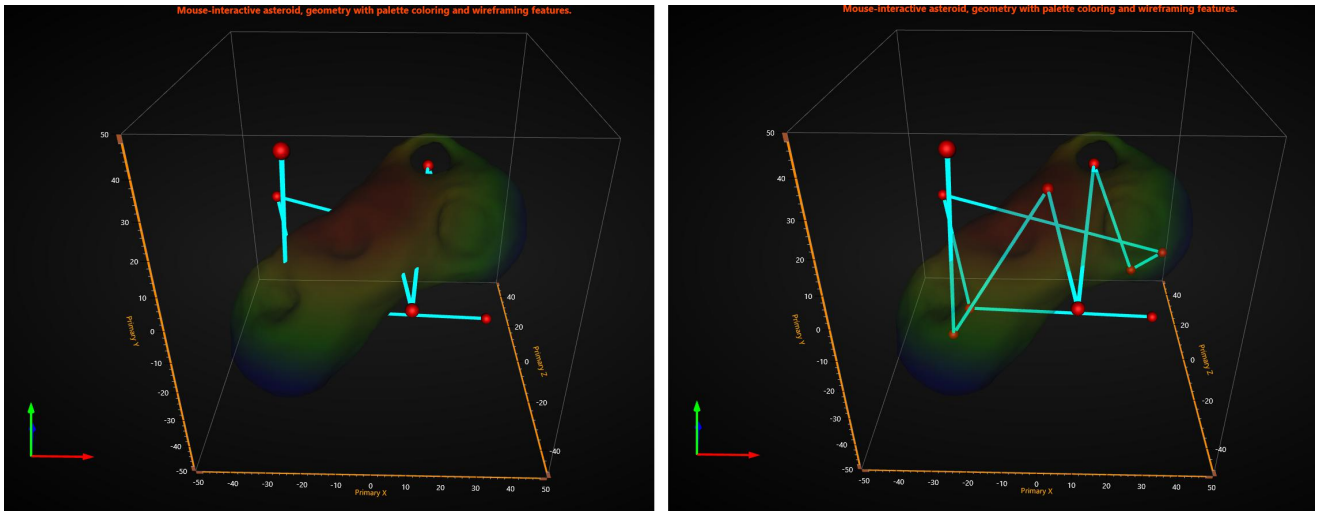
```
meshModel.Cull = Cull.CounterClockwise;
```

MeshModel 렌더링 순서

RenderingOrder 속성은 라이트닝차트 버전 8.5 에 처음 소개 되었다. 이는 MeshModel 이 PointLineSeries3D 및 SurfaceGridSeries3D 등 기타 시리즈 전에 **BeforeSeries** 로 렌더링 되는지 후에 **AfterSeries** 로 되는지 제어한다. 비슷한 **RenderingOrder** 설정을 가진 MeshModel 들은 차트에 추가된 순서대로 그려진다.

```
meshModel.RenderingOrder = MeshModelRenderingOrder.BeforeSeries;
```

RenderingOrder 는 모든 반투명 이상 MeshModel 에 효과가 있다. 이는 모델 뚫고 다른 시리즈가 보이는지를 결정하기 위해 쓰인다. MeshModel 이 투명 색을 사용하지 않는다면 **RenderingOrder** 설정과 상관 없이 뒤에 있는 모든 것들을 보이지 않게 막는다.



보기 6-76. 반투명 메쉬 모델의 RenderingOrder 이 왼쪽에는 BeforeSeries 로 설정 및 오른쪽에는 AfterSeries 로 설정 되었다. BeforeSeries 옵션으로는 모델 색이 투명 하더라도 PointLineSeries3D 등 기타 시리즈는 모델 뒤에 보이지 않는다.

현재 Rectangles3D 및 Polygons3D 는 시리즈로 취급이 안되어 **RenderingOrder** 로 설정할 수 없다.

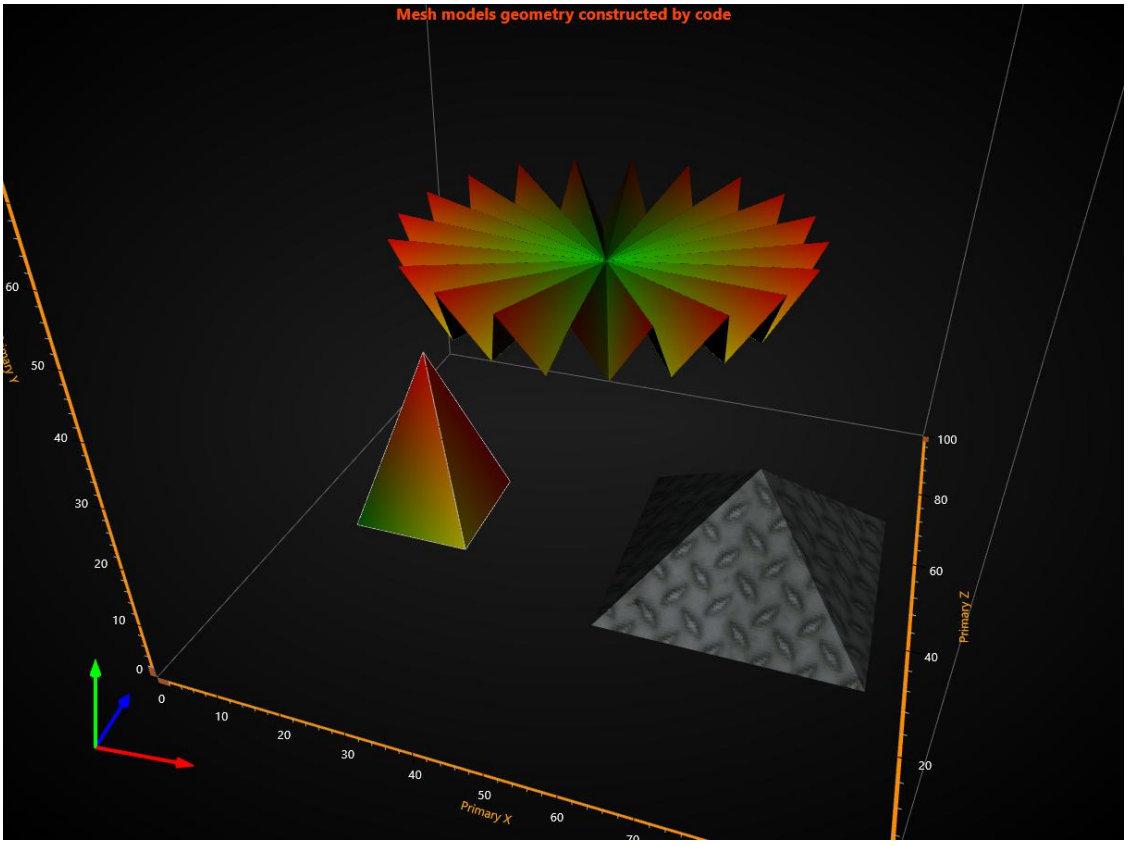
꼭지점에서 MeshModel 프로그램적으로 구성

버전 8.2 부터 **MeshModel** 은 프로그램적으로 **MeshModel** 형상 구성하는 것이 지원된다. 이는 컴퓨터 계산을 통해 생성 된 객체 및 모양을 시각화 하게 해준다.

다음과 같은 **Create** 메소드를 사용 가능하다:

- Create(위치, 색, 지수)
- Create(위치, 색, 노말, 지수)
- Create(위치, textureCoordinates, 비트맵, textureWrapMode, 지수)
- Create(위치, 노말, textureCoordinates, 비트맵, textureWrapMode, 지수)

인덱스 배열 (지수) 매개 변수는 필수 값이 아니다. 제공이 되면 어느 꼭지점, 색, 빛 노말 및 텍스처 좌표를 주어진 배열에서 사용 해야 하는지 정의한다. 여러 세모들이 같은 꼭지점을 공유할 때 지수를 사용하는 것은 리소스를 아낀다.



보기 6-76. 코드로 생성된 MeshModels.

회전, 스케일, 위치 속성 등과 이벤트는 꼭지점에서 프로그램적으로 생성 된 **MeshModel** 에도 적용이 된다. 이는 로딩 된 객체에 적용 되는 방식과 비슷하다.

6.14.8.1 비트맵 채우기를 효율적으로 업데이트

Create 메소드로 **MeshModel** 이 생성 되었을 때 비트맵 및 텍스처 좌표를 제공하는 것은 형상을 재구성 없이 비트맵을 업데이트 하여 아주 효율적이다. **UpdateFillBitmap** 메소드를 불러 업데이트 하라.

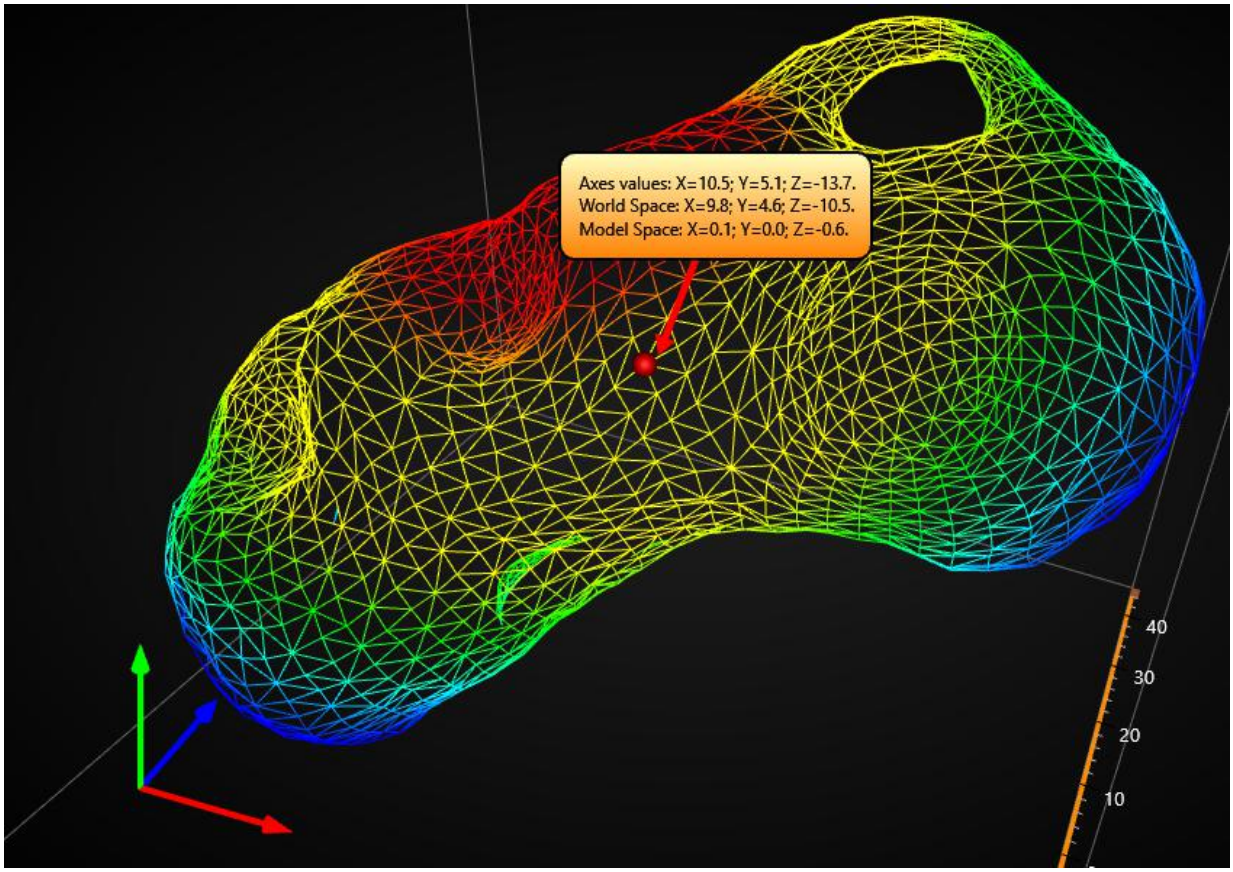
주의! UpdateFillBitmap 메소드는 OBJ 파일에서 로딩 된 모델에 적용이 안된다.

마우스로 모델 트레이싱하기

MeshModel 은 세모 기반 마우스 위치 트레이싱이 있다. **MouseTriangleTraced** 이벤트를 이용해 카메라 및 마우스 위치에 제일 가까운 세모를 찾아라.

이벤트 아규먼트는 다음과 같은 정보를 갖고있다:

- **IntersectionPointAxisValues**: 축 값에 세모 면의 교집합
- **ModelSpaceTriangleCoordinates**: 마우스가 닿은 3개의 세모 모서리 (꼭지점)의 배열을 3D 모델 스페이스 좌표로 돌려줌.
- **WorldSpaceTriangleCoordinates**: 마우스가 닿은 3개의 세모 모서리 (꼭지점)의 배열을 3D 모델 스페이스 좌표로 돌려줌.
- **NearestCoordinateIndex**: 트레이싱 된 세모의 가장 가까운 좌표 인덱스의 인덱스. 0에서 2 사이 값을 갖고 있다. 인덱스를 이용해 **ModelSpaceTriangleCoordinates** 또는 **WorldSpaceTriangleCoordinates** 배열에서 좌표를 얻을 수 있다.



보기 6-77. 마우스로 MeshModels 트레이싱. 트레이싱 결과가 주석에 보인다.

6.15 VolumeModels

데모 예시: 볼륨 헤드; 볼륨 플로우; 볼륨 지오; 볼륨 뼈대; 볼륨 웨이브 방해

VolumeModels 은 볼륨 데이터 시각화를 위한 도구이다. 직접 볼륨 렌더링을 사용한다. **VolumeModel** 는 안의 볼륨 데이터를 가져 와 시각화 한다. 라이트닝차트의 볼륨 렌더링 엔진은 **볼륨 레이 캐스팅** 기반이다.

볼륨 데이터 샘플링이 데이터 세트 내 이동하는 레이의 경로를 따라 만들어진 알고리즘을 통해 이미지가 생성된다. **볼륨 레이 캐스팅** 을 위한 하드웨어 가속의 간단한 깨알림은 볼륨 객체의 태두리를 생성하는 것이 필요하다. 보통 이들은 큐브로 표현 된다. 아티팩트 없이 고품질의 렌더링 및 교체 가능한 레이 함수가 이 기술의 가장 큰 장점들이다.

RayFunction 은 아주 높은 수준의 융통성이 있는 알고리즘의 핵심이다. 이 기술은 매우 강하다. 이는 데이터 샘플링 및 합쳐지는 방식을 지정하기 때문이다. 이는 특징 발굴을 위한 매우 유용한 도구로 만든다.

주의! VolumeModels 은 DirectX 11 렌더러가 사용 되었을 때만 사용 가능하다.

데이터 로딩

데이터를 **VolumeModel** 로 올릴 수 있는 여러 방식이 있다:

- **Data** 속성에 데이터를 데이터세트의 슬라이스를 대표하는 이미지 컬렉션으로 제공
- **VolumeModel** 의 컨스트럭터에 직접 데이터를 여러 방식 대로 제공 가능하다
- **VolumeModel** 에 데이터를 로딩 함수 중 하나를 이용해 제공 할 수 있다.

로딩 함수 및 컨스트럭터는 데이터를 슬라이스 컬렉션으로 제공하는 것을 가능케 한다 (데이터 속성과 비슷하게) 또는 슬라이스가 있는 폴더의 경로를 스트링으로 (Net 에 지원 되는 이미지 익스텐션으로) 도 제공 가능하다. 데이터를 또한 우리 도구로 생성 된 텍스처 지도로 제공 될 수 있다. 텍스처 지도는 슬라이스로 구성되었지만 이의 추가도 그림에 슬라이스 수에 대한 추가 정보가 필요하다. 이는 GPU 입력 버퍼의 효율적인 사용에 위한 것이다. ChartTools.CreateMap 로 텍스처 지도 생성 가능하다. 텍스처 지도의 직접 입력이 큰 데이터세트를 사용할 때 쓰여진다. 이는 어플리케이션의 실행을 빠르게 만들기 위해서다.

속성

VolumeModel 은 라이트닝차트 3D 객체에서 흔한 속성을 갖고 있다. 예를 들어 **Visible, Rotation, Size, Position, MouseInteraction,** 및 **MouseHighLight** 등을 갖고 있다. 추가로 객체 특수 속성이 있어 볼륨 렌더링 엔진이 어떻게 다루는지 정의한다.

Brightness	
B	**10**
G	**10**
R	**10**
Darkness	
B	**0.7**
G	**0.7**
R	**0.7**
EmptySpaceSkipping	128
MouseHighlight	Simple
MouseInteraction	True
Opacity	**0.15000000596046448**
Position	
X	**55**
Y	**50**
Z	**50**
RayFunction	**Accumulation**
Rotation	
X	**270**
Y	0
Z	**180**
SamplingRateOptions	
Enabled	**False**
Inerthness	**2**
ManualSamplingRate	512
SamplingRateRange	
TargetFPS	**15**
Size	
Depth	**100**
Height	**75**
Width	**100**
SliceRange	
Max	
X	**0.7**
Y	**0.8**
Z	**1**
Min	
X	**0.3**
Y	**0.2**
Z	0
Smoothness	**2**
Threshold	
Max	
B	**1**
G	**1**
R	**1**
Min	
B	**0.5**
G	**0.5**
R	**0.5**
Visible	True
XAxisBinding	Primary
YAxisBinding	Primary
ZAxisBinding	Primary

보기 6- 78. VolumeModels 의 속성 트리

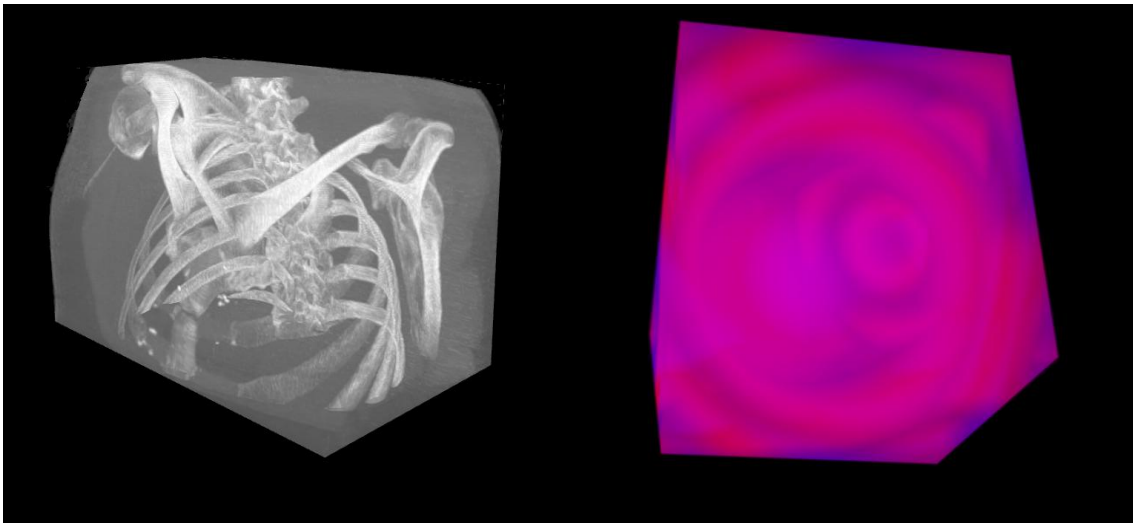
레이 함수

RayFunction 속성은 라이트닝차트 볼륨 렌더링 엔진에서 사용 가능한 복슬 샘플링 및 구성의 3 가지 방식 중 하나 선택 하게 해준다:

- **RayFunction.Accumulation** 은 데이터를 최대한 많이 모아 합친다. 이 기술로 생성된 시각화는 반투명 젤과 비슷하다. 아래 보기가 의료 데이터세트를 시각화한 **RayFunction.Accumulation** 응용 프로그램의 예시를 보여주고 있다.

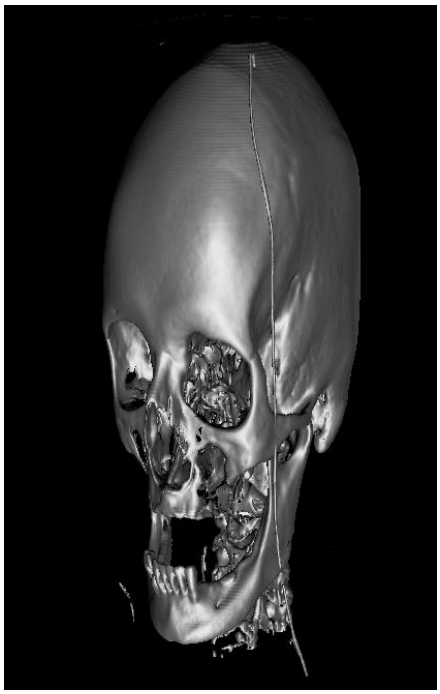
보기 6- 79. RayFunction.Accumulation 을 위한 의료 응용 프로그램의 예시

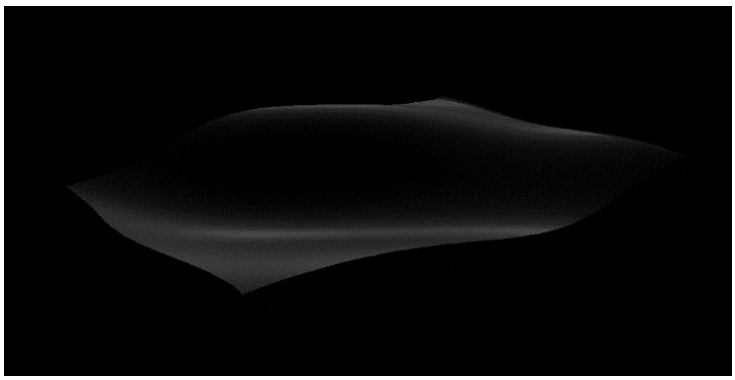
- **RayFunction.MaximalIntensity** 은 레이로 샘플 된 가장 밝은 값들만 고려한다. 시각적으로는 x레이와 매우 비슷한 결과가 발생한다. 이는 객체의 내구 구조에 대한 추가 정보를 얻게 해준다. **RayFunction.MaximalIntensity** 를 뼈대 시각화 및 초음파 웨이브 방해 시뮬레이션에 응용이 아래에 보여졌다.



보기 6-80. 최대 강도 레이 함수 응용의 예시

- **RayFunction.Isosurface** 는 다각형 모델 렌더링과 비슷하게 생긴 모델 표면을 그린다. 결과물은 **Indirect Volume Rendering** 으로 생성 된 모델과 매우 비슷하다. 아래 보기들은 인간 해골 CT 및 물 흐름의 시뮬레이션 시각화를 보여주고 있다.

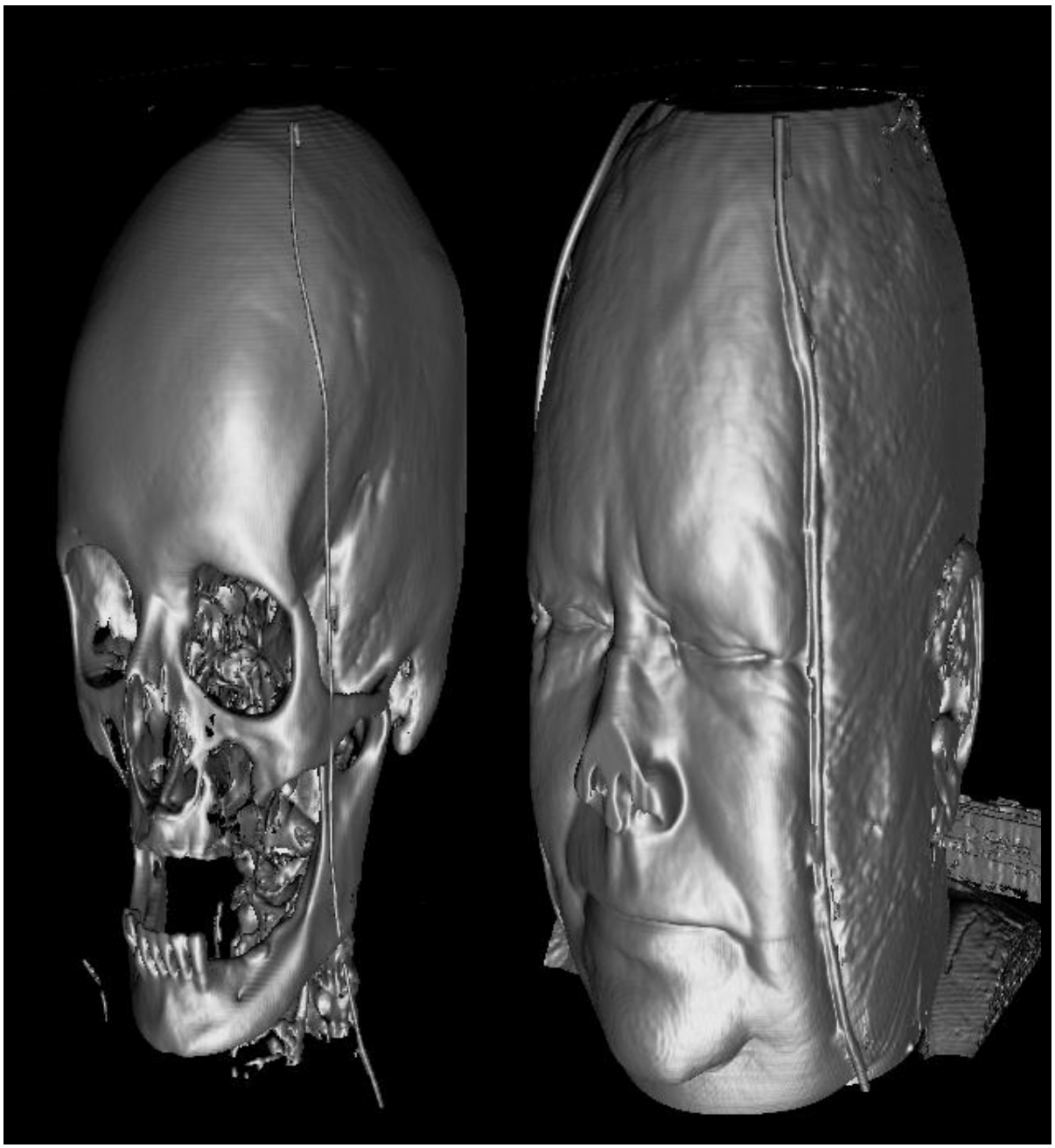




보기 6-81. 아이소표면 레이 함수 응용 예시

문지방

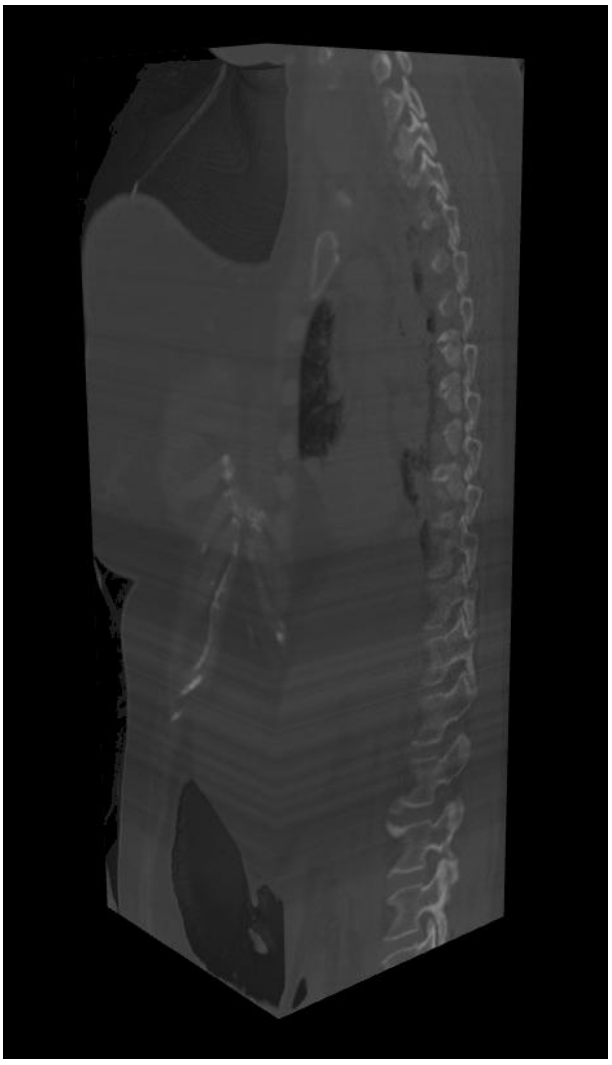
볼륨 렌더링 엔진은 **VolumeModel** 의 속성을 통해 문지방 범위를 적용할 수 있다. 각 색 채널을 위한 각자의 경계가 있다. 해당 색 값이 모든 채널에서 최고 및 최저 공계 내에 있을 때만 복슬이 시각화 된다. 괜찮은 구역들은 보이지 않는다. 이 속성은 마우스 힛 테스트에 포함이 안된다.



보기 6-82. 두개의 다른 문지방 설정 예시

슬라이스 범위

SliceRange 속성은 **VolumeModel** 의 일부를 잘라 내는 것을 허용한다. 이는 객체의 내부 구조를 탐색하는 데에 매우 유용한 도구이다. **SliceRange** 는 두 경계가 있다: **Min** 과 **Max.** 둘다 세 포인트 플로트 값으로 표현 된다.

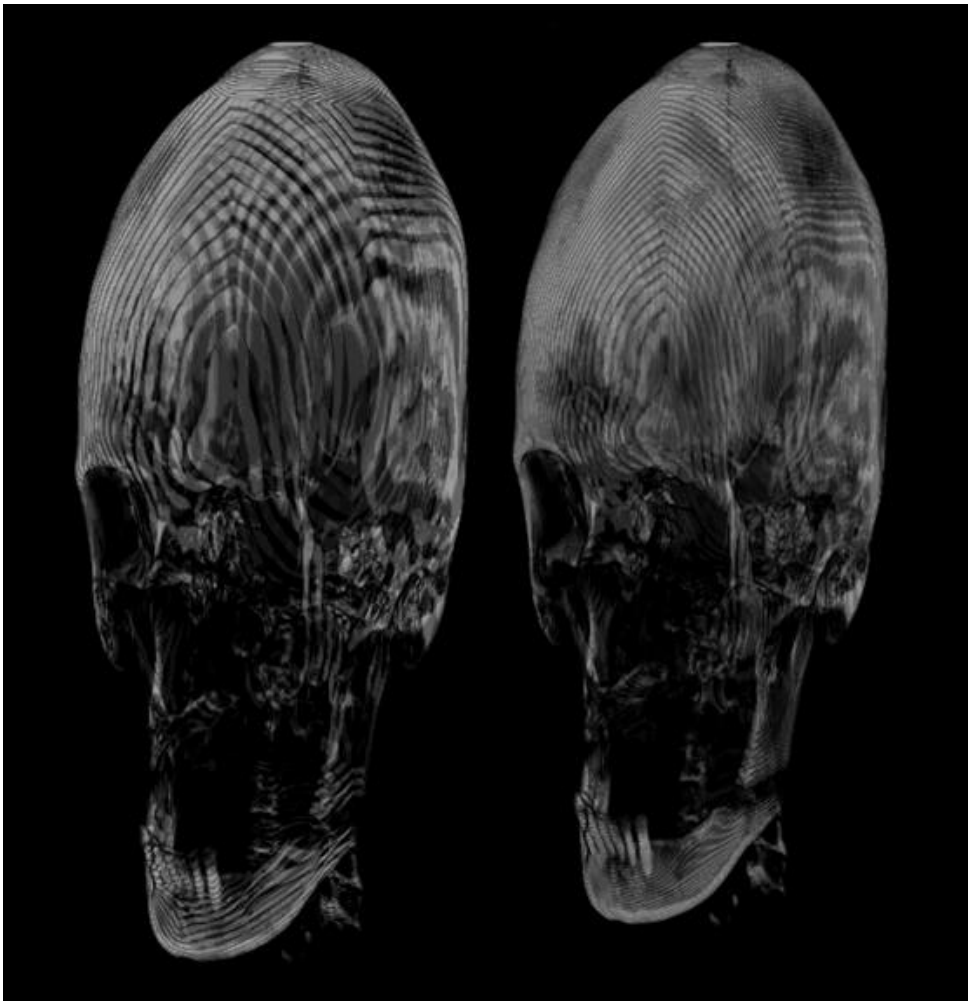


보기 6-83. 축적 레이 함수 및 SliceRange 변형의 예시

레이트 설정 샘플링

SamplingRate 은 마지막 이미지 품질에 매우 중요한 속성이다. 이는 볼륨 데이터 세트가 레이의 경로 도중 얼마나 자주 샘플링 되는지를 정의한다. **SamplingRate** 가 높을수로 질이 좋아지지만 하드웨어 성능이 더욱 좋아야 한다. **SamplingRate 은 RayFunction** 설정들에 영향을 미친다. 특히 **Accumulation** 에 영향을 미친다. 저 샘플링 레이트로 생성된 아티팩트는 **Maximal Intensity** 를 사용할 때 티가 덜 난다. **Isosurface** 가 높은 샘플링 레이트에는 너무 날카로울 수 있다. 보통 적당한 레이트는 레이 트랙에 올려진 쪽의 복슬 수와 같을 때다.

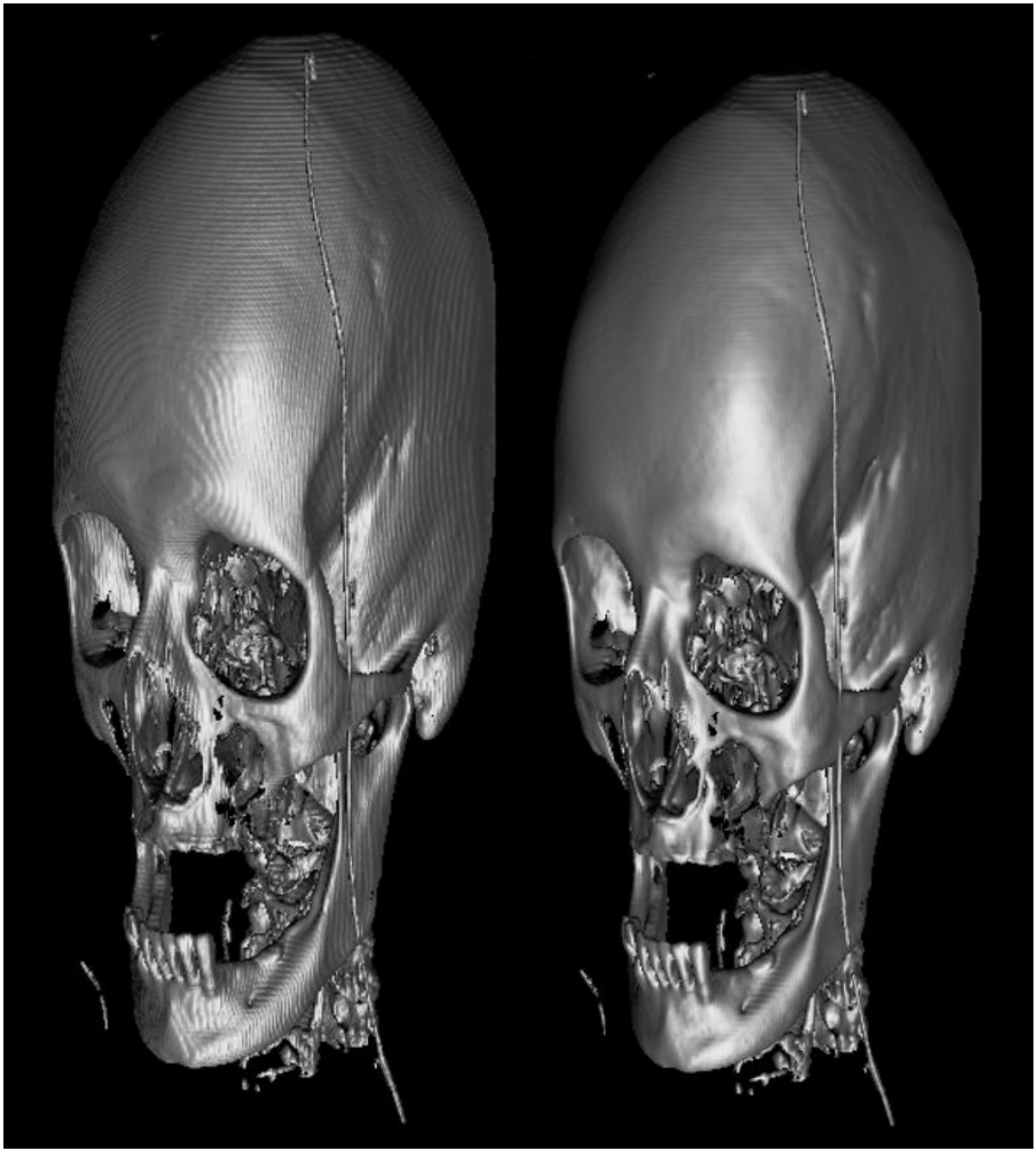
SamplingRateOptions 은 **SamplingRateManager** 를 위한 여러 옵션을 갖고 있다. **SamplingRateManager** 은 사용 하드웨어를 위한 품질 및 프레임율 사이 최적 밸런스를 찾기 위해 필요하다. 기본적으로 **Enabled** 가 참으로 설정 되어 **SamplingRateManager** 가 활성화 되어있다. 거짓으로 설정 되면 **ManualSamplingRate** 값이 사용 된다. **SamplingRateRange** 은 **SamplingRateManager** 의 경계를 정의한다. **Inertness** 는 성능 변화의 경우 대비 샘플링 율의 반응 속도가 얼마나 빠른지를 말해준다. **TargetFPS** 는 샘플링 율 관리자가 도달하려는 목표 값이다.



보기 6- 84. 저샘플링율의 예시: 32(좌), 64(우)

부드러움

Smoothness 속성은 표면의 너무 높은 디테일화를 방지한다. 이는 모델의 표면을 부드럽게 만들고 기타 아티팩트 및 노이즈를 줄인다.



보기 6-85. 너무 높은 샘플링율의 예시. 부드러움 속성으로 고쳐졌다.

EmptySpaceSkipping

EmptySpaceSkipping 속성은 빈 공간의 화질을 정의한다, 샘플링은 건너띄며. 16-32 의 낮은 값은 성능을 향상하지만 모델 가장자리에 아티팩트를 일으킬 수 있다.

보기 6- 86. 너무 낮은 EmptySpaceSkipping 속성 값의 예

불투명도

Opacity 는 **RayFunction** 의 **Accumulation** 옵션의 행동을 설명한다. **Opacity** 가 낮을수로 객체는 더욱 투명해 진다.

보기 6- 87. Accumulation Ray Function Opacity 변형 효과 예시: 15% (좌), 45(우)

밝기 및 어두움

이 속성들은 이미지의 전송 기능을 정의한다. 각 변화가 개인의 전송 기능을 갖고 있다. 이는 선형 함수로 표현 가능하다: 출력 = 밝기 * 입력 – 어두움

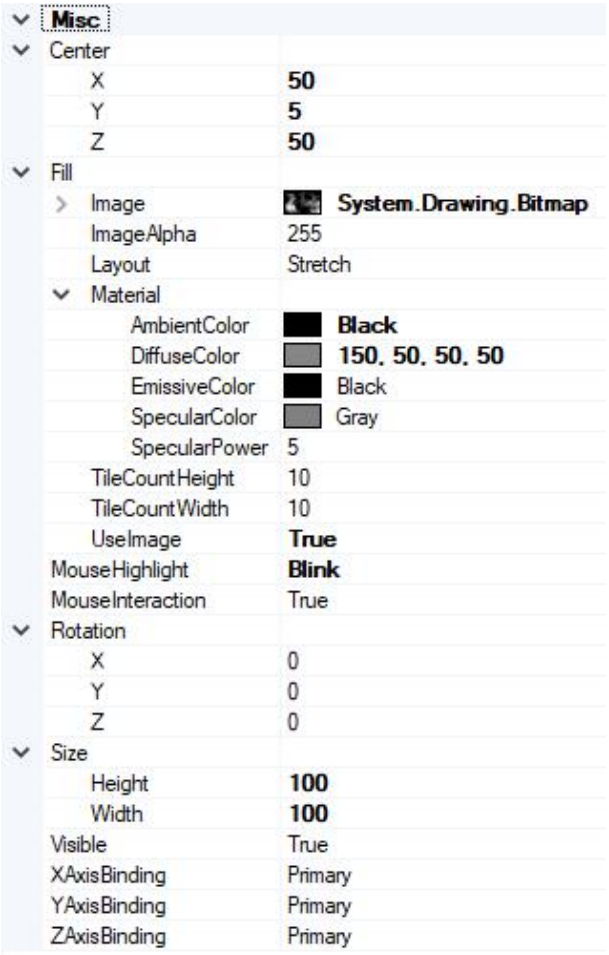
6.16 Rectangle3D objects

데모 예시: 직사각형/평면; 표면 마우스 제어; 평행 좌표 표

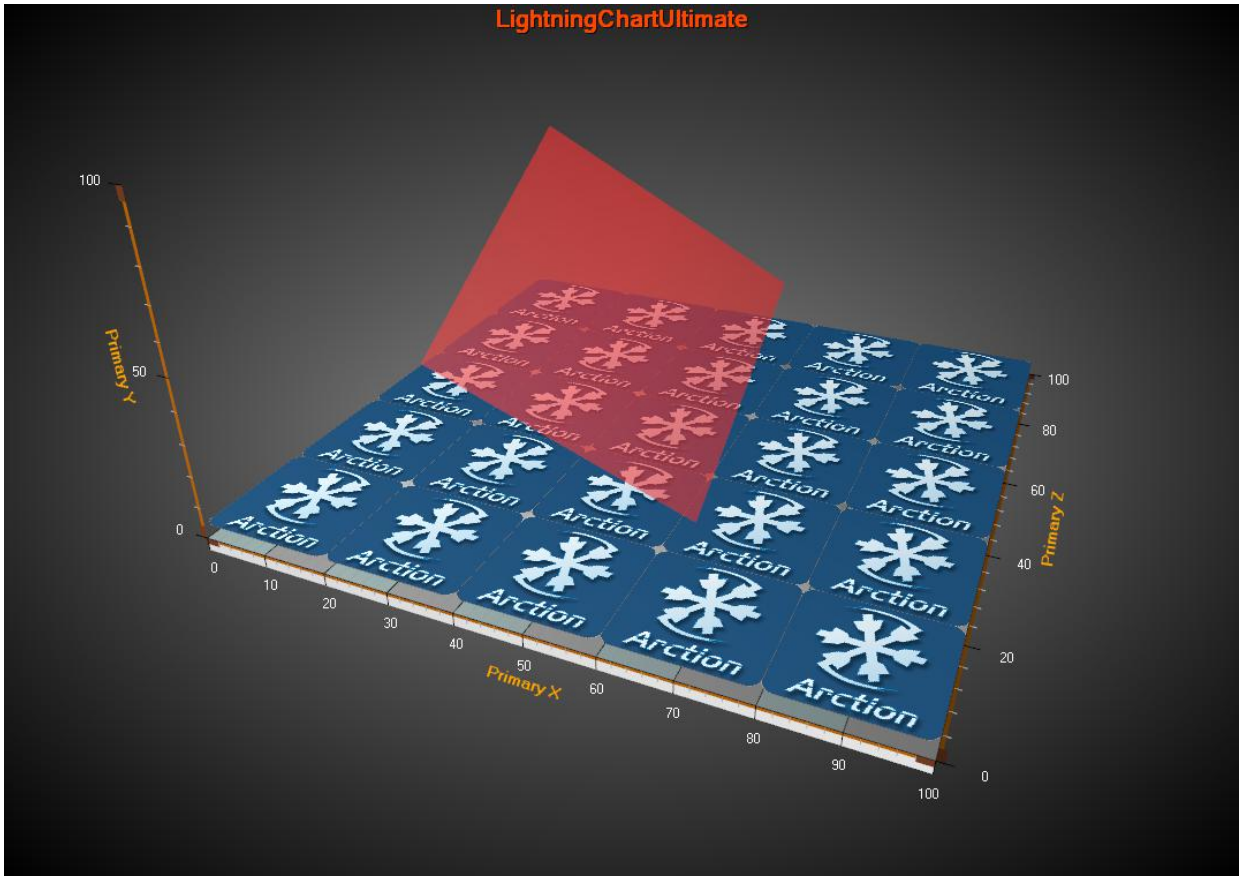
Rectangle3D 은 아무 각도로 회전된 어떤 크기의 어느 위치에든 직사각형을 표현 하는 것을 허용한다. 이는 **View3D.Rectangles** 목록에 추가 가능하다. 직사각형들은 **View3D.Dimensions** 대로 크기를 정의해평면으로도 사용 가능하다.

Size 를 **Width** 및 **Height** 로 3D 세계 차원에 설정하라 (X,Y,Z 축 값이 아닌). 중심점을 X,Y,Z 축 값으로 정의 된 **Center** 속성을 이용해 설정하라. **Rotation** 속성은 회전의 각도를 보여준다.

채우기 설정은 **Fill** 속성을 통해 변경 가능하다. 단색 및 비트맵 채우기가 사용 가능하다. 비트맵 채우기를 사용하기 위해 **Image** 에 비트맵을 설정 후 **UseImage** 를 활성화 하라. **Fill.Layout = Stretch** 를 설정할 때 비트맵이 직사각형을 채우게끔 늘어난다. **Fill.Layout = Tile** 을 설정하면 같은 비트맵이 타일화 되어 직사각형을 채운다. 타일 수는 **Fill.TileCountWidth** 및 **Fill.TileCountHeight** 속성들을 통해 수정 가능하다.



보기 6-88. Rectangle3D 객체의 속성들.



보기 6- 89. View3D 에 두 Rectangle3D 객체. 아래 파란색은 Layout = Tile 을 이용한 비트맵 채우기. 위에 회전 된 빨간색 직사각형은 반투명 색으로 구성 되었다.

6.17 Polygon3D objects

데모 예시: 3D 다각형; 세계 인구

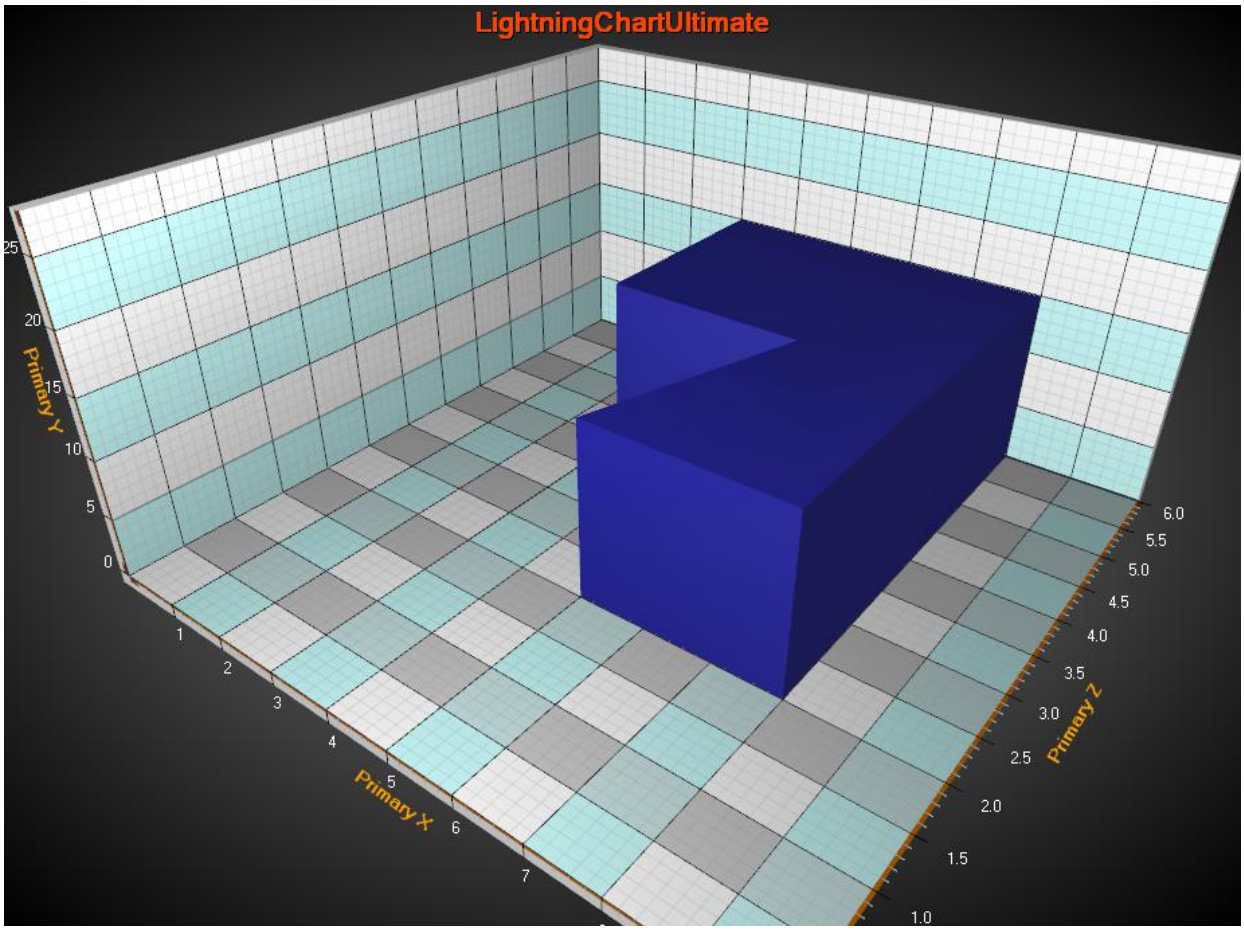
Polygon3D 객체들은 2D 다각형을 주어진 Y 범위 대로 늘어난 것을 표현한다. 이들은 **View3D.Polygons** 목록에 추가 가능하다.

X 와 Z 축 값으로 다각형 경로를 정의하고 **Points** 배열에 값을 저장하라. Y 범위를 **YMin** 및 **YMax** 값들로 설정하라.

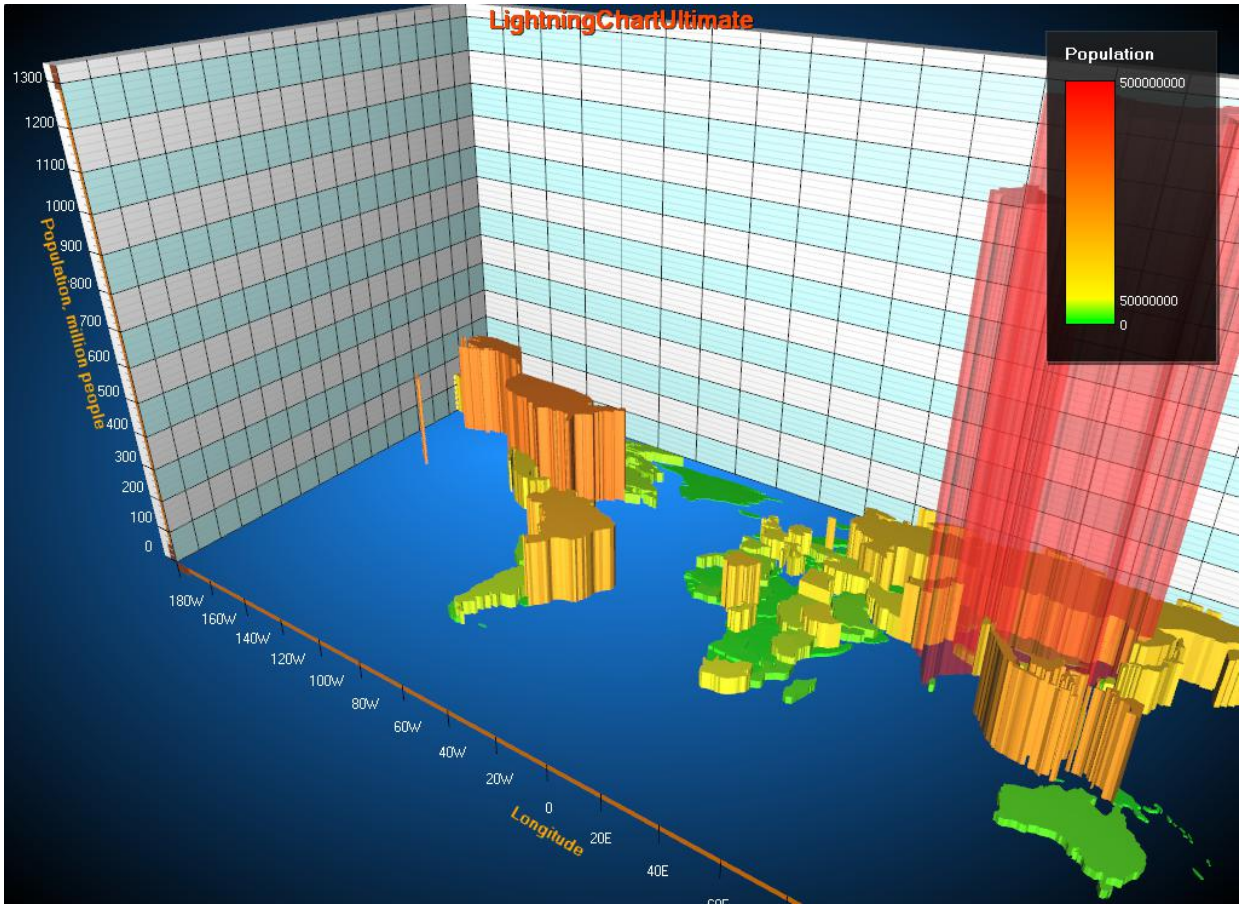
Material.Diffuse 는 직사각형의 주 색을 제어한다. **Rotation.X**, **Rotation.Y** 및 **Rotation.Z** 으로 다각형을 다른 각도로 회전하라.

Misc	
Material	
AmbientColor	Black
DiffuseColor	Magenta
EmissiveColor	Black
SpecularColor	Gray
SpecularPower	5
MouseHighlight	Simple
MouseInteraction	True
Points	Polygon3DPoint[] Array
Rotation	
X	0
Y	0
Z	0
Visible	True
XAxisBinding	Primary
YAxisBinding	Primary
YMax	20
YMin	0
ZAxisBinding	Primary

보기 6-90. Polygon3D 객체의 속성들.



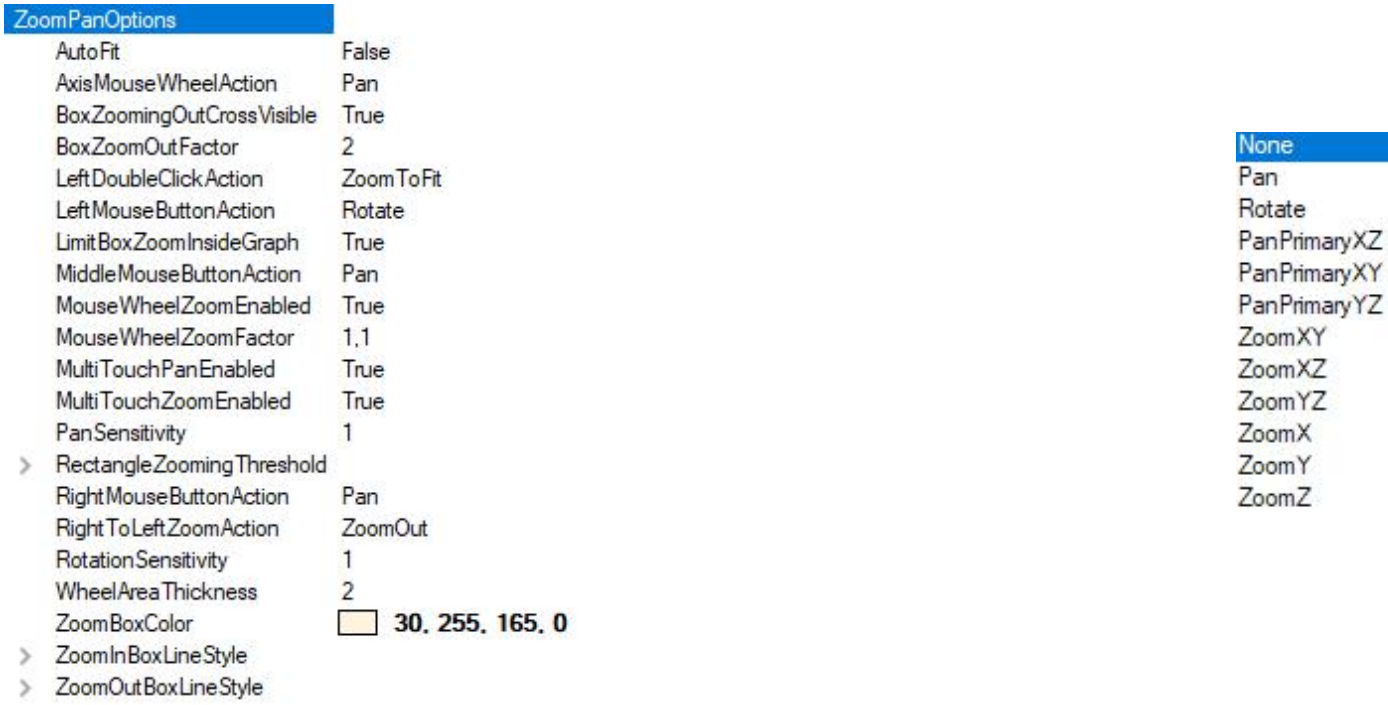
보기 6-91. 6-포인트 다각형. 범위는 YMin = 0, YMax = 15.



보기 6-92. Polygon3D 객체들로 표현한 세계 인구. Polygon3D 객체가 지도 데이터 각 구역에 그려졌다. 나라의 인구 정보를 다각형에 색 적용 및 YMax 를 설정하는 데에 사용되었다. 중국과 인도의 높은 인구 때문에 반투명 색이 사용되었다.

6.18 줌, 패닝, 회전

ZoomPanOptions 속성들은 줌, 패닝 및 회전 설정을 제어하기 위해 사용 가능하다.



보기 6-93. ZoomPanOptions 속성 및 하위 속성. 우측에 LeftMouseButton / MiddleMouseButtonAction / RightMouseButtonAction 옵션들이 있다.

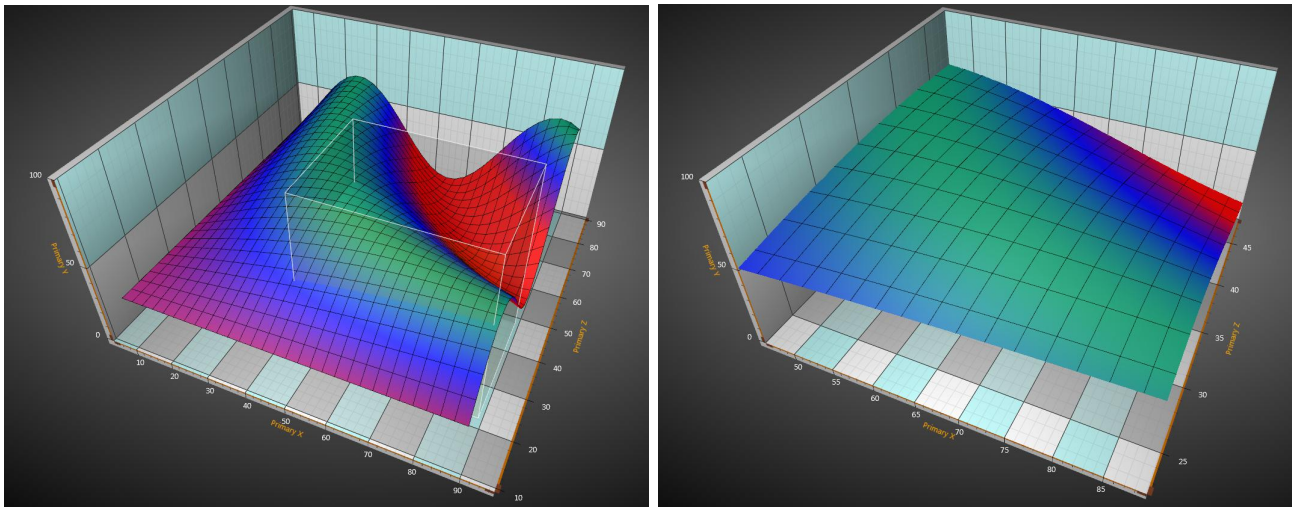
설정에 따라 줌은 마우스 휠, 터치 스크린 또는 선택 3D 평면에 상자를 그려 실행 할 수 있다. 패닝, 박스 줌 및 회전은 모두 좌, 중 또는 우 마우스 버튼으로 실행 구성 가능하다. 패닝은 3D 표 전체를 위해 만들어지거나 3D 장면 위치는 유지하면서 주 축들만 변경하게끔 설정 가능하다.

마우스 휠로 줌

마우스 휠을 위로 스크롤해 확대를 하고 아래로 축소 하라. **MouseWheelZoomFactor** 로 마우스 휠 이벤트 당 적용 되는 줌의 정도를 수정하라. 마우스 휠로 줌을 비활성화 하기 위해 **MouseWheelZoomEnabled** 를 거짓으로 설정하라. 기본적으로 이는 참으로 설정 되었다.

박스 줌

박스 줌을 활성화 하기 위해 마우스 버튼 실행 속성에 박스 줌을 할당 하라. 예를 들어 **LeftMouseButtonAction = ZoomXZ** 설정은 박스 줌이 XZ 평면에 적용되게 한다. Y 차원에는 효과가 없다. 다른 평면에 줌을 적용하기 위해 **ZoomXZ** 또는 **ZoomYZ** 설정하라.



보기 6-94. 왼쪽에는 XZ-평면 줌이 진행 중이다. 오른쪽에는 줌의 결과물이다. X 와 Z 축 값만 변형 되었고 Y 축 범위는 그대로 이다.

박스를 죄에서 우로 드래그해 확대하라. 줌 범위는 선택 평면 해당 축들에 적용된다.

X, Y, Z 중 특별 차원에만 줌을 하기 위해 **ZoomX**, **ZoomY** 또는 **ZoomZ** 을 선택하라.

박스를 우에서 좌로 드래그해 줌을 축소하라. 줌 아웃은 **BoxZoomOutFactor** 설정 팩터만큼만 적용 된다. 줌 아웃은 박스 앞에 십자 모양을 보여준다. 이는 **BoxZoomingOutCrossVisible = False** 을 설정해 비활성화 할 수 있다.

ZoomPadding

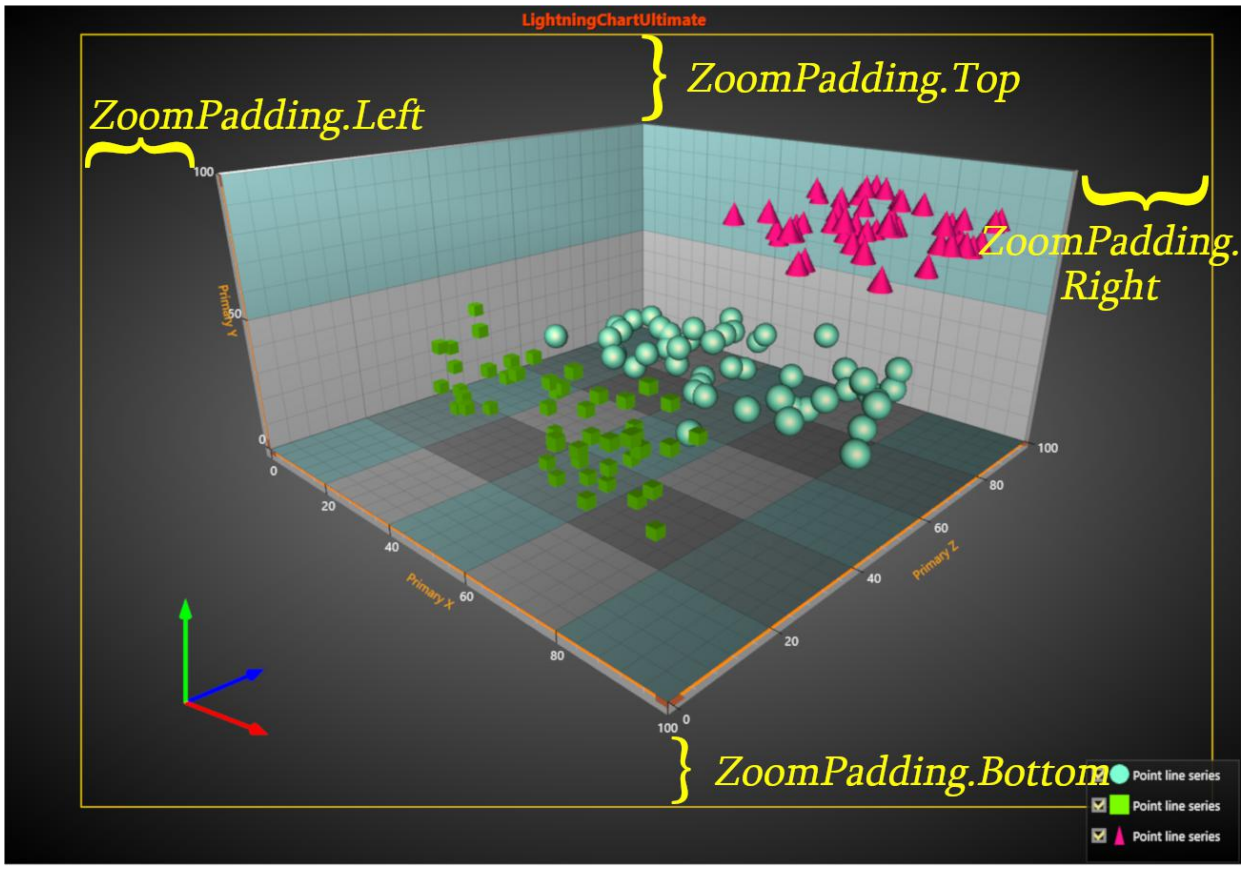
ZoomPadding 속성은 줌 연산 후 3D 모델과 마진 사이 남은 빈 공간 수를 정의한다. **ZoomPadding** 은 차트를 움직이거나 수동으로 줌할 때 아무 효과가 없다. 수동 줌은 마우스 스크롤로 줌 하는 것 등을 얘기한다. **ZoomPadding** 은 또한 직사각형 기반 줌에도 적용되지 않는다.

윈폼에서 **ZoomPadding** 설정하기:

```
chart.View3D.ZoomPanOptions.ZoomPadding = new Padding(10, 30, 10, 10);
```

Wpf 에서 **ZoomPadding** 설정:

```
chart.View3D.ZoomPanOptions.ZoomPadding = new Thickness(10, 30, 10, 10);
```

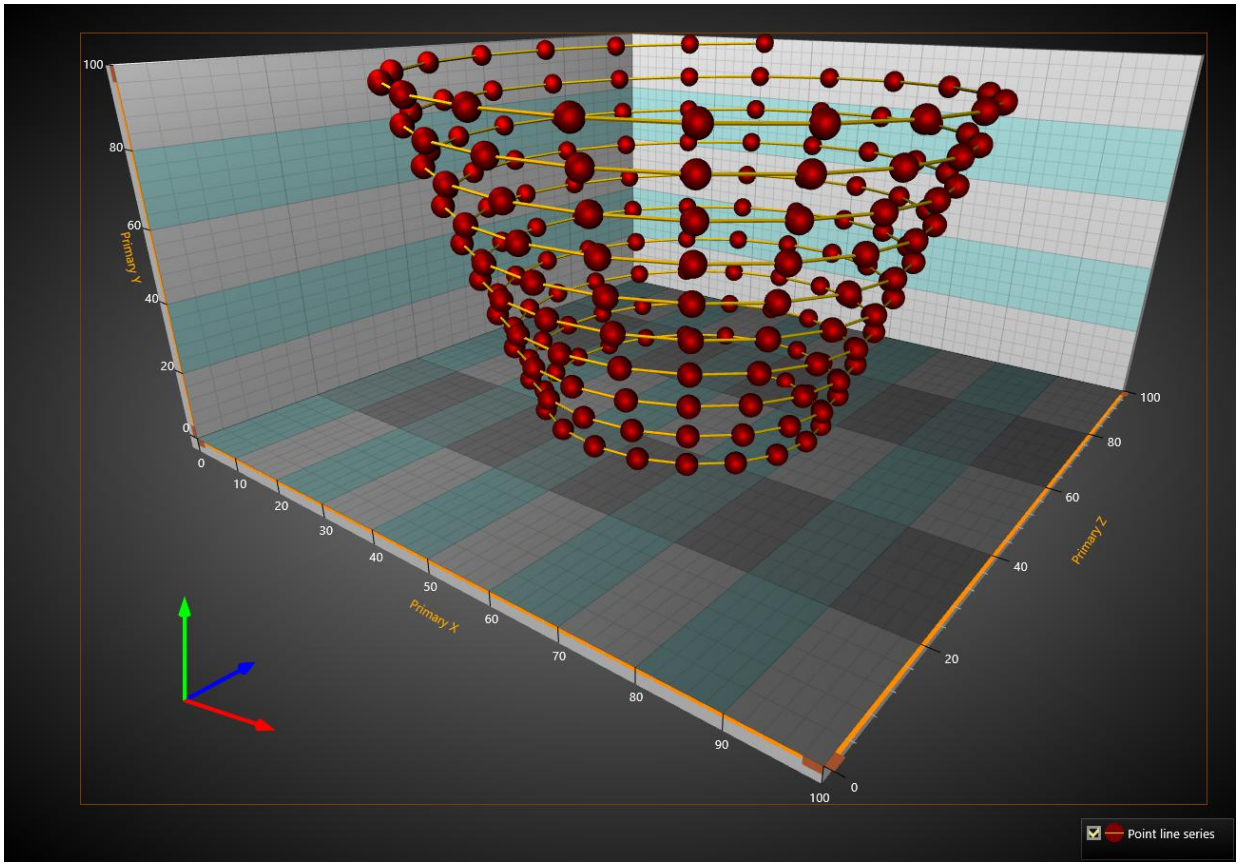


보기 6-95. ZoomPadding 은 데이터/라벨과 마진 사이 빈 공간을 남긴다. 예를 들어 이 경우와 같이 ZoomToDataAndLabels 연산이 사용 되었을 때.

ZoomToDataAndLabels

View3D 에는 **ZoomToDataAndLabels** 연산은 마진 및 **ZoomPadding** 으로 제한 된 사용 가능 구역을 카메라를 더욱 가까이/멀리 움직여 최대한 효율적이게 사용될 수 있게 한다. 축, 라벨, 시리즈 데이터 및 마커가 모두 보이게 설정 된다. 차트 제목, 주석 및 레전드 삭자의 위치는 화면 좌표로 정의 되었기에 모두 무시 된다. **ZoomToDataAndLabels** 은 뷰잉 각도를 유지하면서 뷰의 내용은 중심으로 정렬 된다.

기본 적으로 **LeftDoubleClickAction** 속성이 **ZoomToDataAndLabels** 로 설정 되었다. 이는 마우스 좌클릭 두번 이 연산을 실행한다. 이 속성을 **Off** 로 바꿔 비활성화 하라. 코드로 **View3D.ZoomToFit(ZoomArea3D.DataAndLabelsArea)** 메소드를 불러 **ZoomToDataAndLabels** 실행 가능하다.



보기 6-96. ZoomToDataAndLabels 연산이 실행됐다. 데이터 시리즈, 축, 벽 등이 최적으로 마진 내 맞춰졌다. 오리엔테이션 화살표가 그래프 구역의 왼쪽 아래 구석을 따르지만 렌전드 상자는 무시 되었다. 모든 가장자리는 ZoomPadding = 0 로 설정 되어 마진과 DataAndLabelsArea 사이 빈 공간이 남지 않는다.

ZoomToDataAndLabels 는 카메라의 **MinimumViewDistance** 속성을 고려한다. 어떤 경우에는 카메라가 차트에 충분히 가까이 갈 수가 없어 그래프 구역의 전체가 사용 되지 않을 수도 있다는 것이다. 이런 경우에는 **ChartMessage** 알림이 보내진다.

회전 및 패닝

할당된 마우스 버튼을 눌러 수평 또는 수직으로 드래그 함으로 3D 모델 주변 카메라를 돌릴 수 있다. **RotationX**, **RotationY** 및 **RotationZ** 속성들이 업데이트 된다.

마우스 버튼 액션이 **Pan** 으로 설정 되었을 때 패닝은 카메라의 **Target** 속성을 업데이트 한다. 마우스 버튼 액션이 **PanPrimaryXZ**, **PanPrimaryXY** 또는 **PanPrimaryYZ** 으로 설정 되었을 때 주 X,Y,Z 축 범위가 수정된다. 예를 들어 PanPrimaryXZ 는 마우스로 드래그 할 때 X 와 Z 축들을 수정한다. 보조 X,Y,Z 축들은 변하지 않는다.

LeftMouseButtonAction / **MiddleMouseButtonAction** / **RightMouseButtonAction** 를 **Pan/PanPrimaryXZ/PanPrimaryXY/PanPrimaryZ** 로 설정해 패닝을 활성화 하라. **Rotate** 로 설정해 회전을 활성화 하라. 왼쪽 마우스 버튼으로 패닝 및 회전을 비활성화 하기 위해서는 **None** 으로 설정하라.

PanSensitivity 로 적용 패닝 정도를 제어하라. 비슷하게 **RotationSensitivity** 로 적용 회전율 정도를 제어하라.

터치스크린으로 줌

차트에 두 손가락 올려 모아서 확대 펴서 축소를 하라. 터치스크린으로 줌을 비활성화 하기 위해 **MultiTouchZoomEnabled** 를 거짓으로 설정하라.

터치 스크린으로 패닝

차트에 두 손가락을 올려 같은 방향으로 움직여 패닝을 적용 하라. 이 기능을 비활성화 하기 위해 **MultiTouchPanEnabled** 를 거짓으로 설정하라.

축 위 마우스휠 사용

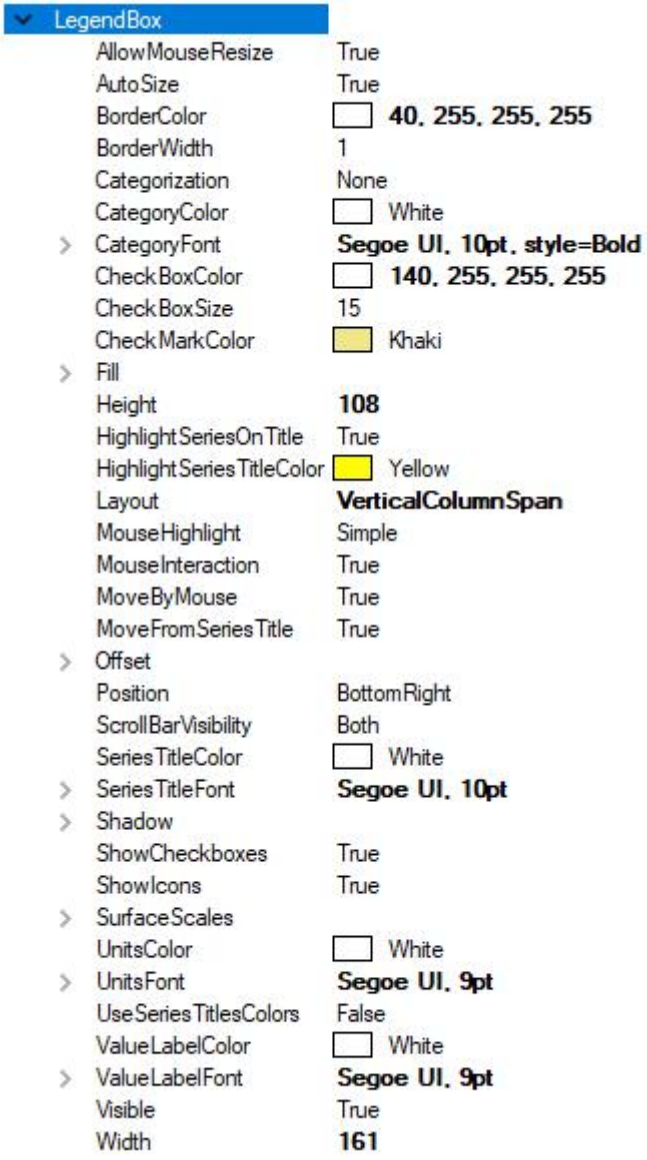
축 위로 마우스 휠을 스크롤 하면 차트가 해당 축 특병 줌 또는 패닝을 만든다. **WheelAreaThickness** 는 축 근처 마우스 휠 반응 구역 넓이를 수정한다. **AxisMouseWheelAction** 로 줌과 패닝 기능 사이 선택할 수 있다.

코드로 줌, 회전, 패닝

3D 뷰는 **View3D.Camera** 를 **RotationX**, **RotationY** 및 **RotationZ** 속성들로 움직여 회전 된다. **Perspective** 및 **Orthographic** 카메라로는 **ViewDistance** 설정으로 줌할 수 있다. **OrthographicLegacy** 카메라로는 **Dimensions** 을 바꿔 줌을 한다. 패닝은 카메라 **Target** 을 3D 모델 좌표로 설정하여 이룰 수 있다.

6.19 레전드 상자

View3D 에 레전드 상자는 ViewXY 의 레전드 상자와 많이 비슷하다 (6.21 을 보아라). 하지만 그래프 당 한 레전드 상자만 사용 가능하다. 또한 세그먼트 기반 속성들은 존재하지 않는다. 이는 View3D 의 축들은 세그먼트로 나눌 수 없기 때문이다. **View3D.LegendBox** 로 레전드 상자 속성을 변경하라.



보기 6-97. View3D 에서 레전드 상자 속성

표면 시리즈 팔레뜨 스케일 숨기기

View3D 는 ViewXY 의 **IntensityScales** 대신 **SurfaceScales** 속성을 갖고 있다. 레전드 상자에 팔레뜨 스케일을 숨기기 위해 **SurfaceScales.Visible = False** 로 설정하라. 크기를 재설정 하기 위해 **ScaleSizeDim1** 및 **ScaleSizeDim2** 속성들을 설정하라.

View3D 에서 레전드 상자 위치 선정

ViewXY 와 같이 View3D 의 레전드 상자는 자동 또는 수동 위치 선정 가능하다. 자동 위치 선정은 뷰 또는 그래프의 좌/상/우/하 에 정렬을 허용한다. **Position** 속성으로 위치를 변경하라. 어떤 위치 선청 옵션은 마진을 고려하고 어떤 옵션들은 하지 않는다.

마진을 무시하는 옵션들 (마진 구역에 레전드 상자 놓기 가능):

TopCenter, **TopLeft**, **TopRight**, **LeftCenter**, **RightCenter**, **BottomLeft**, **BottomCenter**, **BottomRight**, **Manual**

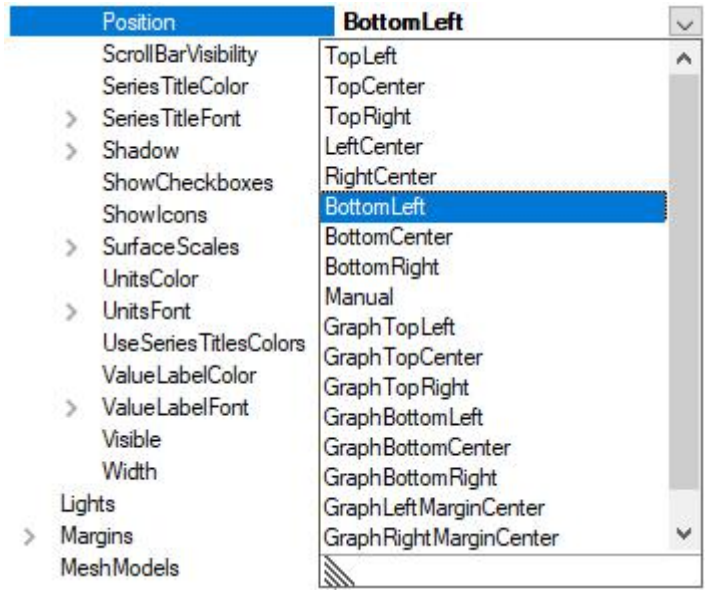
레전드 상자를 마진 구역 내 놓는 옵션들:

GraphTopCenter, **GraphTopLeft**, **GraphTopRight**, **GraphLeftCenter**, **GraphRightCenter**, **GraphBottomLeft**, **GraphBottomCenter**, **GraphBottomRight**

Offset 속성은 주어진 값 만큼 위치에서 **Position** 속성 만큼 멀리 움직인다.

```
// Setting legend box position, offset shifts from RightCenter position
chart.View3D.LegendBox.Position = LegendBoxPosition.RightCenter;
chart.View3D.LegendBox.Offset = new PointIntXY(-15, -70);
```

Manual 위치 선정은 레전드 상자의 왼쪽 위 구석에서 뷰의 왼쪽 위 구석의 오프셋을 계산한다. 이는 그래프 구역 위에서 부터 계산되는 **TopLeft** 옵션과는 다르다.

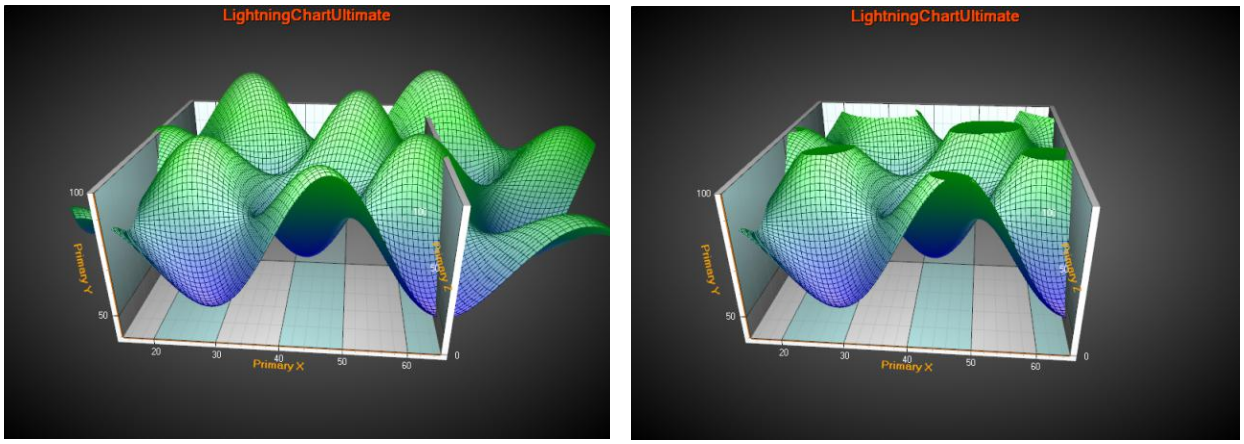


보기 6-98. 레전드 상자를 위한 위치 옵션. 그래프.. 옵션들은 레전드 상자를 마진 안에다가 놓는다.

6.20 축 범위 내 객체 클리핑

데모 예시: 간단 3D 표면 그리드

ClipContents 속성을 참으로 설정함으로 시리즈, 직사각형과 메쉬 모델은 축 값 범위 내로 잘린다. 축은 차원에는 항상 늘려져서 클리핑이 활성화 되었을 때 벽 밖에서 렌더링을 방지한다.



보기 6-99. 좌, ClipContents 사용 안함. 축 범위 외 시리즈 렌더링. 우, ClipContents 활성화 됨.

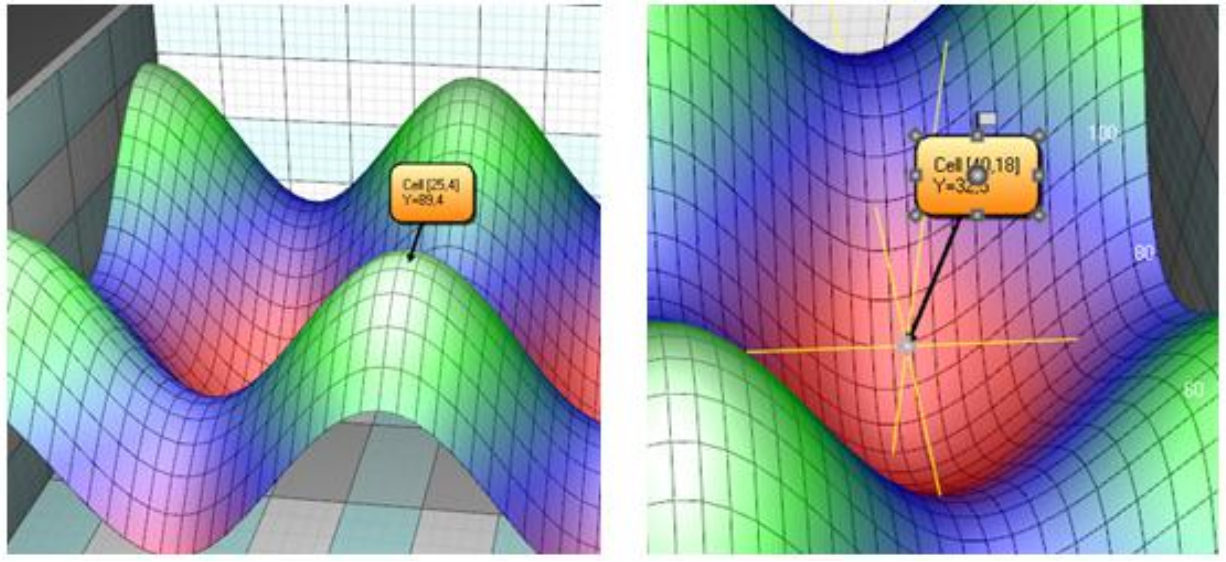
클리핑은 시리즈 데이터 자체를 변경하지 않는다. 클리핑은 렌더링 단계에만 일어난다. 또한 벽 밖의 보이지 않는 잘린 객체들에는 마우스 히트 테스트가 적용된다.

클리핑이 활성화 되었을 때 차트 내 모든 라인들이 자동으로 선 넓이 1 로 설정 된다.

6.21 Annotation3D

데모 예시: 포인트 트래킹; 표면 마우스 제어; 메쉬 모델 색칠, 와이어프레임

Annotation3D 컬렉션은 3D 장면에 주석을 추가할 수 있게 한다. 기본적으로 이는 ViewXY 의 **Annotations** 와 비슷하다 (6.20 을 보아라). X,Y,Z 차원들을 사용하는 **Target** 및 **Location** 속성들만이 다르다.



보기 6-100. 3D 시리즈의 값을 표시하는 Annotation3D 객체. 십자선 커서를 이용해 목표 이동을 도울 수 있다.

Target 은 3D 에서 마우스로 움직일 수 있다. 움직임을 돕기 위해 마우스가 **Target** 노드 위에 올려졌을 때 주석은 십자선을 보여준다. **ShowTargetCrosshair** 속성을 **Auto/On/Off** 로 설정하고 **TargetCrosshairLineStyle** 에서 라인 스타일을 수정하라.

7. 좌표 시스템 컨버터

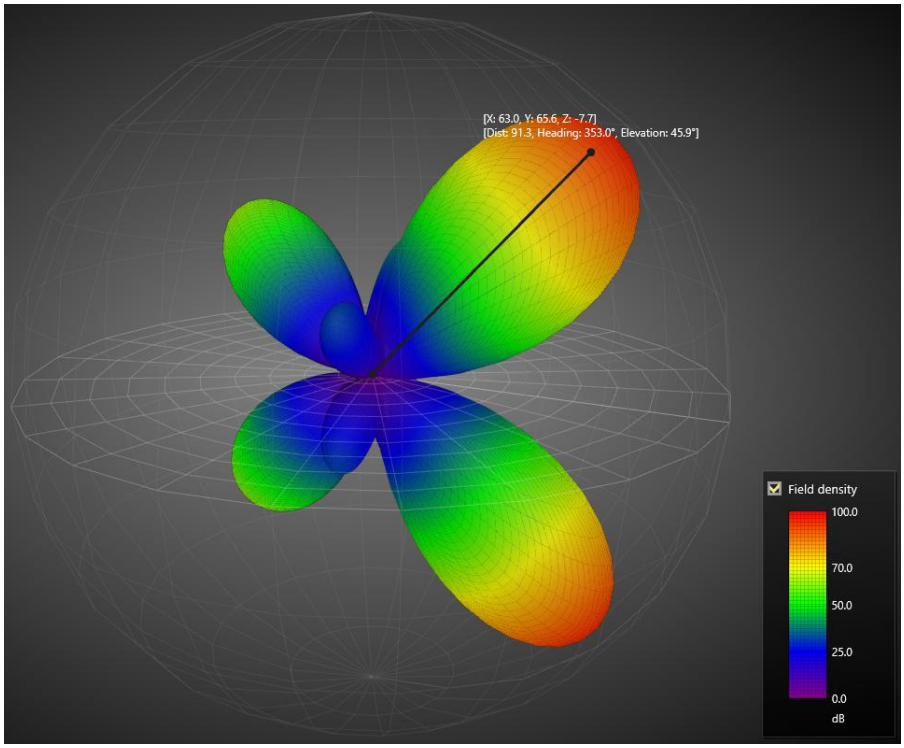
다음 좌표 시스템 컨버터는 **CoordinateConverters** 네임스페이스에서 사용 가능하다. 이는 View3D 와 잘 적용된다.

- Cartesian 3D <-> Spherical 3D
- Cartesian 3D <-> Cylindrical 3D

7.1 SphericalCartesian3D

데모 예시: 원형 좌표

SphericalCartesian3D 컨버터 클래스는 원형 및 3D 카르테시안 좌표 사이 변경한다.



보기 8- 1. SphericalCartesian3D 컨버터로 생성된 예시. SurfaceMeshSeries3D 의 데이터 포인트 및 그리드가 원형 좌표로 정의 되었다. 주석이 제일 가까운 데이터 포인트를 트래킹하고 값을 원형 좌표로 표시한다.

원형 데이터 포인트들은 **SphericalPoint** 객체들로 정의 된다. 이들은 다음과 같은 필드들을 갖고 있다:

- **Distance**: 원점에서 부터의 거리 (0,0,0)
- **ElevationAngle**: 높이 각도. 엘레베이션 또는 고도로도 불림. XZ 평면에서 부터 계산됨. ElevationAngle 은 90 도다 – 기울기 각도.
- **HeadingAngle**: 헤딩 각도. 방위각 및 절대 베어링으로도 불림

높이에는 XZ 평면이 참조 면이다. 높이는 그 평면에서 재진 각도이다.

주의! 이 컨버터 클라스는 View3D.Dimensions 가 일치하기를 예상한다. 아닐 시 컨버션 결과가 사용자 쪽 코드로 스케일링 필요할 수 있다.

View3D 의 시리즈는 주로 데이터 인풋을 X,Y,Z 값으로 받는다. 이 값들은 **SeriesPoint3D**, **SurfacePoint3D** 및 **PointDouble3D** 객체 등에서 찾을 수 있다.

원형에서 카르테시안으로 변경하기

SphericalPoint 를 카르테시안 좌표로 변경하기 위해 **SphericalCartesian3D.ToCartesian()** 메소드를 사용하라. 이는 데이터 인풋을 다음과 같이 받는다:

- SphericalPoint 포인트
- SphericalPoint[] 배열
- SphericalPoint[,] 메트릭스

또는 원형 포인트는 **ToCartesian()** 확장 메소드를 이요해 변경하라.

```
// Create spherical points matrix
SphericalPoint[,] sphericalData = CreateSurfaceData();

// Convert matrix to cartesian matrix
SurfacePoint[,] xyzData = sphericalData.ToCartesian();
```

바인딩 WPF 차트에서 매트릭스를 카르테시안으로 변경하기:

```
SurfacePointMatrix xyzData = sphericalData.ToCartesian();
```

카르테시안에서 원형으로 변경

카르테시안 포인트를 원형 포인트로 변경하기 위해 **SphericalCartesian3D.ToSpherical()** 메소드를 사용하라. 이는 데이터 인풋을 **PointDouble3D** 포인트로 X,Y,Z 필드에 받는다.

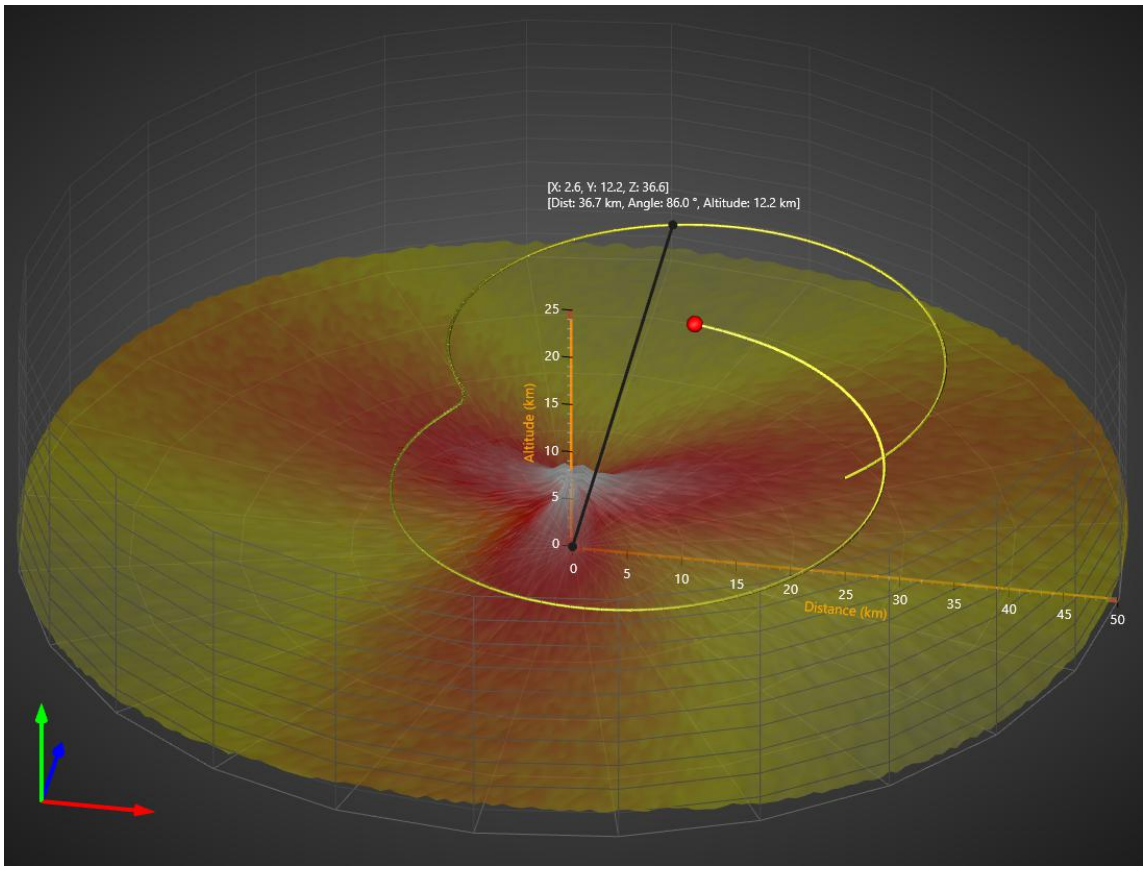
또한, 포인트를 **ToSpherical()** 확장 메소드를 이용해 변경하라.

```
// Define cartesian point
PointDouble3D point = new PointDouble3D(50, 20, 40);

// Convert to spherical point
SphericalPoint sp = point.ToSpherical();
```

7.2 CylindricalCartesian3D

데모 예시: 원기둥 좌표

컨버터 클라스로 원기둥 및 3D 카르테시안 좌표 사이 변경하라



보기 8- 2. CylindricalCartesian3D 컨버터로 생성한 예시. SurfaceMeshSeries3D 및 그리드 데이터 포인트는 원기둥형 좌표로 정의 되었다. 주석이 PointLineSeries3d 의 가장 가까운 데이터 포인트를 트래킹해 값을 원기둥형 좌표로 표시한다.

원기둥형 포인트는 **CylindicalPoint** 객체들로 정의되었다. 다음과 같은 필드를 갖고 있다:

- **Distance**: XZ 평면 위 거리
- **Y**: Y 값
- **Angle**: 헤딩 각도. 방위각 및 절대 베어링으로도 불림

주의! 주의! 이 컨버터 클라스는 View3D.Dimensions 가 일치하기를 예상한다. 아닐 시 컨버션 결과가 사용자 쪽 코드로 스케일링 필요할 수 있다.

View3D 의 시리즈는 주로 데이터 인풋을 X,Y,Z 값으로 받는다. 이 값들은 **SeriesPoint3D**, **SurfacePoint3D** 및 **PointDouble3D** 객체 등에서 찾을 수 있다.

원기둥형에서 카르테시안으로 변경

CylindricalPoint 을 카르테시안 좌표로 변경하기 위해 **CylindricalCartesian3D.ToCartesian()** 메소드를 사용하라. 이는 인풋 데이터를 다음과 같이 받는다:

- CylindricalPoint 포인트
- CylindricalPoint[] 배열
- CylindricalPoint[,] 매트릭스

또한 원기둥형 포인트는 **ToCartesian()** 확장 메소드를 이용해 변경하라.

```
// Create spherical points matrix
CylindricalPoint[,] cylindricalData = CreateData();

// Convert matrix to cartesian matrix
SurfacePoint[,] xyzData = cylindricalData.ToCartesian();
```

바인딩 WPF 차트에서 매트릭스를 카르테시안으로 변경하기:

```
SurfacePointMatrix xyzData = cylindricalData.ToCartesian();
```

카르테시안에서 원기둥형으로 변경

카르테시안 포인트를 원기둥형 포인트로 변경하기 위해 **CylindricalCartesian3D.ToCylindrical()** 메소드를 사용하라. 이는 데이터 인풋을 **PointDouble3D** 포인트의 X,Y,Z 필드로 받는다.

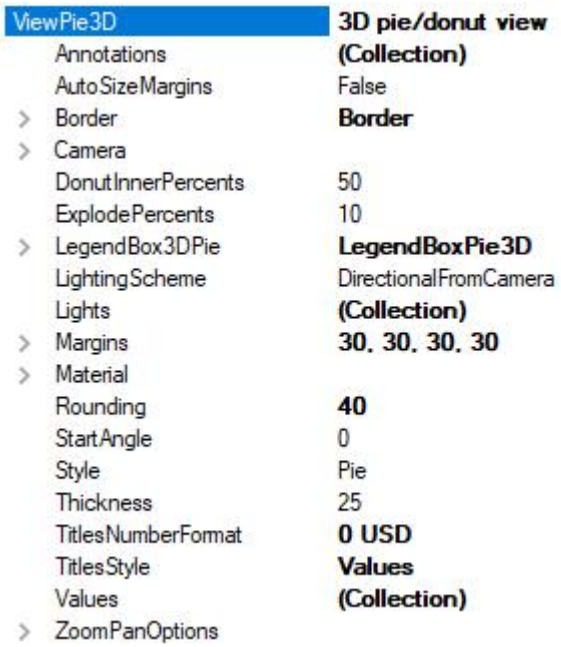
또는 포인트를 **ToCylindrical()** 확장 메소드를 이용해 변경하라.

```
// Define cartesian point
PointDouble3D point = new PointDouble3D(50, 20, 40);

// Convert to spherical point
CylindricalPoint sp = point.ToCylindrical();
```

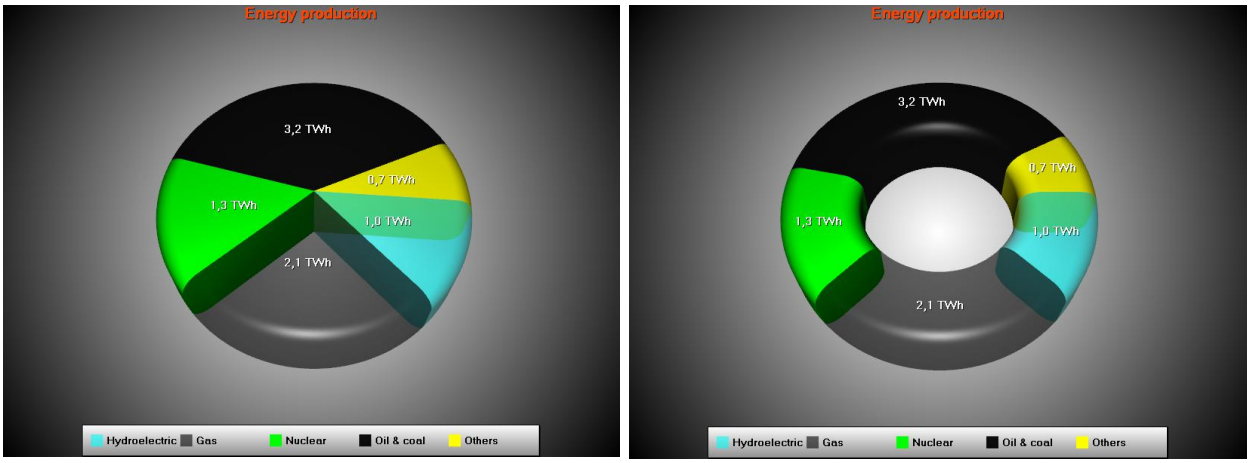
8. ViewPie3D

데모 예시: 파이 2D ; 파이 2D ; 도넛 3D

ViewPie3D 는 3D 로 데이터를 파이 및 도넛 차트로 표현한다.



보기 9- 1. ViewPie3D 객체 트리.



보기 9- 2. 3D 파이 차트 및 도넛 차트 예시

8.1 속성

차트 종류를 **Style** 속성 이용해 **Pie** 또는 **Donut** 사이 선택하라. **ZoomPanOptions** 속성 트리로 줌, 패닝, 회전을 제어하라. 이는 View3D 와 비슷하다 (7.18 을 보아라).

Camera 속성은 뷰포인트를 제어한다 (7.4 를 보아라). 사전 정의된 빛 설정을 **LightingScheme** 속성으로 선택 가능하다. **Material** 속성 및 하위 속성을 이용해 기본 3D 표면 외관 및 반짝임을 수정하라.

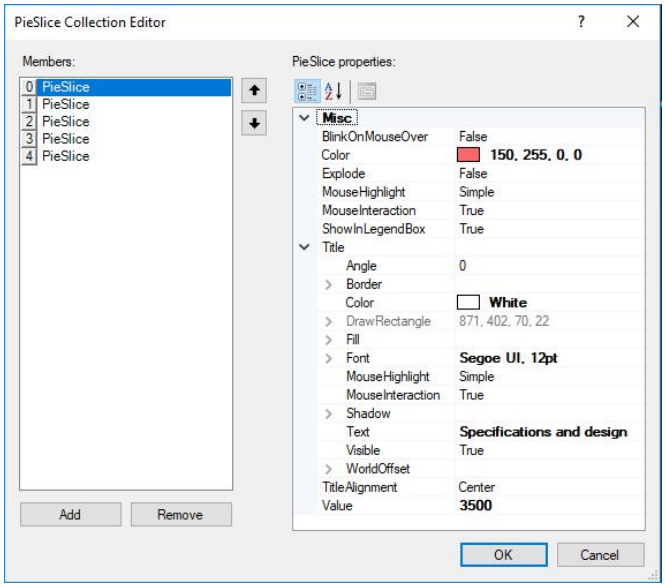
DonutInnerPercents 를 이용해 도넛의 안쪽 반지름을 설정하라. **Rounding** 으로 가장자리 라운딩 반지름 수정, **StartAngle** 로 파이를 회전, **Thickness** 로 파이 두께를 수정하라. **ExplodePercents** 는 조각의 **Explode** 값이 참으로 설정 되었을 때 폭발 파이 조각이 얼마나 멀리 보이는지를 수정한다.

TitlesStyle 는 파이 조각 텍스트를 다음 중 하나로 설정한다 : **Titles**, **Values** 또는 **Percents.** **TitlesNumberFormat** 를 "0.0 TWh"등으로 수정하여 뒤에 유닛을 포함하라.

Annotations 은 View3D 와 같이 사용할 수 있지만 축 값 바인딩 속성이 없다 (7.21 을 보라).

8.2 파이 조각

파이 차트 데이터는 **Values** 컬렉션에 저장 된다. 목록의 있는 각 아이템은 **PieSlice** 종류이다. 데이터 값을 **Value** 속성으로 수정하라. 제목 스트링을 **Title.Text** 속성으로 설정하라. **TitleAlignment** = **Outside** 로 정의함은 제목을 파이 밖에 그린다.

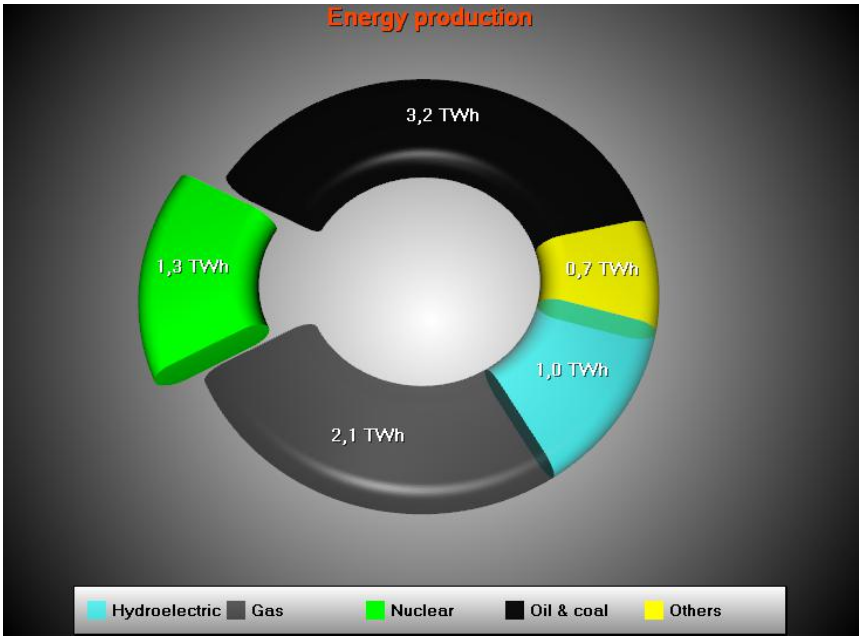


보기 9- 3. 파이 차트의 Values 목록 에디터

8.3 코드로 데이터 설정

데이터는 **PieSlices** 로 **Values** 목록에 저장 된다.

```
//Add pie slice data
//By using 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;
```

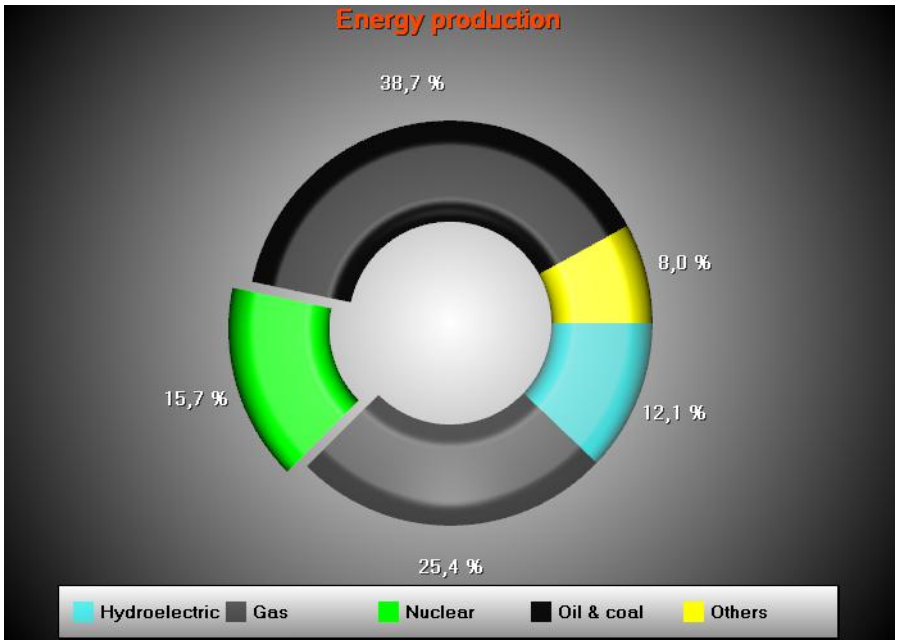


보기 9- 4. 차트로 데이터 설정. 세번째 조각은 slice3.Explode = true 로 따로 떼어 놨다.

8.4 2D 로 파이 차크 보기

카메라를 사전 정의 된 위에서 보는 카메라로 설정하라.

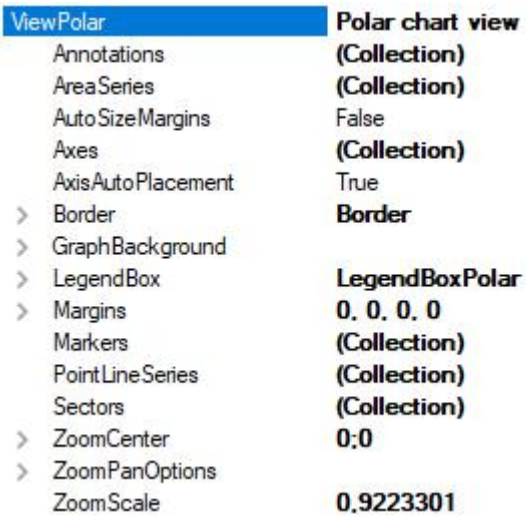
```
chart.ViewPie3D.Camera.SetPredefinedCamera(PredefinedCamera.PieTop);
```



보기 9- 5. 2D 로 보여지는 파이차트. 사전 정의 된 위에서 내려다 보는 카메라가 사용됐다.

9. ViewPolar

ViewPolar 는 극성 형식으로 데이터 시각화를 허용한다. 데이터 포인트 위치는 각도 값 및 앰프 수로 결정 된다 (ViewXY 의 X 는 각도로 비교, Y 는 앰프 수). 극성 뷰도 줌 및 패닝 기능을 갖고 있다.



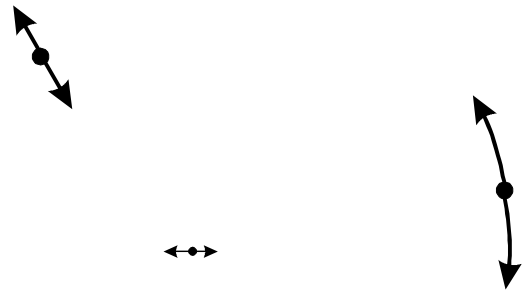
보기 10- 1. ViewPolar 객체 트리.

9.1 축

극성 축은 **Axes** 목록 속성으로 정의 가능하다. 같은 차트 내 여러 축을 사용할 수 있다. 시리즈의 **AssignPolarAxisIndex** 속성을 설정함으로 아무 축에 시리즈를 할당 가능하다. 축은 각도 스케일 및 앰프 스케일 둘다를 표현한다. 이 외에는 극성 축들은 ViewXY 의 축들과 매우 비슷하다 (6.2 를 보라).

AmplitudeAxisAngle	0
AmplitudeAxisAngleType	Relative
AmplitudeAxisLineVisible	True
AmplitudeLabelsAngle	**0**
AmplitudeLabelsVisible	True
AmplitudeReversed	False
AngleOrigin	0
AngularAxisAutoDivSpacing	True
AngularAxisCircleVisible	True
AngularAxisMajorDivCount	8
AngularLabelsVisible	True
AngularReversed	False
AngularTicksVisible	True
AngularUnitDisplay	Degrees
AntiAliasing	True
AutoFormatLabels	True
AxisColor	**Sienna**
AxisThickness	4
GridAngular	
GridVisibilityOrder	**BehindSeries**
InnerCircleRadiusPercentage	0
KeepDivCountOnRangeChange	True
LabelsFont	**Segoe UI, 9pt**
LabelTicksGap	5
MajorDiv	**6**
MajorDivCount	5
MajorDivTickStyle	
MajorGrid	
MarginInner	5
MarginOuter	5
MaxAmplitude	**30**
MinAmplitude	0
MinorDivCount	5
MinorDivTickStyle	
MinorGrid	
MouseDragSnapToDiv	False
MouseHighlight	Simple
MouseInteraction	True
MouseScaling	True
MouseScrolling	True
ScaleNibs	
TickMarkLocation	Outside
Title	**RoundAxisTitle**
Units	**RoundAxisTitle**
UsePreviousAxisDiameter	False
Visible	True

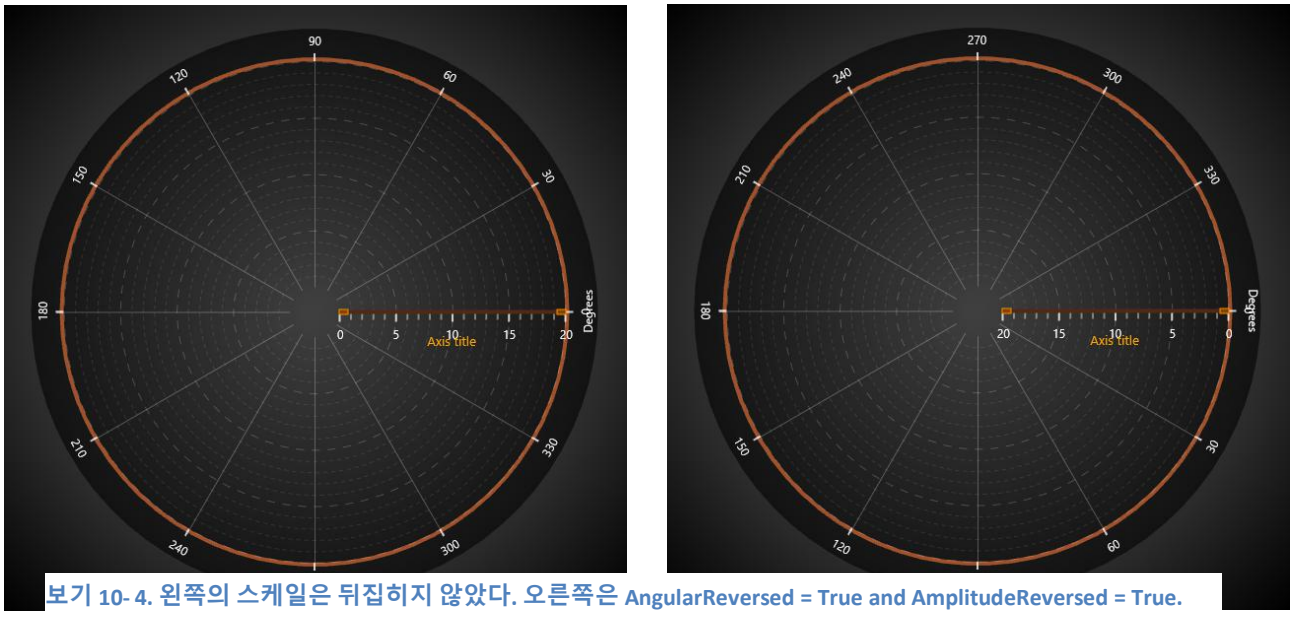
보기 10- 2. AxisPolar 속성 트리



보기 10- 3. 세개의 축. 첫째 (빨강) 바깥 원. 둘째 (초록) 중간. 셋째 (파랑) 중심에서 제일 가까움. 축 AngleOrigin 은 축 원 위로 드래그 해서 변경 가능하다. 앰프 범위는 축에서 부터 드래그 해서 변경 가능하다. 최소 또는 최대 축 앰프 범위는 앰프 스케일 끝에 작은 표시를 드래그 해서 변경 가능하다.

역 축

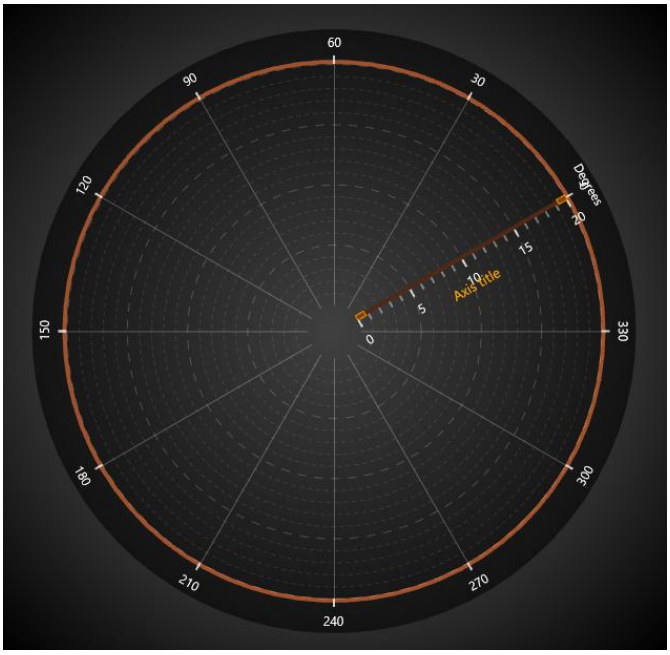
축은 앰프, 각도 또는 둘 다를 사용해 거꾸로 만들 수 있다. 각도 스케일을 뒤집기 위해 **AngularReversed = True** 로 설정하라. 앰프 스케일을 뒤집기 위해 **AmplitudeReversed = True** 설정하라.



보기 10- 4. 왼쪽의 스케일은 뒤집히지 않았다. 오른쪽은 AngularReversed = True and AmplitudeReversed = True.

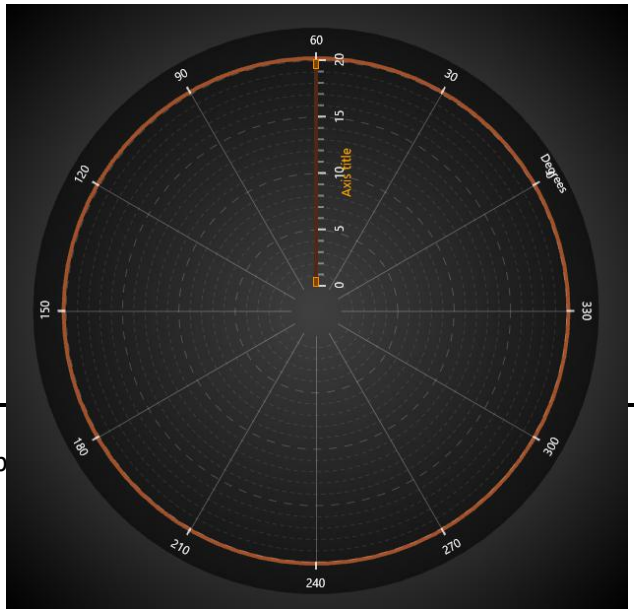
스케일의 회전 각도 설정

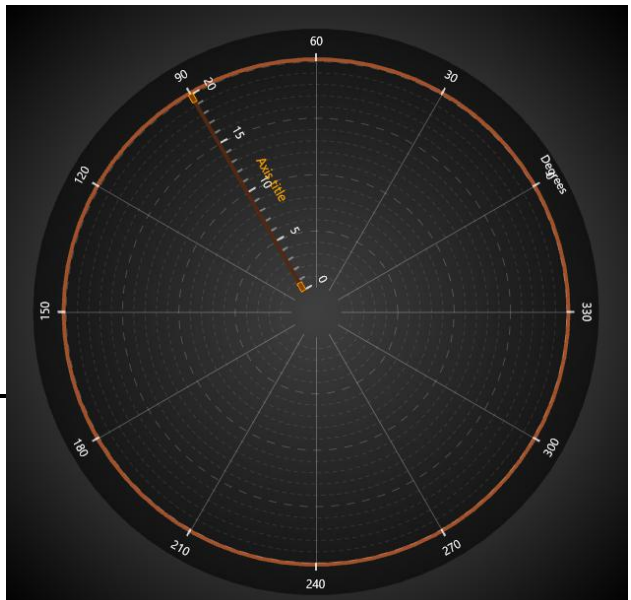
AngleOrigin 으로 각도 스케일의 회전 각도를 설정하라.



보기 10- 5. AngleOrigin = 30.

AmplitudeAxisAngle 로 앰프 축 위치를 회전하라. 앰프 스케일 각도는절대 값의 각도로 (**AmpitudeAxisAngleType = Absolute)** 또는 각도 스케이의 각도에 상대적 (**AmpitudeAxisAngleType = Relative)** 으로 설정 가능하다.





보기 10-6. AngleOrigin = 30. AmplitudeAxisAngle = 90. 왼쪽에는 AmplitudeAxisAngleType = Absolute. 오른쪽에는 AmplitudeAxisAngleType = Relative. 전체적으로 이 경우에는 앰프 스케일이 120 도 회전한다.

나누기 설정

앰프 나누기 카운트를 **MajorDivCount** 로 설정하고 나누기 크기를 **MajorDiv** 속성으로 설정하라. 앰프 스케일이 알아서 수정 된다 (**MaxAmplitude** 를 업데이트 하여). 앰프 마이너 나누기 카운트를 **MinorDivCount** 로 설정하라.

기본적으로 차트는 최대한 많은 각도 나눔을 포함하려 한다. 각도 나눔을 제어하기 위해 **AngularAxisAutoDivSpacing** 를 거짓으로 설정하라. 그러면 차트는 **AngularAxisMajorDivCount** 의 나눔 수를 시도한다. 차트 공간이 모든 나눔 및 라벨을 렌더링 하기에 너무 작으면 더욱 낮은 나누기 카운트를 사용한다.

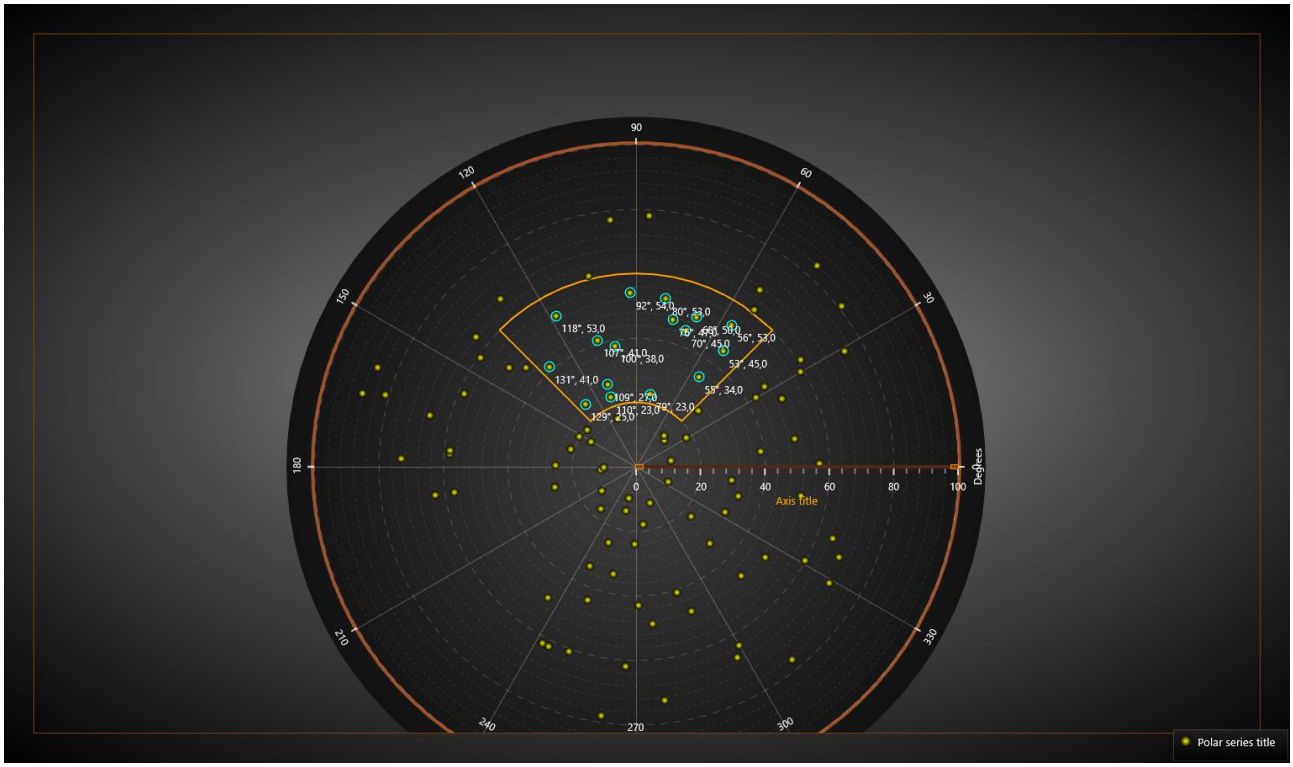
9.2 마진

AutoAdjustMargins 가 활성화 되었을 때, 그래프 크기는 모든 차트 및 차트 제목을 위한 공간을 확보하기 위해 수정된다. 비활성화 되었을 때 **ViewPolar.Margins** 속성이 적용되어 마진의 수동 설정을 허용한다.

런타임에서 마진 직사각형이 픽셀로 불러질 수 있다. 자동과 수동 마진 둘 다에 적용 되는 **ViewPolar.GetMarginsRect** 메소드를 불러라.이는 화면 좌표 기반 계산 또는 객체 놓을 때에 유용하다.

ViewPolar.MarginsChanged 이벤트는 마진 직사각형이 변경 되었을 때 실행 되게끔 설정할 수 있다.

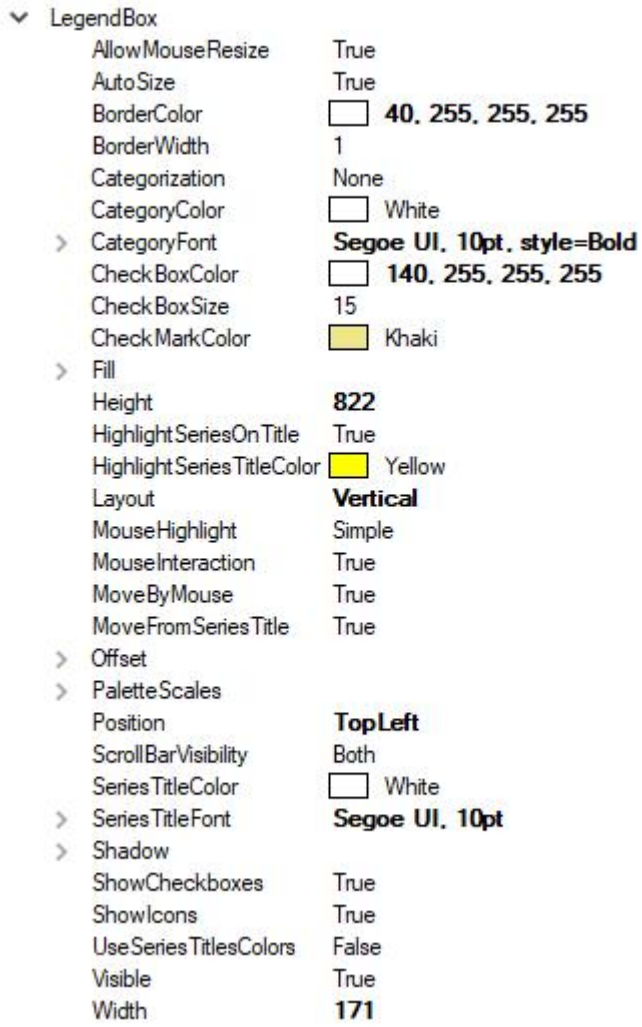
뷰의 내용은 마진 밖으로 자동으로 잘린다. 차트 제목, 주석 및 레전드 상자를 제외하고 모든 내용물이 잘린다. 이 셋의 위치는 화면 좌표로 정의되어 마진에도 자유롭게 놓일 수 있기 때문이다. 1 픽셀 넓이의 직사각형 태두리 **Border** 를 마진이 어디있는지 표시하기 위해 그릴 수 있다. 기본적으로 태두리는 ViewPolar 에 보이지 않는다. 직사각형의 색은 **Border.Color** 로 변경 가능하다.



보기 9-7. 마진 밖의 극성 차트의 내용은 잘렸다. 마진 구역을 표시하기 위해 태두리가 그려졌다. 축 라벨들은 태두리 안에서도 보인다.

9.3 레전드 상자

ViewPolar.LegendBox 로 레전드 상자 속성을 변경 하라. ViewXY 와는 다르게 ViewPolar 는 오직 한 레전드 상자만 사용할 수 있다.



보기 9-8. ViewPolar 에서 레전드 상자 속성.

팔레뜨 스케일 숨기기

레전드 상자에 팔레뜨 스케일을 숨기기 위해 **PaletteScales.Visible = False** 로 설정하라. 크기를 재설정 하기 위해 **ScaleSizeDim1** 및 **ScaleSizeDim2** 속성들을 설정하라.

ViewPolar 에서 레전드 상자 위치 선정

ViewPolar 의 레전드 상자는 자동 또는 수동으로 위치 선정이 가능하다. 자동 위치 선정은 뷰 또는 그래프 구역의 좌/상/우/하에 정렬 가능하다. 위치를 **Position** 속성으로 제어하라. 어느 위치선정 옵션은 마진 구역을 고려하고 어떤 옵션들은 하지 않는다.

마진을 무시하는 옵션들 (레전드 상자를 마진 구역 내 놓음):

TopCenter, **TopLeft**, **TopRight**, **LeftCenter**, **RightCenter**, **BottomLeft**, **BottomCenter**, **BottomRight, Manual**

레전드 상자를 마진 구역 내에 놓는 옵션들:

GraphTopCenter, **GraphTopLeft**, **GraphTopRight**, **GraphLeftCenter**, **GraphRightCenter**, **GraphBottomLeft**, **GraphBottomCenter**, **GraphBottomRight**

Offset 속성은 위치를 **Position** 속성에서 정해진 주어진 값 만큼 위치를 움직인다.

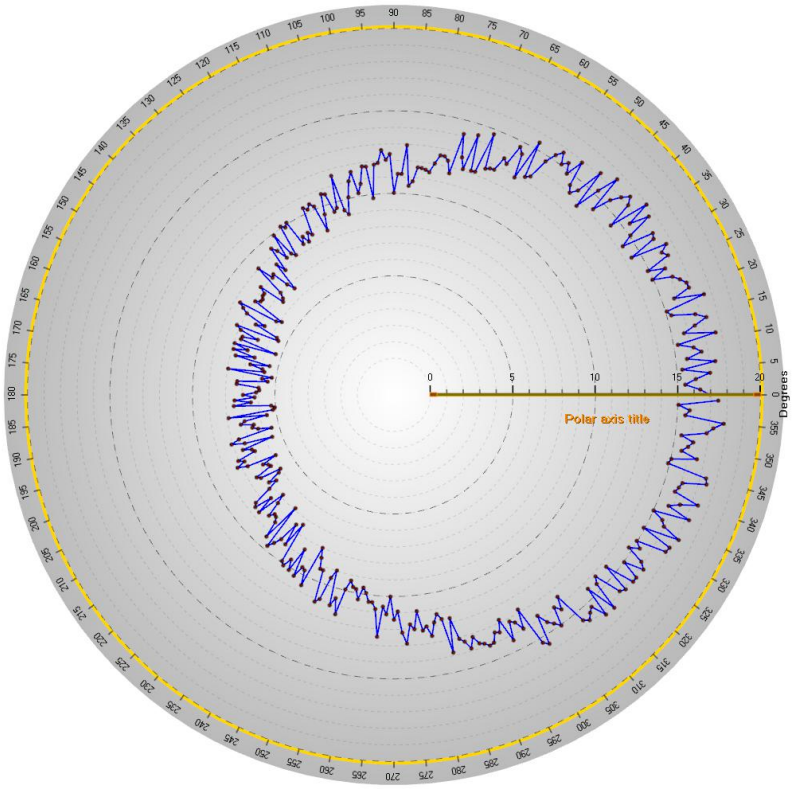
```
// Setting legend box position, offset shifts from RightCenter position
chart.ViewPolar.LegendBox.Position = LegendBoxPosition.RightCenter;
chart.ViewPolar.LegendBox.Offset = new PointIntXY(-15, -70);
```

Manual 위치 선정은 레전드 상자의 왼쪽 위 모서리와 뷰의 왼쪽 위 모서리의 오프셋을 계산한다. 이는 그래프 위에서 부터 계산하는 **TopLeft** 옵션과는 다르다.

9.4 PointLineSeries

데모 예시: 라인 시리즈, 섹터; 팔레뜨 색칠 라인 시리즈; 이벤트 색칠 라인 시리즈; 스캐터 포인트 선택

ViewPolar 의 **PointLineSeries** 는 라인, 포인트 그룹 또는 포인트 라인을 그리는 데에 사용 가능하다. 많은 라인 및 포인트 스타일이 **LineStyle** 및 **PointStyle** 속성들에 있다.



보기 9-9. ViewPolar 의 PointLineSeries 로 데이터가 표현 됐다. 라인 및 포인트가 둘 다 보인다.

데이터 설정

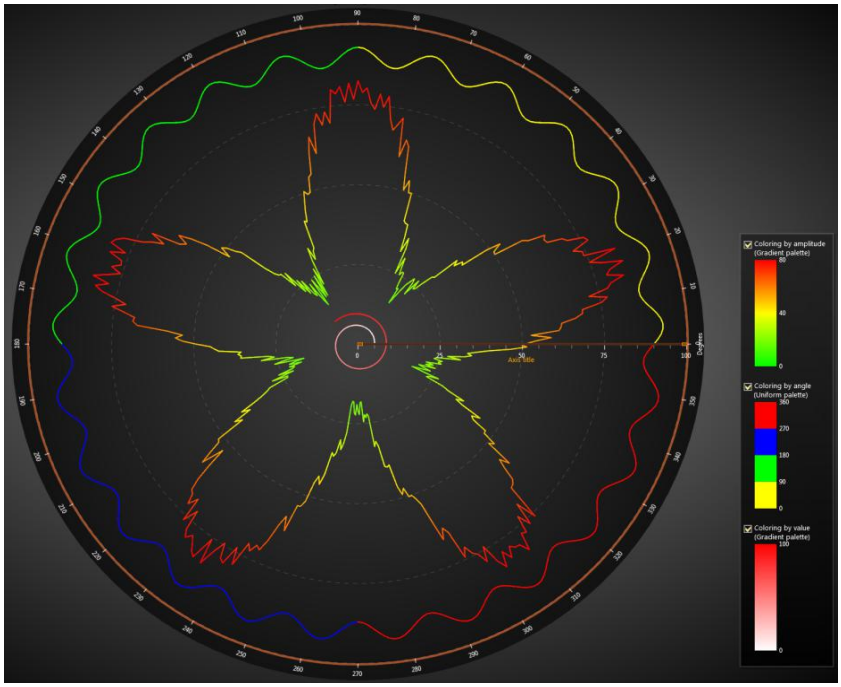
이전 보기의 데이터 설정을 표현하는 코드.

```
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;
```

팔레뜨 색칠

라인 색칠은 팔레뜨를 지원한다. **ColorStyle** 속성으로 팔레뜨 색칠이 어떻게 적용 되는지 선택 가능하다.

- **LineStyle**: 팔레뜨 필 없음. LineStyle.Color 속성에 있는 색이 적용됨.
- **PalettedByAngle**: 데이터 포인트 **Angle** 필드가 색을 정함
- **PalettedByAmplitude**: 데이터포인트 **Amplitude** 필드가 색을 정함
- **PalettedByValue**: 데이터 포인트 **Value** 필드가 색을 정함



보기 9-10. 팔레뜨 색칠 적용

ValueRangePalette 속성을 이용해 색 및 값 단계를 정의하라. ViewXY 와 View3D 의 시리즈와 비슷하게 적용 된다.

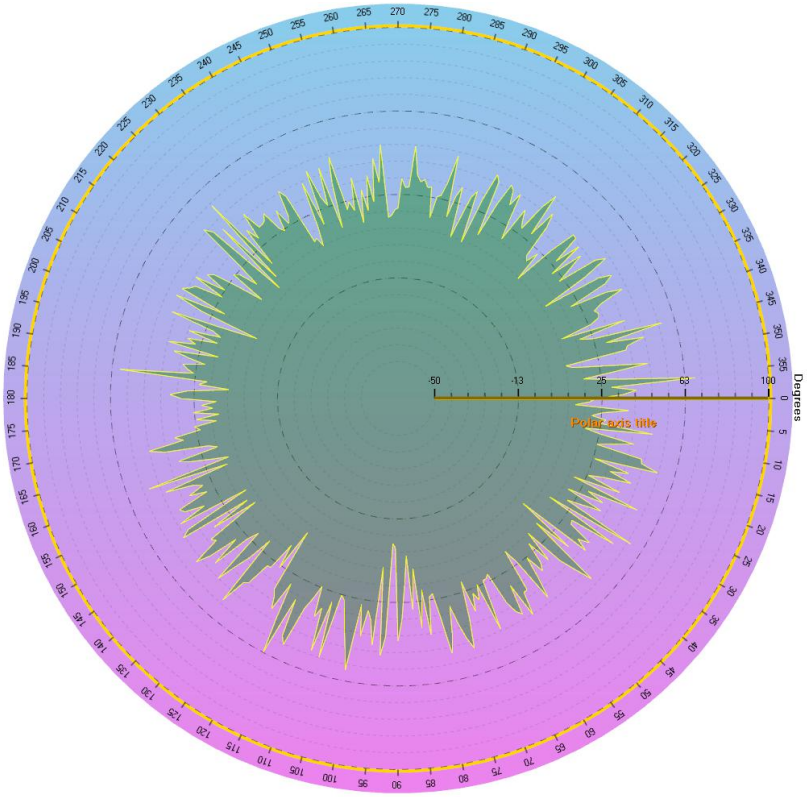
CustomLinePointColoringAndShaping 이멘트로 커스텀 쉐이핑 및 색칠

커스텀 색칠 및 좌표 수정은 **CustomLinePointColoringAndShaping** 이벤트로 만들 수 있다. 이 이벤트는 차트의 렌더링 단계 입성 직전 불러진다. 이는 ViewXY의 **FreeformPointLineSeries** 의 **CustomLinePointColoringAndShaping** 이벤트와 비슷하게 작용한다 (6.9.2를 보라).

9.5 AreaSeries

데모 예시: 구역 시리즈; 마커로 합침; 거미/레이더 차트; 속도계 게이지

구역 시리즈는 채워진 구역 스타일로 데이터 시각화를 허용한다. 가장자리의 라인 스타일은 **LineStyle** 속성으로 수정 가능하다. **FillColor** 속성으로 채우기 변경 가능하다.



보기 9-11. ViewPolar 의 AreaSeries 로 표현된 데이터

데이터 설정

이 코드는 이전 보기의 데이터 설정을 표현한다.

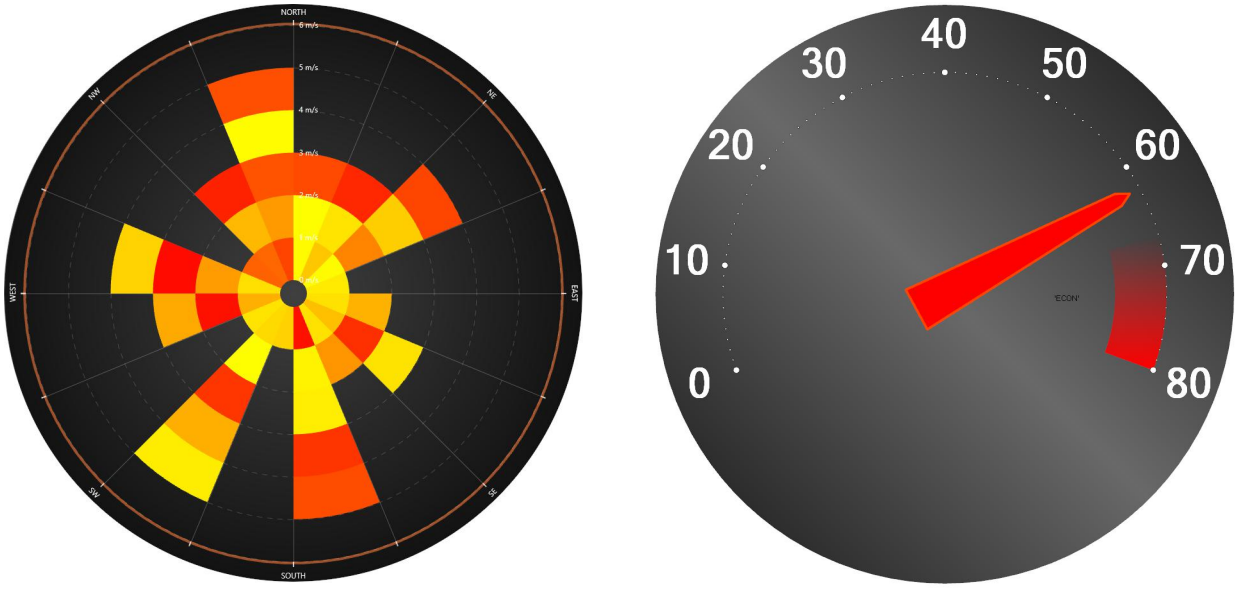
```
int iCount = 360;
```

```
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;
```

9.6 섹터

데모 예시: 라인 시리즈, 섹터; 윈드 로즈 도표; 스캐터 포인트 선택; 스캐닝 레이더

Sectors 는 어느 각도 및 앰프 범위를 표시하기 위해 정의 가능하다. 앰프 범위를 **MinAmplitude** 및 **MaxAmplitude** 속성들로 정의하라. 각도 범위를 **BeginAngle** 및 **EndAngle** 로 정의하라. 마우스 드래그로 섹터를 움직여라.

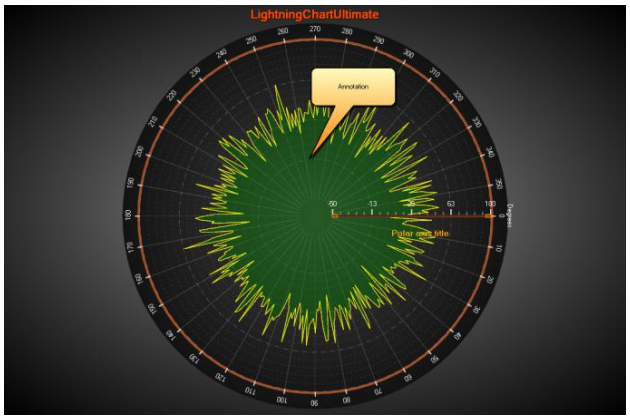


보기 9-12. 섹터를 사용한 두개의 예시. 첫번째 윈드 로즈 도표는 다른 색의 여러 섹터로 만들어 졌다. 두번째 그림에는 AreaSeries 로 다이얼이 RPM 미터기 레드존과 같이 만들어 졌다.

9.7 주석

데모 예시: 벡터

Annotations 은 ViewXY 의 **Annotations** 과 비슷하다 (6.20 장). **Target** 및 **Location** 이 극성 축 값으로 정의 된거 제외하고 말이다. 축 값대로 크기 설정은 적합하지 않아 **Sizing** 속성은 **Automatic** 및 **ScreenCoordinates** 값들만 갖고 있다.



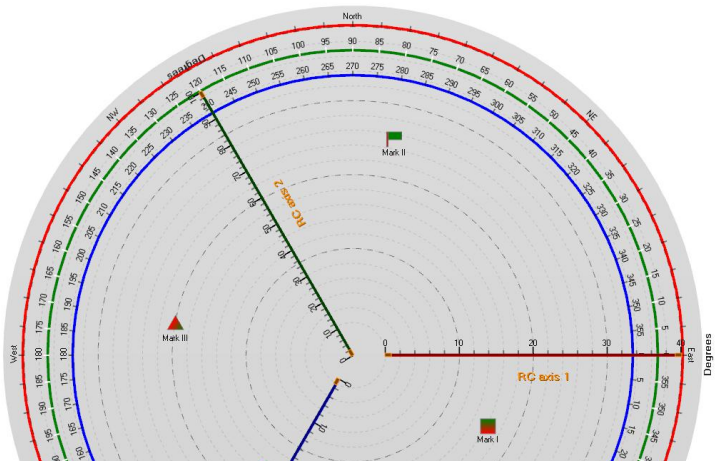
보기 9-13. 극성 뷰에 주석

9.8 마커

데모 예시: 스캐터 포인트 선택; 마커로 합쳐짐; 소나 생선 시표; 스캐닝 레이더

마커는 어느 위치에 있는 특별 데이터 값을 표시하기 위해 사용할 수 있다. **AssignPolarAxisIndex** 를 설정해 선호 축에 마커를 할당하라. **Amplitude** 및 **AngleValue** 속성들을 정의해 자리에 놓아라. **Symbol** 을 수정해 원하는 생김세 및 **Label** 속성으로 마커 텍스트를 정의하라.

마커는 마우스로 드래그해서 움직일 수 있다. **SnapToClosestPoint 를 Selected** 또는 **All** 로 설정해 드래그 할때 가장 가까운 데이터 포인트로 바로 가기를 설정하라. **Selected** 는 이 마커가 **SetSnapSeries()** 메소드로 바로 가게 설정된 시리즈만 트래킹한다. **All** 은 모든 시리즈를 트래킹한다.



보기 9- 14. 극성 차트에 두 마커

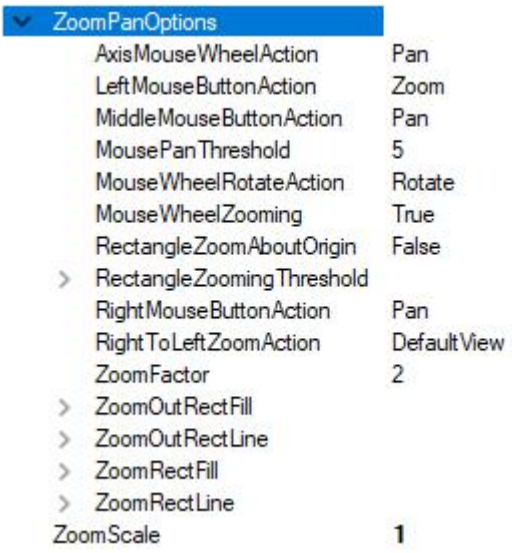
9.9 줌 및 패닝

줌은 코드로 적용 가능하다. **ZoomCenter** 및 **ZoomScale** 속성들을 설정하라. **ZoomCenter** 는 상대적 X-Y 범위로 정의 된다.

X = -1: 극성 뷰의 왼쪽 가장자리가 차트 중심에
X = 0: 극성 뷰의 중심이 차트 구역의 중심에
X = 1: 극성 뷰의 오른쪽 가장자리가 차트 구역의 중심에

Y = -1: 극성 뷰의 하단 가장자리가 차트 구역 중심에
Y = 0: 극성 뷰의 중심이 차트 구역 중심에
Y = 1: 극성 뷰의 상단 가장자리가 차트 구역 중심에
ZoomScale 은 확대 팩터다. 예를 들어 2 의 값은 X 와 Y 방향에 차트를 1 보다 두배로 크게 보이게 설정한다.

마우스 줌 기능이 **ZoomPanOptions** 속성 트리에서 구성 가능하다.



보기 9- 15. ViewPolar 의 ZoomPanOptions.

줌 연산 및 메소드

ZoomPanOptions 아래 있는 여러 줌 연산은 마우스 액션으로 설정 가능하다. **DefaultSettings** 은 초기 줌 및 센터링 설정으로 돌린다. **ZoomToData (v8.4** 이전 **FitView** 라 불림)은 뷰포인트를 마진 내 모든 데이터를 보여주게끔 움직인다. **ZoomToLabelsArea** 은 마진 내 라벨까지 포함한 전체 데이터 프레임을 보여준다.

ViewPolar 는 **ZoomPadding** 속성을 갖고있다. 이는 View3D 와 비슷하게 작용한다 (7.18.3 장).

줌 연산은 코 메소드 **ZoomToFit(ZoomAreaRound.AreaName)**를 불러 접근 가능하다. 예를 들어 **ZoomToFit(ZoomAreaRound.LabelsArea)** 메소드를 부르는 것은 마우스 액션으로 **ZoomToLabelsArea** 연산을 실행 시키는 것과 같은 결과를 낸다.

보기 9-16. 줌 연산 전 후의 극성 차트, ZoomPadding = 50. 위에는 차트가 수동적으로 확대 되었지만 아무런 줌 연산이 불러지지 않았다. ZoomPadding 은 효과가 없다. 아래에는 ZoomToLabelsArea 가 사용됐다. 이는 줌할 때 라벨까지 신경 쓴다.

9.10 ViewPolar 에서 데이터 클리핑

PointLineSeriesPolar, AreaSeriesPolar, Sectors 와 PolarEventMarkers 는 **ClipInsideGraph** 속성을 갖고 있다. 이는 그래프 구역 내에 위치 되어 있지 않으며 데이터 포인트를 숨긴다. 정확한 클리핑 지점은 가장 바깥 각진 축이다. 기본적으로 **ClipInsideGraph** 는 모든 시리즈 위해 활성화 되었다.

```
// Disabling clipping outside the graph
pointLineSeriesPolar.ClipInsideGraph = false;
```

CenterClipping 은 라이트닝차트 버전 8.5.1 에 소개 되었다. 이는 **ClipInsideGraph** 와 비슷하게 작용하지만 극성 차트 중심에 데이터가 어떻게 클리핑 되는지 제어한다. 예를 들어 앰프 축이 마우스로 드래그 되었을 때. **CenterClipping** 은 세개의 옵션이 있다

-**None**: 버전 8.5.1 이전 행동. 시리즈가 중심점 반대쪽으로 움직여지고 아무런 클리핑이 일어나지 않는다 (섹터 제외).

-**Center**: 데이터가 그래프 중심점에 클리핑 되고 반대쪽으로 절대 움직여 지지 않는다. 이것이 기본 설정이다.

-**InnerCircle**: 데이터가 앰프 축의 가장 안쪽 값대로 클리핑 된다. 이는 축이 뒤집혔는지에 따라 축의 최소 또는 최대 값이 될 수 있다. 차트가 여러 앰프 축을 갖고 있다면 할당된 축에 의해 클리핑 된다.

```
// Setting PointLineSeriesPolar to be clipped below amplitude axis minimum, as reversed axis is not used.
_chart.ViewPolar.Axes[0].AmplitudeReversed = false;
pointLineSeriesPolar.CenterClipping = CenterClipping.InnerCircle;
```

Center 및 **InnerCircle** –옵션들은 항상 같은 위치에 있지 않다. 그래프 중심 근처 빈 공간을 놓을 수 있는 **InnerCircleRadiusPercentage** 속성이 있기 때문이다. 다른 말로 이는 앰프 축이 어디에서 시작 되는지 정의한다. **InnerCircleRadiusPercentage** 은 설정된 축에 특수하며 다른 앰프 축들을 건드리지 않는다.

```
// Setting InnerCircleRadiusPercentage to 10 percent for this axis
chart.ViewPolar.Axes[0].InnerCircleRadiusPercentage = 10;
```

보기 9-110-7. 왼쪽에는 CenterClipping 이 None 으로 설정. 가운데는 Center. 오른쪽 이미지에는 InnerCircle. InnerCircleRadiusPercentage 은 세 이미지 다 10 으로 설정 됐다.

10. ViewSmith

스미스 차트는 주로 전자 임피던스 측정 및 임피던스 매칭 어플리케이션에 사용 된다.

스미스 차트는 데이터를 실제 및 상상 값으로 플롯한다 **(R + jX).**

Terms

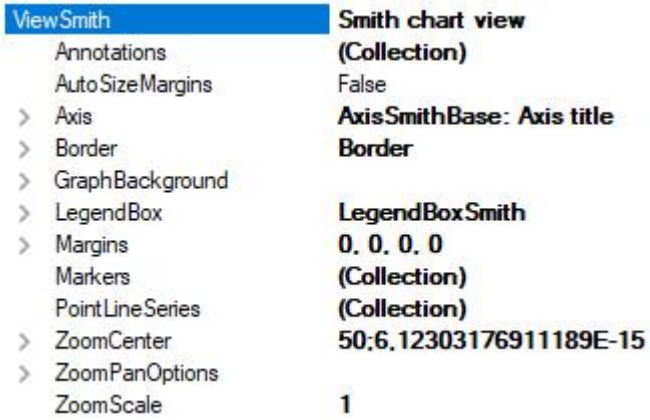
Impedance = **Z** = **R** + j**X**

R = Resistance, Real part

X = Reactance, Imaginary part

X > 0: Capacitive
X < 0: Inductive

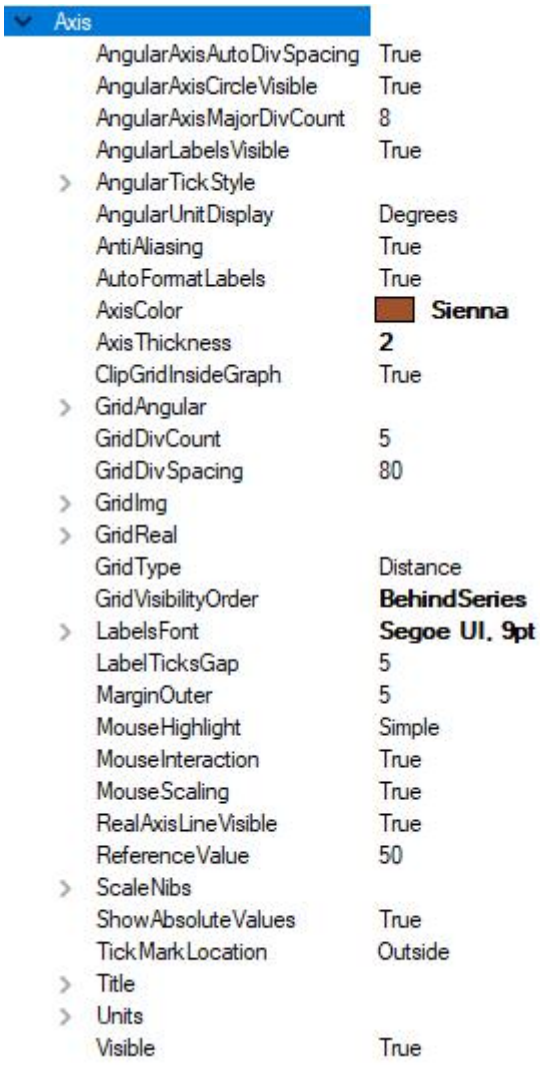
데이터 위치는 2D 플롯에 원형 Real 및 Imaginary 로그-로그 스케일에 정해진다.



보기 11- 1. ViewSmith 속성 트리.

10.1 축

스미스 차트는 축이 1개 밖에 없다. 이는 확장된 속성 트리 **Axis** 로 구성 가능하다. 보기 10-2를 보아라.



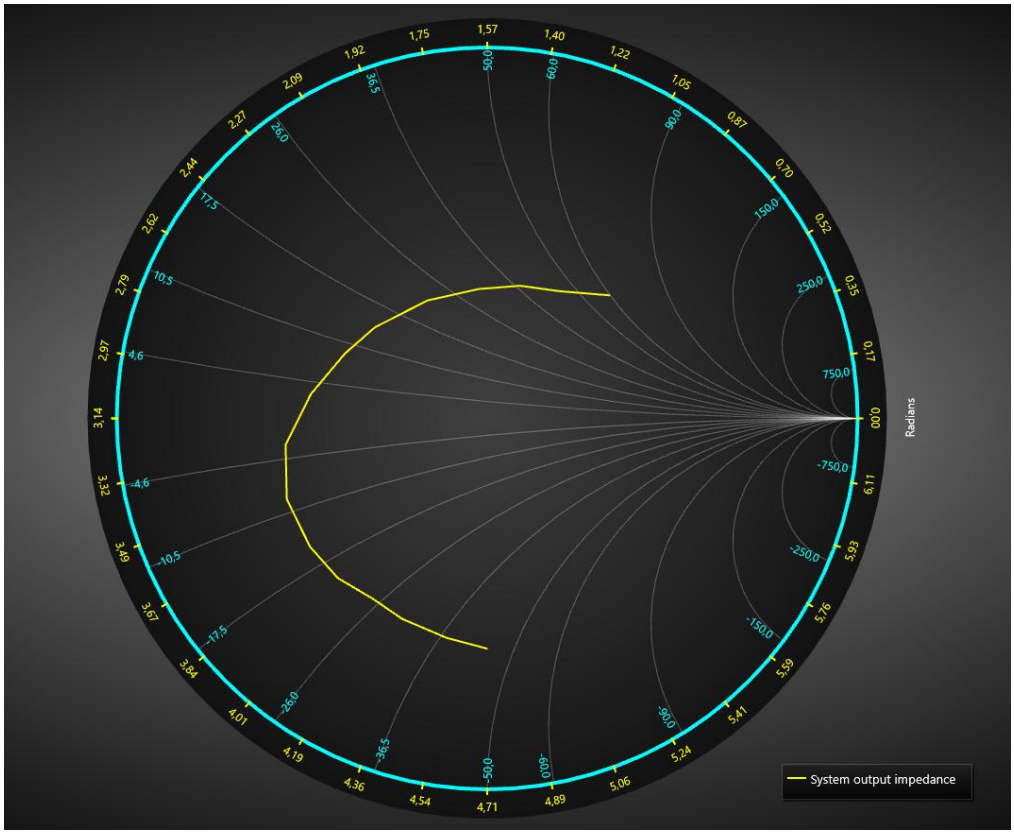
보기 11- 2. Smith 축 속성 트리.

대부분의 외관 설정 속성은 ViewPolar의 축과 ViewXY의 축과 똑같다. ViewSmith 수정 특수 고급 속성이 있다. **GridDivCount**, **GridImg** and **GridReal**, **RealAxisLineVisible**, **ShowAbsoluteValues**, **ClipGridInsideGraph** 등이 있다.

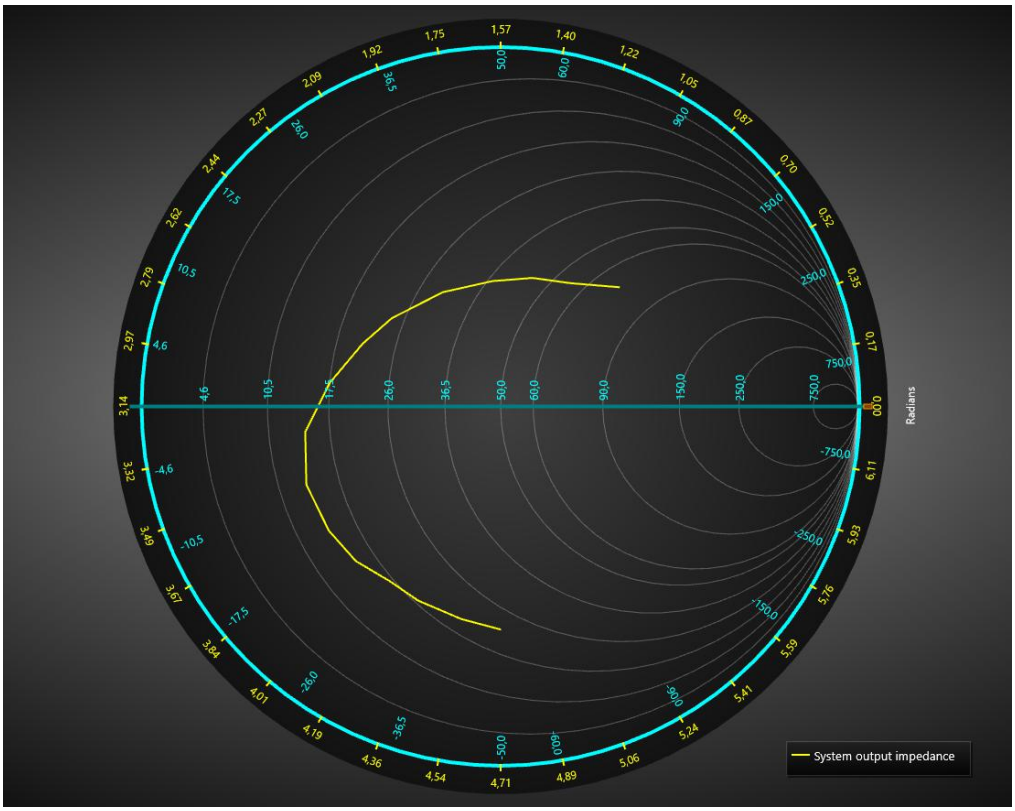
GridDivCount 는 실제 축 및 상상 스케일 위 로그 그리드 라인의 수를 정의한다.

GridImg 및 **GridReal** 속성들은 **Real** 또는 **Imaginary** 스케일에 그리드 라인을 커스터마이징하는 데에 쓰인다. 추가로 **Visible** 속성을 사용해 그리드를 숨길 수 있어 사용자가 하나를 숨기고 다른 것을 작업할 수 있다.

RealAxisLineVisible 속성은 축 라인을 숨긴다.



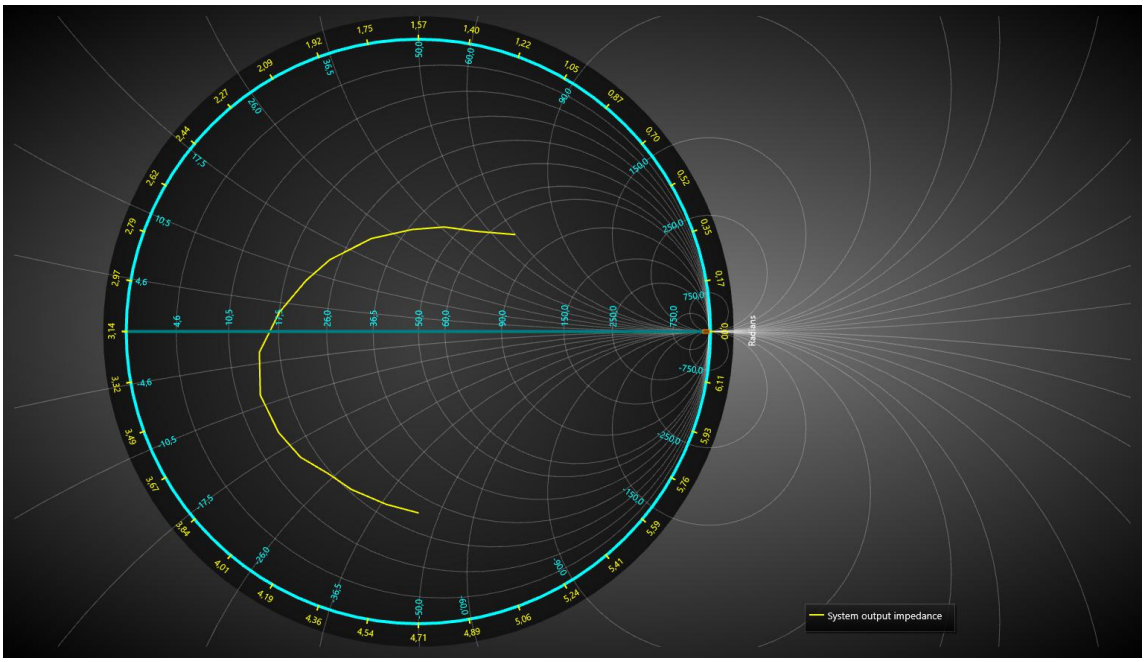
보기 11- 3. 실제 그리드 라인이 숨겨졌다. 상상 라인이 보인다.



보기 11- 4. 상상 그리드 라인이 숨겨지고 실제 라인이 보인다.

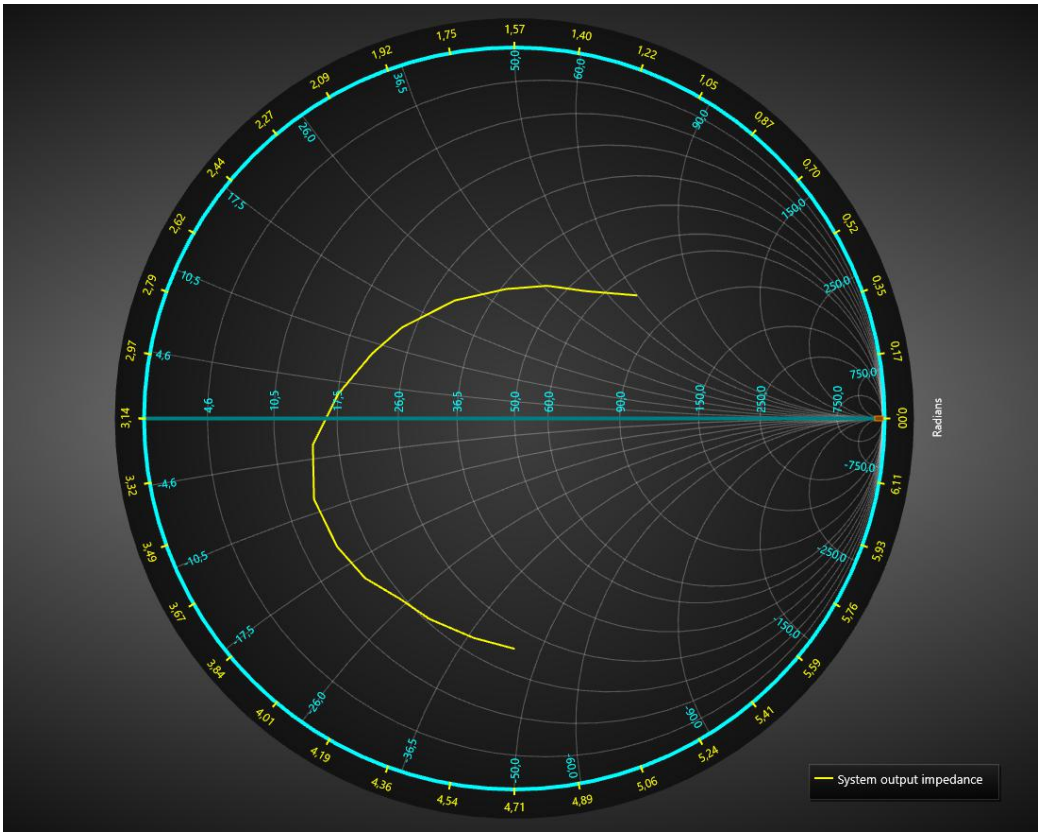
ShowAbsoluteValues 속성은 스케일 위 값을 정의한다 (절대 또는 노멀화).

ClipGridInsideGraph 은 차트 원 밖에 그리드 라인이 보이게 한다.



보기 11- 5. ClipGridInsideGraph = False.

완전히 커스터마이징 된 스미스 차트를 아래에서 볼 수 있다.



보기 11- 6. 커스터마이징 된 스미스 차트

10.2 마진

AutoAdjustMargins 가 활성화 되었을 때 그래프 크기가 모든 축 및 차트 제목을 위한 공간을 확보 하고자 수정 된다. 비활성화 되었을 때는 **ViewSmith.Margins** 속성이 적용 되어 마진의 수동 설정을 허용한다.

런타임에서 **ViewSmith.GetMarginsRect** 메소드를 불러 마진 직사각형을 픽셀로 불러올 수 있다. 이는 자동 및 수동 마진 둘 다에 적용 된다. 이는 화면 좌표 기반 계산 또는 객체 놓을 때 유용하다.

ViewSmith.MarginsChanged 이벤트를 마진 직사각형의 크기 등이 변경 되었을 때 실행 하게 설정 가능하다.

뷰의 내용물은 마진 밖으로 자동 클리핑 된다. 차트 제목, 주석 및 레전드 상자 외엔 모든 내용이 클리핑 된다. 이의 위치가 화면 좌표로 정의 되어 마진에서도 자유롭게 놓일 수 있게 되서 그렇다. 1 픽셀 넓이의 태두리 직사각형, **Border** 를 그려 마진이 어디에 있는지 표시할 수 있다. 기본적으로 태두리는 ViewSmith 에 보이지 않는다. 직사각형의 색은 **Border.Color** 로 변경 가능하다.

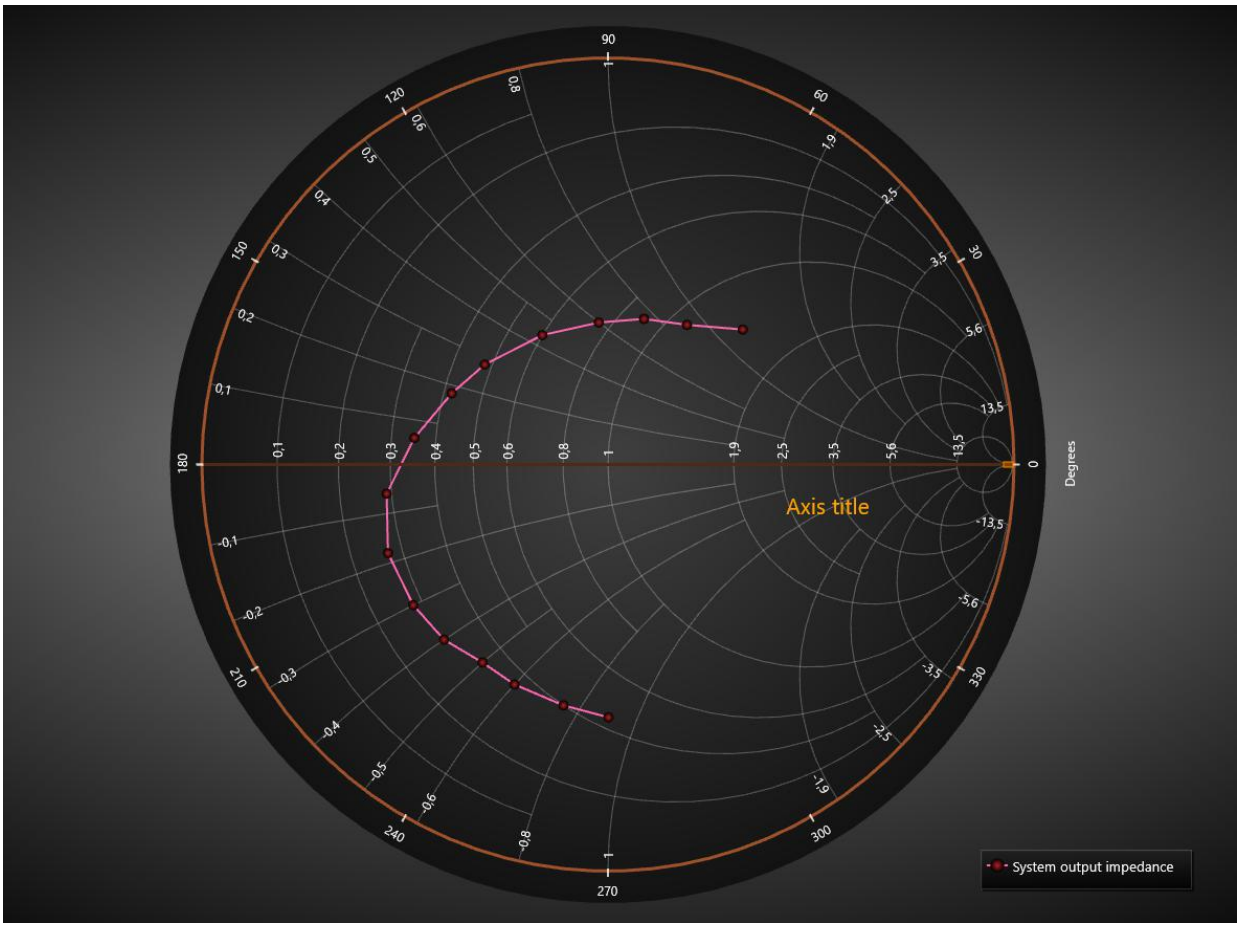
10.3 레전드 상자

ViewSmith 에서 레전드 상자는 ViewPolar 와 똑같이 작용한다 (10.3 장을 보라). **ViewSmith.LegendBox** 로 레전드 상자 속성을 수정하라.

10.4 PointLineSeries

데모 예시: 라인 및 데이터 커서

ViewSmith 의 **PointLineSeries** 로 ViewPolar 에서와 같이 라인, 포인트 그룹 또는 포인트 라인을 그려라. **LineStyle** 및 **PointStyle** 속성에 많은 라인 및 포인트 스타일을 사용할 수 있다.



보기 11-7. 스미스 데이터 시리즈.

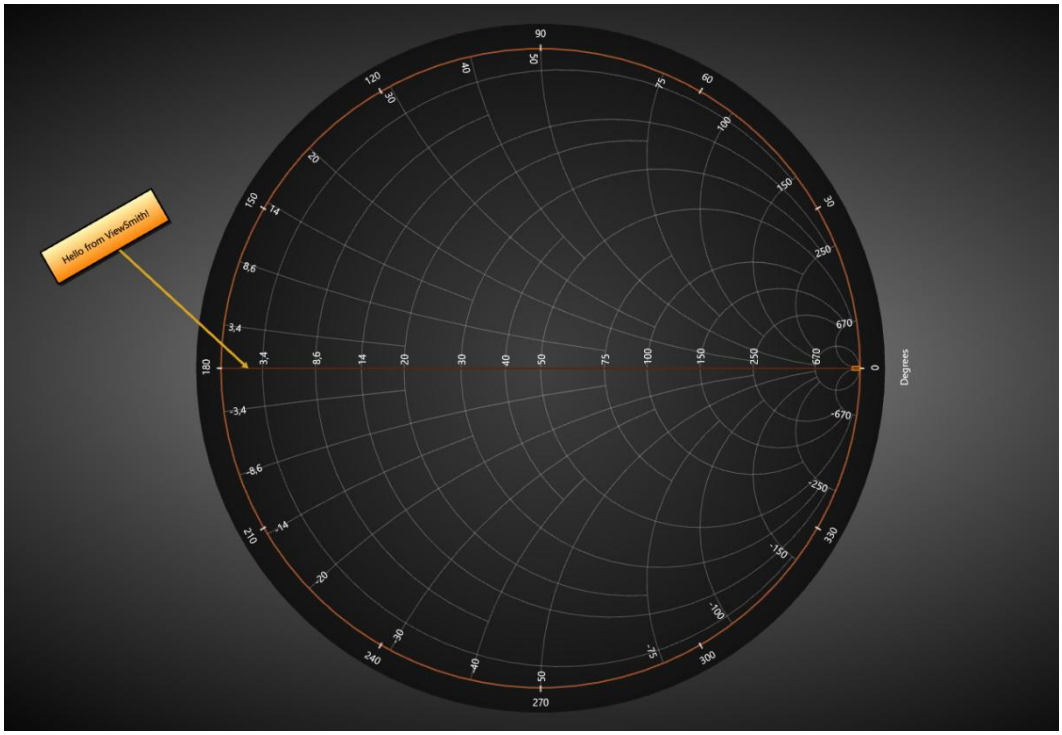
10.5 데이터 설정

아래 코드는 스미스 차트 컬렉션에 한 데이터 포인트 세트를 추가할 것이다.

```
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);
```

10.6 주석

Target 및 **Location** 이 스미스 축 값 (실제 및 상상)으로 정의 된 것 외에 **Annotations** 는 ViewXY의 **Annotations** 과 똑 같 다 (6.20 장). **Sizing** 속 성 은 **Automatic** 및 **ScreenCoordinates** 두 값 만 갖 고 있 다 .



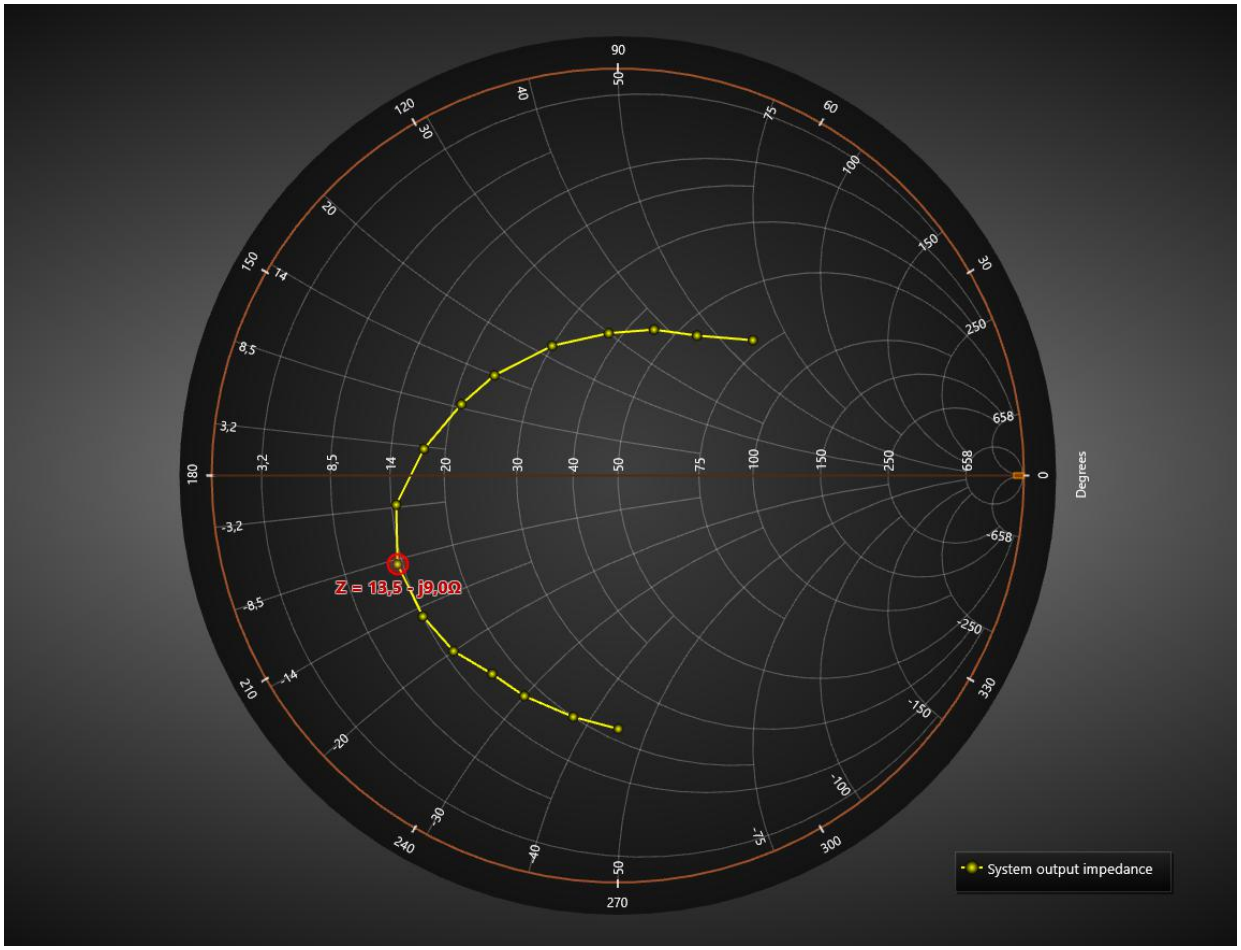
보기 11- 8. ViewSmith 에서 주석.

10.7 마커

데모 예시: 라인 및 데이터 커서

마커는 특유 위치에 특별 데이터 값을 표시하는 데 사용 가능하다. 마커는 마우스로 드래그해서 움직일 수 있다. 이 속성은 ViewPolar의 마커와 똑같은 정의를 갖고 있다 (10.8장을 보라).

ImgValue 및 **RealValue** 속성들을 정의해 움직여라. **Symbol** 을 수정해 원하는 생김세로 설정하고 텍스를 **Label** 속성으로 정의하라.



보기 11- 9. 스미스 뷰에서 시리즈를 트래킹하고 있는 마커

10.8 줌 및 패닝

ViewSmith 에서 줌 및 패닝 옵션 및 메소드는 ViewPolar 에서와 똑같이 작용한다 (10.9 장).

11. 색 테마 설정

차트의 전체적 색 테마는 **ColorTheme** 속성으로 설정 가능하다. 테마를 설정하는 것은 차트에 생성된 대부분의 객체 색을 덮어 쓴다. **ColorTheme** 을 먼저 설정하고 그 후에 객체 색을 수정하는 것을 추천한다.

```
chart.ColorTheme = ColorTheme.SkyBlue; // Changing the color theme
```

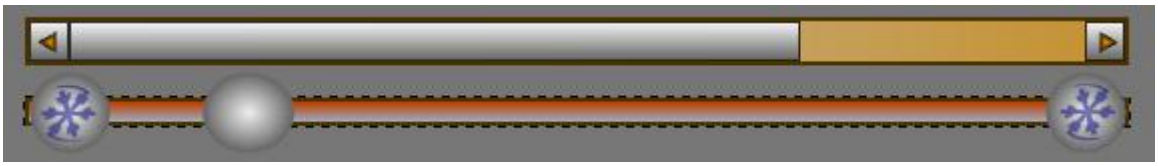
주의! 색 테마를 설정함으로 각 수동 할당 된 색은 비주얼 스튜디오 속성 그리드에 경고 없이 사라진다.

보기 12- 1. 다른 색 테마 사용. 왼쪽에는 커스텀 색이 있는 다크 테마. 오른쪽에는 연파랑 테마 설정.

12. 스크롤바

데모 예시: 스크롤바; 역사적 데이터 리뷰; 스케일 브레이크

하나 이상의 스크롤바를 **HorizontalScrollBars** 또는 **VerticalScrollbars** 컬렉션 속성으로 추가 가능하다. 생김세는 완전히 커스터마이징 할 수 있다. 타원형 버튼 및 스크롤 박스까지 정의 가능하다. 예를 들어 비트맵을 버튼 아이콘으로 사용 가능하다. 스크롤바는 모든 뷰와 같이 사용 가능하지만 ViewXY 에서 가장 잘 쓰인다.



보기 13- 1. 두개의 다르게 생긴 스크롤바

12.1 스크롤바 속성

HorizontalScrollBar 는 그래프 넓이에 맞게 정렬 될 수 있다. **Alignment** 속성을 **BelowGraph**, **AboveGraph** 또는 **GraphCenter** 로 설정하라. **VerticalScrollBar** 또한 그래프 높이에 맞게 정렬 가능하다. **Alignment** 속성을 **LeftToGraph**, **GraphCenter** 또는 **RightToGraph** 로 설정하라. **Alignment** 를 **None** 으로 설정하면 스크롤바는 **Offset** 속성으로 자유롭게 움직일 수 있다. 크기를 **Size** 속성으로 수정하라.

```
// Setting the position and the size of a scrollbar. Offset is based on the
   top-left corner of the chart
horizontalScrollBar.Alignment = HorizontalScrollBarAlignment.None;
horizontalScrollBar.Offset = new PointIntXY(100, 10);
horizontalScrollBar.Size = new Size(500, 30);
```

스크롤바는 64-bit 비사인 정수 값을 더욱 많이 쓰이는 32-bit 사인된 정수 값을 사용한다. **Value** 이 현재 위치다. **Minimum** 은 최소 값 범위고 **Maximum** 이 초대 값 범위다. 이는 높은 샘플링 회수를 가진 긴 측정의 직접 지원을 허용한다. 예를 들어 **SampleDataSeries** 가 측정에 사용 되면 샘플 인덱스를 스크롤바 값으로 직접 설정하라. **Minimum** 값이 첫 샘플 인덱스를 표현하고 **Maximum** 은 마지막 샘플 인덱스를 표현한다.

SmallChange 속성이 스크롤 버튼이 눌렸을 때 증가 및 감소하는 정도다. **KeyControlEnabled** 가 활성화 되고 스크롤바에 포커스가 있다면 화살표 키를 사용해 **Value** 를 **SmallChange** 만큼 변경 가능하다. **LargeChange** 는 페이지 변경을 표현한다. 이는 크르롤 박스 또는 스크롤 버튼 밖에 스크롤바가 눌리면 일어난다. **PageUp** 및 **PageDown** 키를 사용해 값을 변경하라. **MouseWheelChange** 는 스크롤바 위에 마우스휠이 스크롤 되었을 때 값이 변경된다.

Scroll 이벤트 핸들러를 코드에 사용해 스크롤바 값 변경에 반응을 하게 설정 가능하다. **ValueChanged** 이벤트 핸들러도 사용 가능하다. 하지만 **Scroll** 이 어떻게 스크롤이 되었는지에 대한 훨씬 많은 정보를 제공한다.

12.2 소수점 또는 음수 값의 스크롤바

스크롤바는 긴 측정을 지원하기 위해 사인 안된 정수를 사용하게끔 디자인 되었기에 축 값이 소수 또는 음수 값일 때 바로 사용할 수 없다. 이런 경우에는 결과가 에러 또는 반올림 오류로 인한 나쁜 유용성일 것이다. 하지만 스크롤바 **Minimum** 및 **Maximum** 값들은 축 범위에 묶여 있지 않아도 된다. 이는 축 값 및/또는 데이터 값이 작은 소수 또는 음수여도 바의 스케일링을 가능케 한다.

예를 들어 보여질 값의 범위가 0.500 에서 1.500 라면 스크롤바를 500->1500 의 범위를 사용하게 설정 가능하다. 비슷하게 -150 에서 0 의 데이터 범위가 주어졌으면 스크롤바는 0 에서 150 의 값을 사용할 수 있다.

다음 예시는 Y 축 값들이 음수에서 양수 사이의 소수 값들일 때 수직 스크롤바를 어떻게 변경할 수 있는지 보여준다.

```
double yMin = -0.178; // Some arbitrary axis values
double yMax = 1.253;
double upShift = 0.178; // To prevent the scrollbar from using negative values
double scaleFactor = 1000.0; // To scale the scrollbar to use integer values
_chart.ViewXY.YAxes[0].Minimum = yMin;
_chart.ViewXY.YAxes[0].Maximum = yMax;
_chart.VerticalScrollBars[0].Minimum = (ulong)((yMin + upShift) * scaleFactor);
_chart.VerticalScrollBars[0].Maximum = (ulong)((yMax + upShift) * scaleFactor);
_chart.VerticalScrollBars[0].LargeChange = _chart.VerticalScrollBars[0].Maximum
          - _chart.VerticalScrollBars[0].Minimum;
_chart.ViewXY.YAxes[0].SetRange(0.3, 0.6); // Display only a portion of the axis


// The scroll event
private void VerticalScrollBar_Scroll(object sender, ScrollEventArgs e)
{
    double newMin = (yMax + upShift) * scaleFactor - (double)e.NewValue -
           currentRangeY + (yMin + upShift) * scaleFactor;
    if (newMin < (yMin + upShift) * scaleFactor)
    {
        // The scroll bar cannot go below the minimum axis value
        newMin = (yMin + upShift) * scaleFactor;
    }

    double newMax = newMin + currentRangeY;
    if (newMax > (yMax + upShift) * scaleFactor)
    {
        // The scroll bar cannot go above the maximum axis value
        newMax = (yMax + upShift) * scaleFactor;
        newMin = newMax - currentRangeY;
    }
```

```
// Adjusting axis range based on the scrollbar value, triggers RangeChanged
event. Converts the values used by the scrollbar back to unscaled axis values
    _chart.ViewXY.YAxes[0].SetRange((newMin / scaleFactor - upShift) ,
           (newMax / scaleFactor - upShift) );
}

// RangeChanged -event to modify the axis ranges correctly
private void AxisY_RangeChanged(object sender, RangeChangedEventArgs e)
{

     // Prevent axis minimum value from going below yMin set earlier
     if (e.NewMin < yMin)
     {
         double newRange = e.NewMax - e.NewMin;
         if (newRange <= yMax - yMin)
         {
             _chart.ViewXY.YAxes[0].SetRange(yMin, yMin + newRange);
         }
         else
         {
             _chart.ViewXY.YAxes[0].SetRange(yMin, yMax);
         }
     }
     // Prevent axis maximum value from going above yMax set earlier
     else if (e.NewMax > yMax)
     {
         double newRange = e.NewMax - e.NewMin;
         if (newRange <= yMax - yMin)
         {
             _chart.ViewXY.YAxes[0].SetRange(yMax - newRange, yMax);
         }
         else
         {
             _chart.ViewXY.YAxes[0].SetRange(yMin, yMax);
         }
     }
     // Modify the scrollbar based on the the new axis range.
     else
     {
         double newRange = (e.NewMax - e.NewMin) * scaleFactor;
         _chart.BeginUpdate();

         _chart.VerticalScrollBars[0].LargeChange = (ulong)(newRange + 0.1);
         _chart.VerticalScrollBars[0].Value = (ulong)((yMax - e.NewMax +
           yMin + upShift) * scaleFactor);

         _chart.EndUpdate();

         currentRangeY = newRange;
     }
}
```

13. 수출 및 프린트

13.1.1 비트맵 이미지 수출

차트를 **SaveToFile()** 메소드로 .PNG, .BMP and .JPG 파일로 수출 가능하다. **SaveToFile(…)** 메소드는 해상도 감소 및 부드럽게/앤티 앨리어싱 옵션 있는 이미지 파일 수출을 허용한다. 스트림을 수출할 때에 **SaveToStream()** 메소드를 사용하라.

13.1.2 벡터 이미지 수출

ViewXY, ViewPolar 및 ViewSmith 도 .WMF, .EMF 및 .SVG 형식으로 수출 가능하다. View3D 및 ViewPie3D 은 아직 지원 안된다. **SaveToFile** 또는 **SaveToStream** 메소드를 선택 벡터 파일 형식과 함께 사용하라.

주의! 벡터 출력은 간단화 되었고 복잡 포인트 스타일 등 모든 디테일이 단순한 색 과 모양으로 표현될 수 있다. 벡터 산출에 비트맵 요소들이 있을 수 있다.

13.1.3 클립보드에 복사

CopyToClipboard(…) 를 불러 차트를 클립보드에 복사할 수 있다. ViewXY, ViewPolar 및 ViewSmith 은 **CopyToClipboardAsEmf()** 메소드로 벡터 형식으로 복사할 수 있다.

13.1.4 바이트 배열로 캡쳐

차트는 **CaptureToByteArray** 메소드가 있다. 이는 빠른 원시 이미지 데이터를 외부 구성 요소로 복사 또는 데이터를 추가 프로세싱 위해서다.

사용

```
int width;
int height;
byte[] aData = _chart.CaptureToByteArray(out width, out height);

Bitmap bitmap = new Bitmap(width, height,
System.Drawing.Imaging.PixelFormat.Format32bppArgb);

System.Drawing.Imaging.Bitmapdata bitmapData = bitmap.LockBits(new
System.Drawing.Rectangle(0, 0, width, height),
System.Drawing.Imaging.ImageLockMode.ReadOnly,
System.Drawing.Imaging.PixelFormat.Format32bppArgb);

IntPtr ipDst = bitmapData.Scan0;
int iRowByteCount = width * 4;
int iSrcIndex = 0;
for (int iY = 0; iY < height; iY++)
{
            Marshal.Copy(aData, iSrcIndex, ipDst, iRowByteCount);
            ipDst = new IntPtr(ipDst.ToInt64() + bitmapData.Stride);
            iSrcIndex += iRowByteCount;
}

bitmap.UnlockBits(bitmapData);
```

13.1.5 지속적 프레임 쓰기 위한 산출 스트림 설정

chart.OutputStream 속성을 이용해 차트가 렌더링 된 프레임을 쓸 스트림을 설정하라.

이 속성은 차트에서 지속적 프레임을 캡쳐하기 가장 빠른 방식으로 의도 되었다. 특별히 **Headless mode** 에 말이다 (23 장을 보라).

이 스트림은 원시 바이트 스트림이다. 각 픽셀은 4 바이트로 설명 된다, 채널당 1 바이트. 채너의 순서는 렌더러와 이의 설정에 따라 다르다.

GetLastOutputStreamFormat 및 **GetLastOutputStreamSize** 메소드들을 사용해 마지막으로 쓰여진 이미지의 형식 및 산출 크기를 알아 내라.

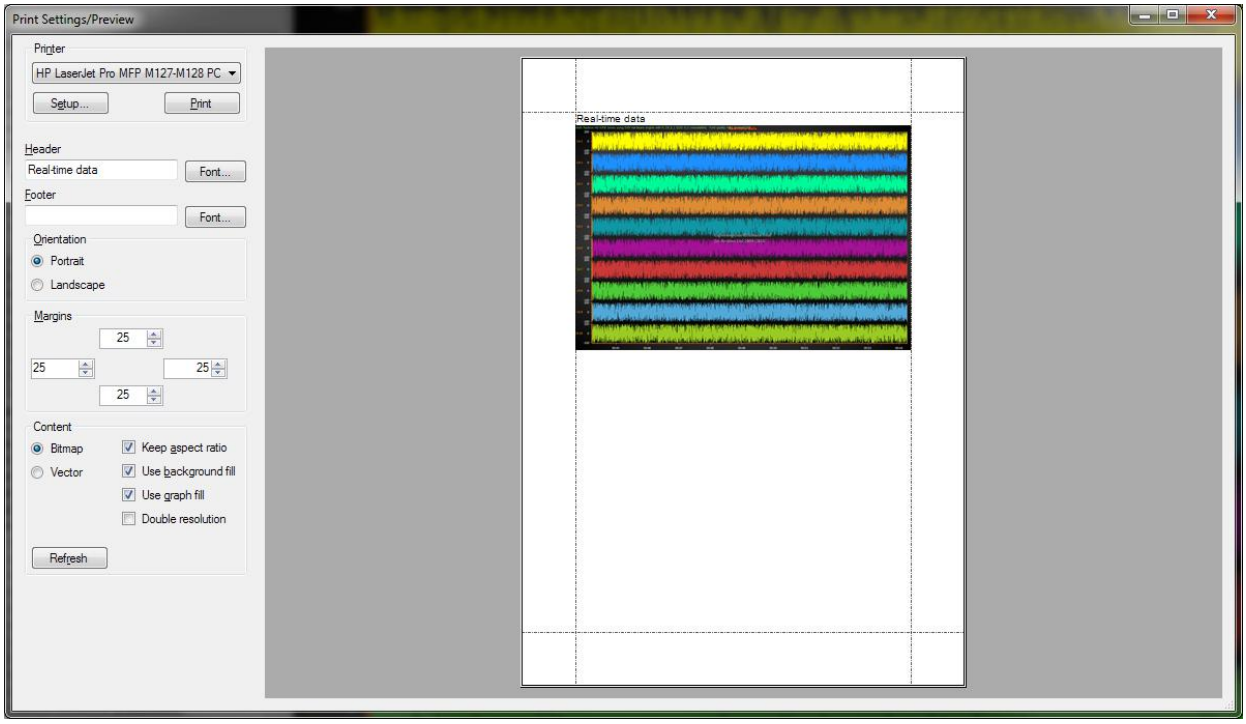
생산된 이미지 크기는 차트의 픽셀 크기와 같아야 한다.

주의! 라이트닝차트의 다른 속성들과는 다르게 스트림 설정은 차트의 제거와 함께 제거 되지 않는다.

주의! 다른 속성들과는 다르게 이 속성을 설정한다고 새로운 프레임을 렌더링 하지 않는다.

13.1.6 프린트

PrintPreview() 메소드를 불러 프린트 프리뷰 대화창을 열어라. 또는 **Print()** 로 기본 설정으로 바로 프린트를 하여라. **Print(...)** 를 불러 매뉴얼 설정으로 프린트 하라. ViewXY, ViewPolar 및 ViewSmith 에서 프린트는 벡터 프린트도 지원한다. 라스터 또는 벡터 형식을 매게 변수에 제공하여 **Print(...)** 메소드를 사용하라.



보기 14- 1. 프린트 미리보기 대화창.

14. 라이트닝차트 성능

14.1 제대로 된 API 판 선택하기

1.1 장에 설명 대로 차트 판을 선택하라. 꼭 필요할 때 배고는 시리즈 데이터 바인딩 기능은 사용하지 마라.

14.2 제대로 된 렌더링 옵션 설정

라이트닝차트의 DirectX9 렌더링 엔진이 어떤 어플레케이션에는 DirectX11 엔진보다 살짝 빠를 수 있지만 대체적으로 DirectX11 을 선호 렌더러로 두는게 좋은 선택이다. DirectX11 의 생김세 또한 훨씬 낫다.

글꼴 질 설정도 중요하다.

5.10 장을 보라.

14.3 차트 데이터 또는 속성 업데이트

각 속성 또는 시리즈 데이터 값 변화는 라이트닝차트 제어를 거두게 한다. 각 다시 그리기는 CPU 및 디스플레이 어댑터 오버헤드를 일으킨다. 하나 이상의속성이 프로그램적으로 동시에 변경되면, **BeginUpdate() 및 EndUpdate()** 사이 배치로 속성 변화를 만들어야 한다. **BeginUpdate()** 는 **EndUpdate()** 가 불릴 때까지 제어를 그만 그린다. 펜딩 **BeginUpdate()** 콜을 위한 내부 카운터가 있다. 같은 수의 **EndUpdate()** 콜에 도달하면 **EndUpdate()** 가 제어를 다시 그린다. 다음 예시는 컴퓨터에 부담을 최소화 하며 차트를 업데이트 할 수 있는지 보여준다.

```
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);
```

```
// 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
```

내부 카운터는 다음과 같은 네스팅 업데이트 사용을 허용한다.

```
void MainMethod()
{
    chart.BeginUpdate();
    chart.Title.Text = "My title";
    chart.ViewXY.XAxes[0].AxisColor = Colors.Red;

    UpdateSeriesColors();

    chart.EndUpdate();
    // Repaints only once.
}

private void buttonCreate_Click(object sender, EventArgs e)
{
    UpdateSeriesColors(); // Repaints only once
}

void UpdateSeriesColors()
{
   chart.BeginUpdate();

   foreach(PointLineSeries series in chart.ViewXY.PointLineSeries)
   {
            series.LineStyle.Color =  Color.Yellow;
   }

    chart.EndUpdate();
}
```

시리즈 데이터 업데이트는 데이터가 어떻게 저장 되었는지에 따른다. 배열 시리즈에는 **InvalidateData() 메소드**가 배열 내용을 업데이트 후 불려야 한다. 그러지 않고서는 UI 가 변경에 대해 알림을 받지 못한다.

WPF 반바인딩 및 바인딩에서 데이터 바인딩에 유용한 ObservableCollection 은 매 포인트 또는 포인트의 필드 마다 자신을 업데이트 한다. 이는 자동 알림 때문이다. 그럼으로 **InvalidateData()** 는 필요 없다. 하지만 ObservableCollections 는 배열 또는 목록에 비해 살짝 감소된 성능을 일으킨다. 이는 데이터 포인트의 수의 양이 많을 때 티가 난다.

14.4 라인 시리즈 팁

- 라인 시리즈를 사용할 때 어플리케이션에 적합하면 **SampleDataSeries** 를 사용하라. 이가 그리기 가장 빠르고 다른 라인 시리즈 종류만큼 메모리 공간이 많이 필요 없다. 사용 없션이 아니면 **FreeformPointLineSeries** 보다 **PointLineSeries** 를 선호하라.
- **PointsVisible** 속성을 거짓으로 설정하라. 데이터 포인트가 보이지 않아도 될 때다.
- 라인 넓이를 **LineStyle.Width** 속성으로 1 로 설정하라.
- **LineStyle.Pattern 을 Solid** 로 설정해 실선을 사용하라.
- 앤티 앨리어싱을 비활성화하라. **LineStyle.AntiAliasing 를 None** 으로 설정하고 차트의 **AntiAliasLevel** 을 0 으로 설정하라.
- 모든 마우스 대화식을 비활성화 하라. 시리즈의 **MouseInteraction** 설정을 거짓으로 설정하라. **chart.Options.MouseInteraction** 을 거짓으로 설정해 차트 전체의 마우스 대화식을 비활성화 하라.
- 필요 없으면 레전드 상자를 숨겨라. **ShowInLegendBox** 속성은 모든 시리즈를 위해 비활성화 가능하다. 한 차트에 시리즈가 여러개 있을 때 틀히 적용 된다.

```
chart.ViewXY.LegendBoxes[0].Visible = false;
chart.ViewXY.PointLineSeries[0].ShowInLegendBox = false;
```

14.5 강도 시리즈 팁

IntensityGridSeries, **IntensityMeshSeries** 에 적용 됨:

- 시리즈의 **Optimization** 속성을 **StaticData** 로 변경하라. 지속적으로 데이터가 업데이트 안되면 **DynamicData.** 데이터가 초당 여러번 변경 되면 더 좋은 선택이다.
- **Optimization**: **DynamicValuesData** 로 **Data** 배열의 **IntensityPoint** 구조의 **Value** 필드만 업데이트 하고 **InvalidateValuesDataOnly** 메소드를 불러 차트를 업데이트 하라. 그러면 시리즈의 형상이 재계산 되지 않아 업데이트가 훨씬 빠르다. 이것이 노드의 데이터 X 와 Y 값들이 같은 위치에 있는 어플리케이션에 사용 되게 의도 된 것이다. 열 이미징 솔류션 등이 있다.

IntensityGridSeries 에 적용됨

- 고 해상도 열 이미징 어플리케이션을 위해서는 **IntensityGridSeries** 를 위한 **PixelRendering** 을 활성화 하라.

- 빠르게 업데이트 되는 데이터 세트를 위해서는 **SetValuesData** 및 **SetColorsData** 메소드를 **Data** 속성 대신 사용해 메모리 공간 아끼고 성능을 향상 시켜라.

14.6 3D 직교 뷰 팁

OrthographicLegacy 대신 **View3D.Camera.Projection = Orthographic** 을 사용해 성능을 최대화 하라. 많은 시리즈 및 데이터가 있는 뷰를 줌 할 때 차이가 많이 난다 (7.4 장 보라).

14.7 3D 표면 시리즈 팁

SurfaceGridSeries3D, **SurfaceMeshSeries3D**, **WaterfallSeries3D** 에 적용됨

- 빛 반사 및 쉐이딩이 필요 없으면 **SuppressLighting** 를 거짓으로 설정해 빛을 비활성화 하라.
- 윤곽선이 사용 되었을 시, **FastColorZones** 또는 **FastPalettedZones** 를 **ColorLines** 또는 **PalettedColorLines** 대신 사용하라.

SurfaceGridSeries3D 에 적용됨

- 스크롤링 데이터 (3D 스펙트럼 또는 스펙토그램과 같은) 에는 **InsertRowBackAndScroll** 및 **InsertColumnBackAndScroll** 메소드를 이용해 데이터 및 축 값을 업데이트 하라.

14.8 지도 팁

ViewXY.Maps 에 적용됨

- **ViewXY.Maps.Optimization** 를 **CombinedLayers** 로 설정하라. X 와 Y 축 범위는 변하지 않지만 다른 데이터가 지도에 표현 되었을 경우에 말이다. 이것은 같은 버퍼 이미지에 지도 레이어 렌더링을 허용해 더욱 효율적인 렌더링이 된다.
- 지도 제목이 **IntensityGrid** 또는 **IntensityMesh** 시리즈 위에 표시 되어야 할 때 **ViewXY.Maps.Optimization** 를 **None** 으로 설정하라.

14.9 하드웨어

라이트닝차트 어플리케이션을 위한 최고 성능을 위해서는 컴퓨터 하드웨어가 강해야 한다. 많은 어플리케이션에는 디스플레이 어댑터 전원이 CPU 힘보다 중요하다. 최대한 최신 디스플레이 어댑터를 사용하라. DirectX 9.0c 레벨 디스플레이 어댑터도 괜찮다. 'c' 는 몇 효과에 필요한 DirectX 쉐더 모델 3 에서 온다.

GetRenderDeviceInfo() 메소드를 불러 어느 기능이 사용 디스플레이 어댑터에 지원 되지 않는지 알 수 있다. 특히 돌려지는 정보가 **FastVertexFormat** 가 지원되지 않는다고 한다면 성능에는 안좋은 것이다.

주의! 라이트닝차트는 GPU 하드웨어 가속 차트다. 좋은 GPU 없이는 성능이 최적화 경우보다 매우 낮을 수 있다. 다른 GPU 의 성능을 비교하기 좋은 자원은 **PassMark 의 비디오 카드 벤치마크** 이다 (http://www.videocardbenchmark.net/gpu_list.php). 다른 카드 보다 10 배 좋은 스코어를 받은 비디오 카드는 라이트닝차트에서도 10 배 빠를 수도 있지만 다른 하드웨어가 보틀넥이 되어 전체적인 새로고침 율은 대부분 거의 차이가 없다.

15. LightningChart 알림, 에러 및 예외 처리

버전 8.4 이상에는 라이트닝차트가 차트에서 사용자에게 **ChartMessage** 를 통해 메시지를 전달한다. 이 메시지들은 차트 성능, 잘못된 사용, 경고 및 에러에 대한 알림 등을 전달한다. **chart.ChartMessage** 이벤트가 메시지를 듣게끔 핸들러를 정의하라. 이 이벤트는 메시지의 정보를 갖고 있는 **ChartMessageInfo** 구성이 있다.

버전 8.4 이전에는 차트가 **ChartError** 이벤트를 통해 문자를 보냈다. **ChartMessage** 보다 훨씬 적은 정보가 들었었다. 사용자는 **ChartMessages** 대신에 **ChartError** 이벤트를 듣고 같은 베이스 정보를 받을 수 있지만 **ChartMessage** 를 사용하는 것을 추천한다.

ChartMessageInfo 의 **MessageSeverity** 속성이 메시지가 얼마가 심각한지 알려준다. 메시지는 심각도에 따라 필터 가능하다. 메시지의 심각도는 다음과 같다:

- **Debug** – 디버그 정보. 사용자에게 별로 흥미롭지 않고 아무런 조취가 필요 없다.
- **Information** – 차트의 잘못된 사용. 예를 들어 잘못된 속성 설정 사용으로 인해 차트 성능에 효과가 있었음. 사용자 액션은 주로 요구 되지 않는다.
- **Warning** – 차트의 잘못된 사용. 예를 들어 제거 된 객체 사용. 차트에 문제를 일으키고 성능 손실을 일으킴. 사용자 조취가 필요할 수도 있다.
- **RecoverableError** – 차트가 회복 가능한 에러가 일어났다. 사용자는 **ChartMessage** 이벤트 또는 메시지를 읽어야한다.

- **UnrecoverableError** – 회복하지 못한 차트 에러. 예외를 지시하는 걸수도 있다. 사용자는 **ChartMessage** 이벤트 또는 메시지를 들어야 한다.
- **Critical** – 심각한 에러가 일어났다. 익셉션으로 던져짐.

MessageType 속성이 메시지의 기본 유형을 설명하고 **Details** 속성이 더욱 세부 정보를 갖고 있다. 가능한 문자 종류는 라이트닝차트 네임스페이스에 위치 한 **MessageType** 에서 찾을 수 있다.

원하지 않는 메시지는 **chart.Options.ChartMessageMinimumLevel** 속성 값을 변경해 필터링 가능하다. 이 속성은 지정된 최소 레벨 이상 메시지만 이벤트 시스템을 통해 보내지게 한다. 이는 기본적으로 **MessageSeverity.Warning** 로 설정 되었다.

예외처리가 **ChartException** 객체로 던져진다. **ExceptionInfo** 구성을 갖고 있고 예외 처리 관련 세부 정보를 갖고 있다. 어떤 경우에는 차트가 렌더링 엔진 예외 등 다른 종류의 예외를 던질 수 있다. 사용자가 **MessageSeverity.Warning** 이상의 메시지만 보고 싶다면 **chart.Options.ThrowChartExceptions** 속성이 참으로 설정 되어야 한다 (기본적으로 거짓으로 설정).

ChartMessage 이벤트에 항상 구독해 차트 에러 및 가능 예외 처리에 대한 알림을 받는 것을 추천한다. 차트와 문제가 있고 지원이 필요하면 어플리케이션에 작동하는 메시지/예외 처리 핸들러가 있는 것을 확인하고 **ChartMessages** 를 기록해 지원 요청에 포함하라.

16. ChartManager component

ChartManager 제어를 이용해 여러 라이트닝차트 얼티밋 제어의 상호 운용을 지휘하는 데에 사용 가능하다. **ChartManager 제어**를 폼에 추가하라. 그 후 관리자 제어를 모든 라이트닝차트 얼티밋 제어의 **ChartManager** 속성에 할당하라.

16.1 차트 상호 운용, 드래그 드롭

ChartManager 는 윈폼에서 한 차트에서 다른 차트로 시리즈를 드래그 드롭이 허용 된다. WPF 에는 기술적인 이유로 사용할 수 없다.

시리즈는 **DisableDragToAnotherAxis** 속성이 있어 드래깅 활성화하기 위해 거짓으로 설정 되어야 한다. 기본적으로 참으로 설정 되어있다.

축은 **AllowSeriesDragDrop** 속성이 있어 이는 특별 축 위로 드래깅 방지하기 위해 거짓으로 설정 가능하다. 기본 값은 참이다.

마우스를 드래그 할 시리즈 위로 올리라. 왼쪽 마우슨 버튼을 눌러 드래그 하라

Y 축 위 드래그: 시리즈를 다른 차트의 Y 축 위로 드래그하고 버튼을 놓아라. 다른 차트가 시리즈를 소유하게 되고 시리즈가 해당 Y 축에 할당 된다. 이는 또한 시리즈의 첫 X 축도 할당한다.

X 축 위 드래그: 시리즈를 다른 차트의 X 축 위로 드래그하고 버튼을 놓아라. 다른 차트가 시리즈를 소유하게 되고 시리즈가 해당 X 축에 할당 된다. 이는 또한 시리즈의 첫 Y 축도 할당한다.

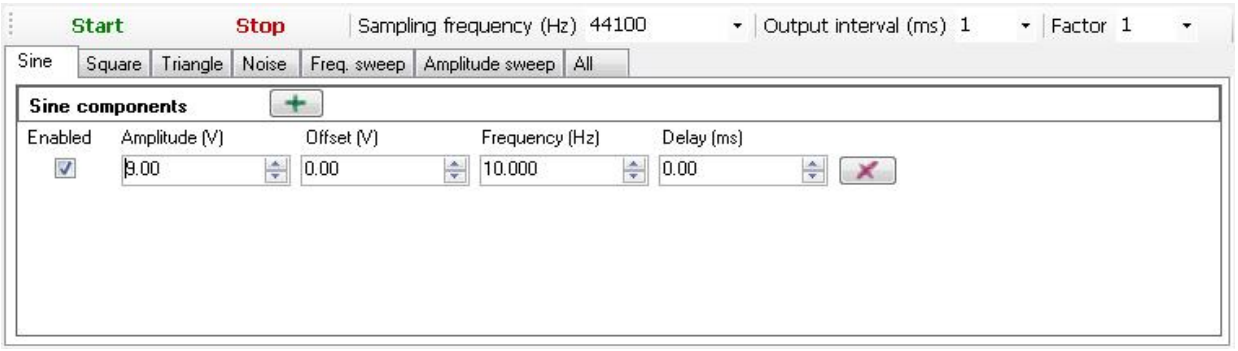
16.2 메모리 관리 향상

몇개의 극한 실시간 모니터링 어플리케이션에는 높은 CPU 부가량으로 실행 되면 .NET 쓰레기 컬렉터가 제대로 비사용 메모리를 비우지 못한다. 쓰레기 컬렉터가 한번에 모든 메모리를 비워 차트를 업데이트 할 때 티가 나게 얼거나 멈춘다. 차트가 더욱 부드럽게 업데이트 하게 만들기 위해 차트관리자의 **MemoryGarbageCollecting** 속성을 활성화 하라. 이것은 CPU 부가와 상관없이 따로 쓰레드를 이용해 메모리를 더욱 자주 비운다. 멀티 코어 프로세서를 사용하는 기계를 사용할 때 **MemoryGarbageCollecting** 를 추천한다. 이는 쓰레드 실행이 CPU 를 살짝 더욱 부가하기 때문이다.

17. SignalGenerator 구성 요소

데모 예시: 구역; 오실로스코프; SignalGenerator -> 스피커; 강도 지속 레이어, 시그널; 고속 데이터, 스택 축 (윈폼에만)

SignalGenerator 구성 요소는 실시간 시그널을 생성하는 데에 사용 가능하다. 이 시그널은 여러 웨이브형의 합으로 생성된다. 여러 **SignalGenerator** 구성요소를 마스터-노예 관계로 링크 가능하다. 이는 동기화 멀티 채널 출력을 생성하기 위함이다. **SignalGenerator** 는 시그널 모니터링 또는 라이트닝차트로 데이터 취득 소프트웨어를 개발할 때 유용하다.



보기 18- 1. 시그널 발생기 구성요소에 사인 페이지가 선택 되었다.

웨이브형은 다음과 같은 종류로 나뉜다: **Sine**, **Square**, **Triangle**, **Noise**, **Frequency sweep**, **및 Amplitude sweep.** 구성 요소에 해당 관련 탭 페이지를 볼 수 있다. 사인 페이지에서는 사인 웨이브형을 추가할 수 있다. 네모 및 세모 페이지에는 네모 및 세모 웨이브형을 각각 추가할 수 있다. 노이즈 페이지에서는 랜덤 소리 웨이브형을 생성할 수 있다. 주파수 및 앰프 스위프 페이지에서는 주파수 및 앰프 스위프를 추가할 수 있다. All 페이지에는 모든 웨이브형을 스택 뷰로 설정 가능하다.

17.1 샘플링 회수, 출력 간격 및 팩터

SamplingFrequency 은 초당 몇개의 시그널 포인트가 생성되는지 알려준다. 높은 샘플링 회수는 더욱 정확한 시그널을 생성하지만 이는 데이터 플로우 및 오버헤드 또한 상승한다. 높은 샘플링 회

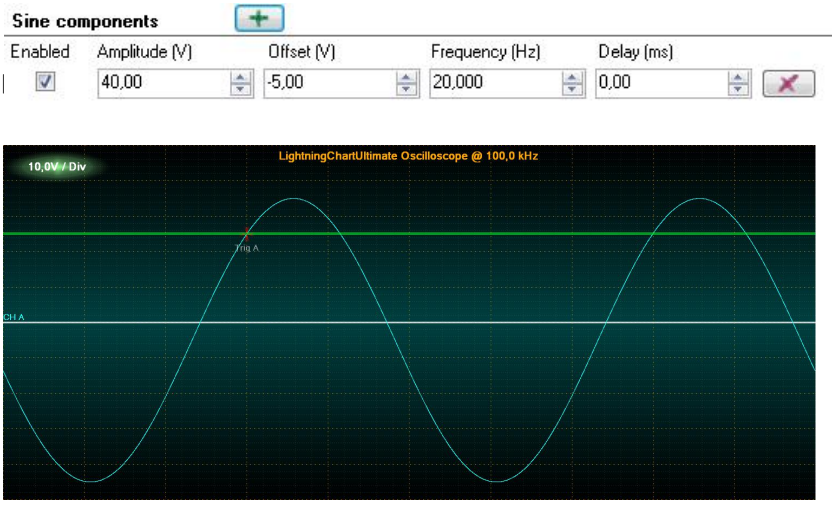
수와는 높은 주파수를 포함한 시그널을 표현할 수 있다. 샘플링 회수는 시그널 회수의 두배 이상이어야 한다. 나이키스트 주파수 이론을 충족하기 위함이다.

OutputInterval 은 계산된 출력 샘플의 선호 간격을 미리 초로 설정한다. 예를 들어 **OutputInterval** 이 100 으로 설정 되면 샘플 번들을 초당 10 번 받게 된다, 이는 100ms 기간을 얘기한다. 낮은 값을 이용하는 것은 더욱 부드러운 실시간 모니터링 출력을 생성한다. **OutputInterval** 은 별로 정확하지 않으며 컴퓨터 부가에 따라 다를 수 있다. 기간이 예상보다 길면 출력 데이터 스트림은 자동으로 더 많은 샘플을 생성한다. 데이터 스트림은 높은 데이터 율 및 무거운 컴퓨터 오버헤드로도 모양을 잡는다.

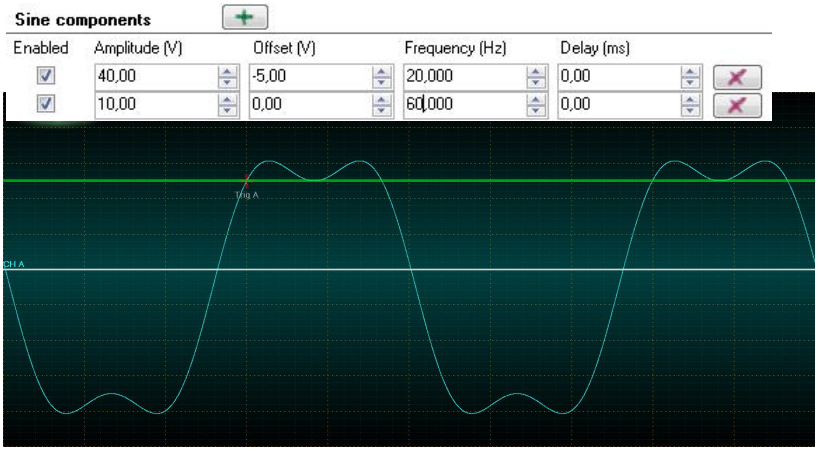
Factor 는 출력 샘플을 선택 값으로 곱한다. 예를 들어 V 시그널 대신 mV 시그널을 생성하기 위해 **Factor** 를 1E-3 로 설정하라.

17.2 사인 웨이브형

사인 웨이브형은 **Amplitude**, **Offset**, **Frequency** 및 **DelayMs** 매개 변수로 구성된다. **Amplitude** 는 0 레벨부터 최대 전압 차이다. 총 범위는 양극형이다. 피크-투-피크 값은 2* **Amplitude** 이다. **Offset** 은 시그널에 추가된 DC 레벨이다. 다른 말로는 양수 값은 시그널을 위로 움직이고 음수 값은 값 범위에서 아래로 움직인다. **Frequency** 는 시그널 사이클 수를 헤르츠로 말한다. 초당 사이클이 1 헤르츠의 주파수다. **DelayMs** 는 시그널을 미리초로 딜레이를 만든다.



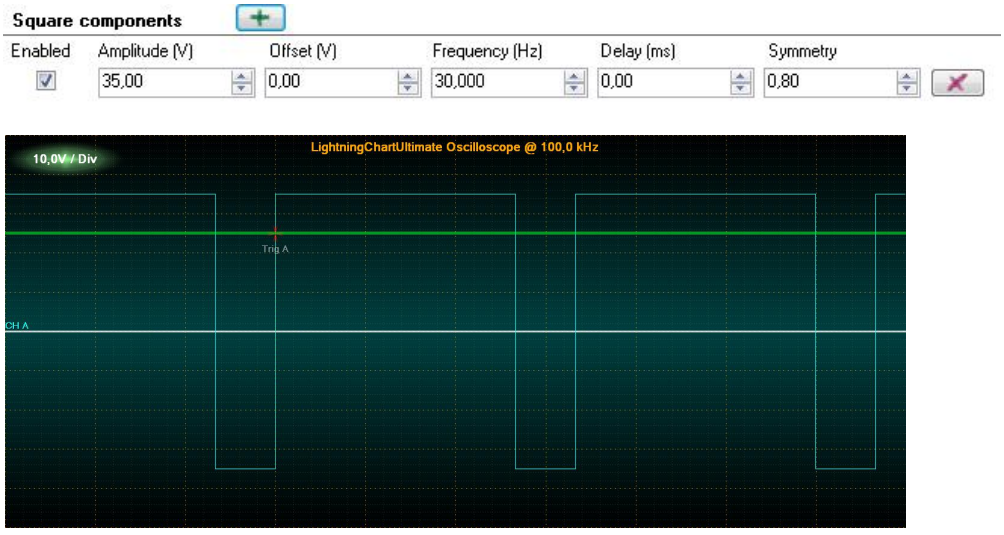
보기 18- 2. 위의 설정으로 생성된 간단 사인 웨이브형.



보기 18- 3. 위 설정으로 두 사인 웨이브형의 시그널.

17.3 네모 웨이브형

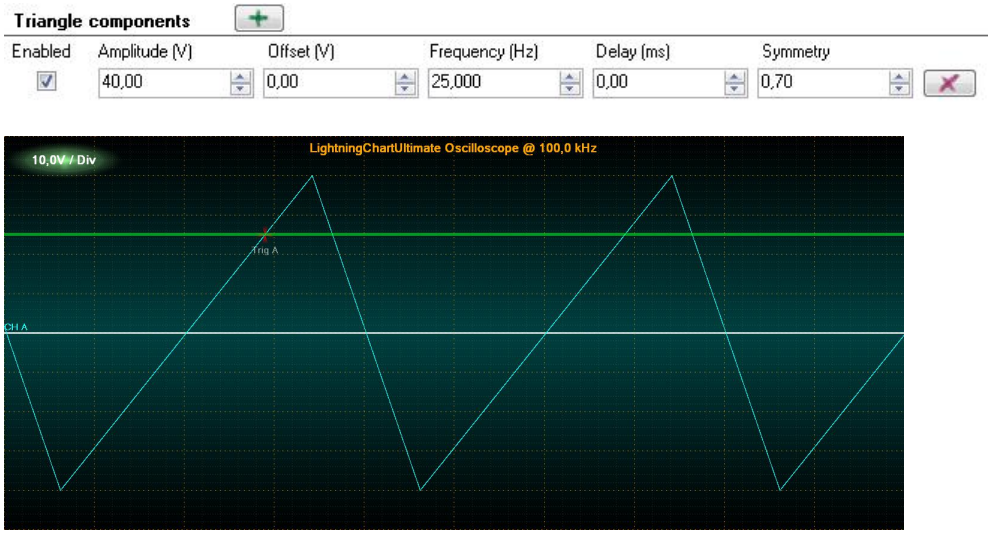
네모 웨이브형은 사인 웨이브형에 비해 한 매개 변수가 더 있다, **Symmetry. Symmetry** 의 범위는 0 에서 1 이다. **Symmetry** 는 사이클 기간 관련 얼마나 오래동안 시그널이 높은 상태에 유지하는지를 말해준다. 0.5 의 값은 저 및 고 시그널 상태가 같은 길이다.



보기 18- 4. Symmetry 설정이 0.8 로 된 한 네모 웨이브형 시그널.

17.4 세모 웨이브형

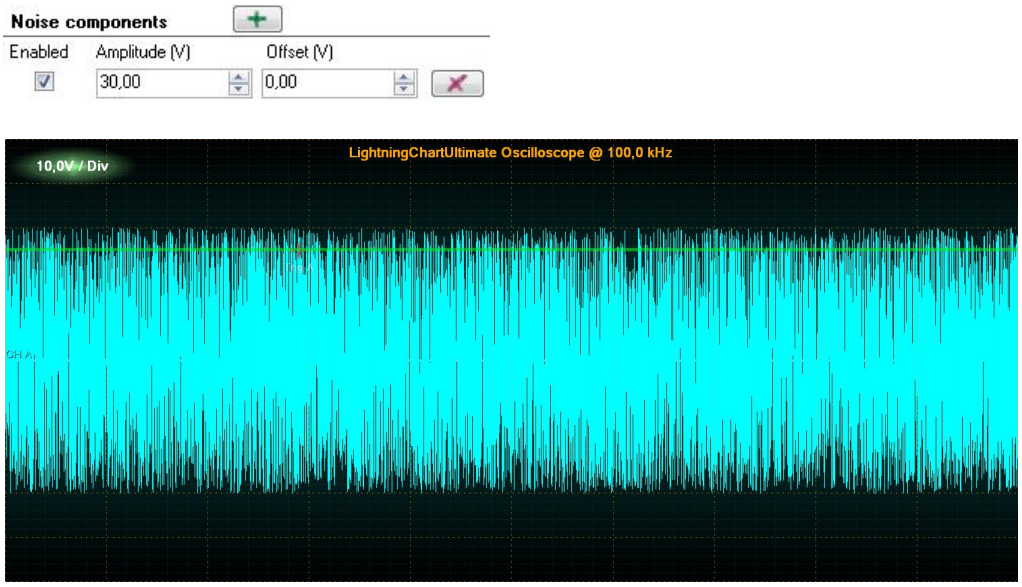
세모 웨이브형도 **Symmetry** 매개 변수를 갖고 있다. 이는 세모의 기울기를 제어한다. 0.5 의 값은 대칭 되는 세모의 값이다. 0.5 이하의 값의 세모는 왼쪽으로 기울고 이상의 값은 오른쪽으로 기운다.



보기 18- 5. Symmetry 0.7 로 설정 된 한 세모 웨이브형 시그널

17.5 노이즈 웨이브형

노이즈 웨이브형은 랜덤으로 생성된 시그널이다. 포인트는 **–Amplitude** 와 **+Amplitude** 사이 랜덤으로 지정된다.



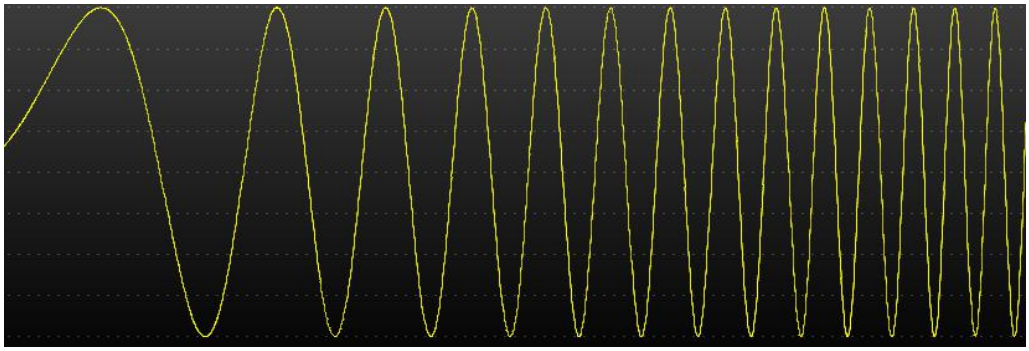
보기 18- 6. -30 과 30 사이 앰프의 랜덤 데이터 포인트를 생성하는 노이즈 웨이브형

17.6 주파수 스위프

주파수 사인 스위프는 주어진 시간 동안 일정한 앰프로 주파수 1 에서 2 로 실행된다.

FrequencyFrom 을 사용해 주파수 시작을 설정하고 **FrequencyTo** 로 주파수 끝을 설정하라.

Amplitude 로 일정 앰프를 설정하고 **DurationMs** 로 미리초로 주어진 기간을 설정하라.

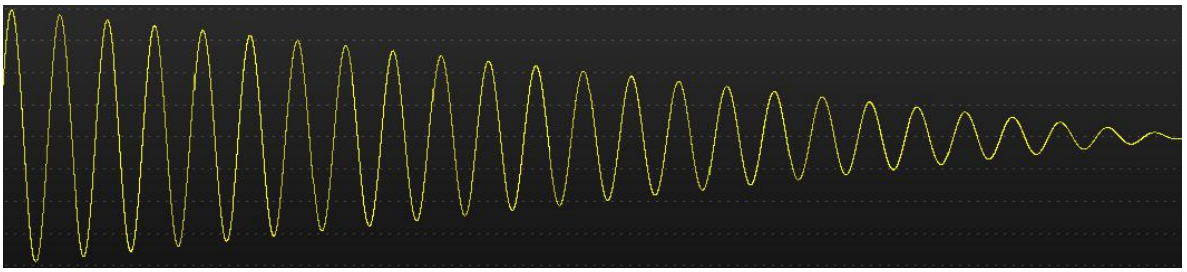


보기 18- 7. 주파수 스위프.

17.7 앰프 스위프

앰프 사인 스위프는 앰프 1 에서 2 로 주어진 시간 기간 동안 일정한 회수로 실행된다.

AmplitudeFrom 으로 앰프 시작을 설정하고 **AmplitudeTo** 고 앰프의 끝을 설정하라, **Frequency** 로 일정 회수를 설정하고 **DurationMs** 으로 미리 초로 주어진 기간을 설정하라.



보기 18- 8. 앰프 스위프.

17.8 시작 및 멈추기

제너레이터를 시작 버튼을 누르거나 **Start** 메소드를 불러 시작하라. 스톱 버튼을 누르거나 **StopRequest** 를 불러 멈추라. 멈춤이 완료되면 **Stopped** 이벤트가 실행된다.

17.9 주인-노예 구성 멀티 채널 제너레이터

여러 **SignalGenerator** 구성 요소가 연결되어 동기화 된 멀티 채널 출력을 생성할 수 있다.

MasterGenerator 는 샘플링 회수를 제어하고 모든 발성기의 시작, 멈춤 및 출력을 제어한다. 이는 출력 데이터 스트림의 첫 채널을 생성한다.

노예 발성기는 **MasterGenerator** 속성을 할당함으로 주인 발성기에 연결 된다. 시그널 웨이브형을 자유롭게 정의하라. 노예 발성기는 마스터 발성기로 시작과 멈춰진다. 출력 데이터 스트림 채널 인덱스를 연결 순서대로 받는다. 노예 발성기는 마스터 발성기 실행 전 연결 되어야 한다.

17.10 출력 데이터 스트림

출력 데이터 스트림은 2 차원적 배열로 구성 되었다. 이는 **DataGenerated** 이벤트 핸들러를 통해 얻게 된다. 주로 이벤트는 매 **OutputInterval** 이후 올려진다.

이벤트 핸들러는 샘플 배열의 참조를 얻게 되고 이 기간 동안 받게 되는 첫 샘플 번들의 타임 스탬프를 얻게 된다. 샘플 배열의 첫 차원은 채널을 표현하고 두번째는 각 채널의 샘플을 표현한다. 모든 채널은 같은 샘플 카운트를 갖고 있다.

DataGenerated 이벤트 올리기:

```
m_signalGenerator.DataGenerated += m_signalGenerator_DataGenerated;

private void m_signalGenerator_DataGenerated(DataGeneratedEventArgs args)
```

```
{
   // Event code
{
```

To investigate the channel count of the data stream, get the length of first dimension:

```
channelCount = args.Samples.Length;
```

To get the sample count of a channel:

```
sampleBundleCount = args.Samples[0].Length;
```

The following code will demonstrate how to forward the output data directly to **SampleDataSeries** list of LightningChart while updating real-time monitoring scroll position.

```
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();
 }
```

args.Samples[0]로 마스터 발성기의 데이터를 접근할 수 있다. **args.Samples[1]**은 첫 노예 발성기 데이터에 접근을 허용하고 **args.Samples[2]**는 두번째 노예 등에 접근을 허용한다.

18. SignalReader 구성 요소

데모 예시: 시그널 리더; 웨이브형 및 스펙트럼; 웨이브형, 3D 스펙토그램; 오디오 L+R, 구역, 스펙토그램

SignalReader 구성 요소는 시그널 소스 파일에서 데이터를 읽고 선택 레이트로 재생 가능하다. **SignalReader** 출력 데이터 스트림 유형은 **SignalGenerator** 과 비슷하다 (18.10 을 보라).

SignalReader 구성 요소는 현재 wav 및 sid 유형을 지원한다.

18.1 주요 속성

FileName 은 열을 파일을 정의한다. 예를 들어: `"c:\\wavedata\\audioclip1.wav"`

Factor 는 출력 팩터를 설정한다. 원시 시그널 샘플이 이 값으로 곱해진다.

OutputInterval 은 **SignalGenerator** 의 속성과 비슷하다 (18.1 장을 보라).

IsLooping 은 파일 읽기가 파일의 끝에 도달 했을 때 파일의 첫 부분으로 돌아가게 허용한다.

파일이 열린 후 다음과 같은 속성들로 파일의 정보를 얻을 수 있다:

ChannelCount: 파일의 채널 카운트.

SamplingFrequency: Hz 로 샘플링 회수.

FileSize: 바이트로 파일 크기.

Length: 각 채널의 샘플 카운트. 모든 시그널 파일 형식에 정확하지 않을 수 있다.

IsReaderEnabled: 구성 요소가 데이터를 시작하고 읽고 있는지에 대한 상태. Looping 이 거짓으로 설정되고 파일이 끝에 도달했으면 **IsReaderEnabled** 가 거짓으로 변경된다.

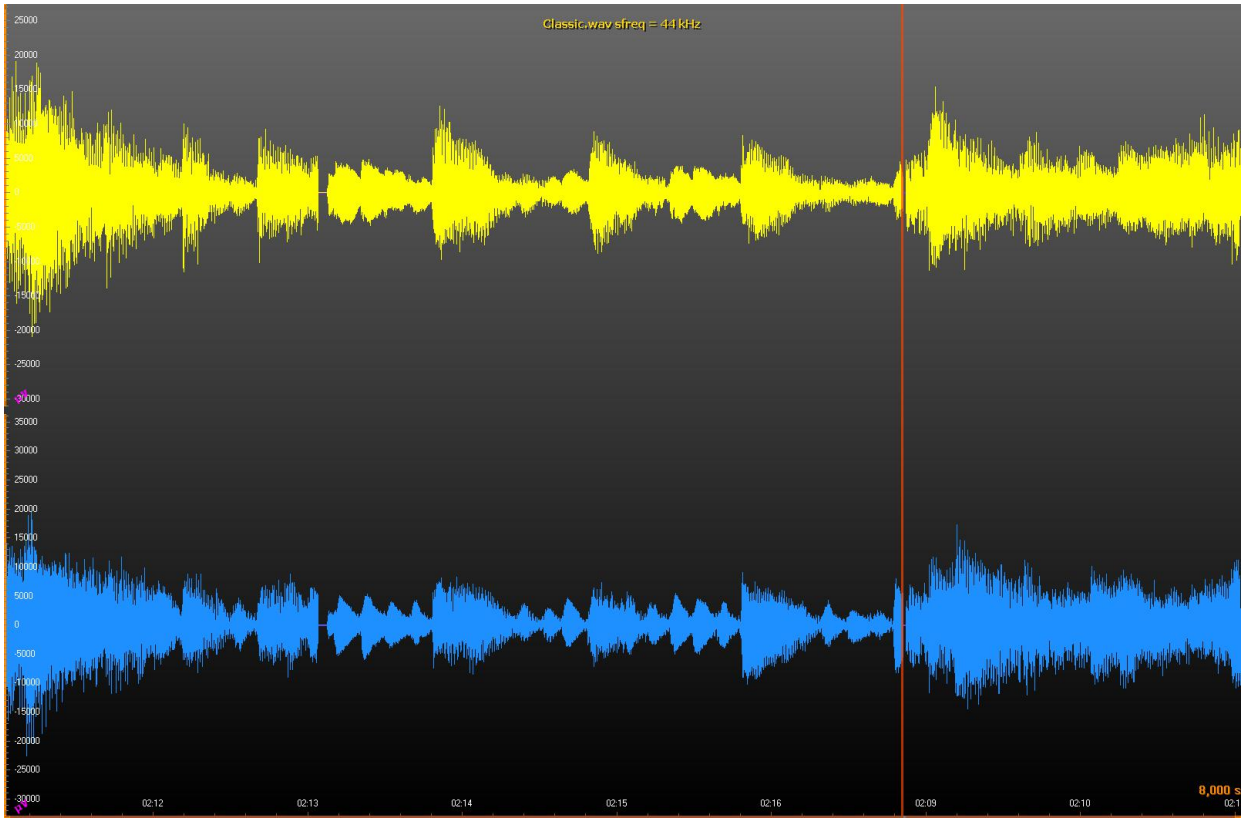
18.2 재생을 위해 빠르게 파일 열기

OpenFile(…) 메소드에 파일 이름을 제공해 불러라. 파일 이름은 지원된 형식중 하나의 확장이 있어야 한다. 그 후 **Start()** 메소드를 불러라.

```
signalReader.OpenFile("c:\\wavedata\\audioclip1.wav");
signalReader.Start();
```

PCM-형식의 WAV 파일이 재생된다.

StopRequest() 메소드를 불러 재생을 멈출 수 있다.



보기 19- 1. SignalReader 가 wav 파일을 읽고 라이트닝차트의 SampleDataSeries 가 시그널을 그린다. 커서 라인이 현재 읽기 위치 및 X-축 스크롤 위치를 표시하기 위해 사용된다.

19. AudioInput 구성 요소

데모 예시: 오디오 입력, 웨이브형; 오디오 입력, 스펙토그램

AudioInput 구성 요소는 사용자가 윈도우 녹음 기계의 시그널 System.Double 값으로 잡게 해준다. 이 값들은 라이트닝차트에 렌더링 되어 오디오출력 구성 요소로 보내어 파일에 저장한다.

19.1 속성

BitsPerSample – 샘플당 몇 비트가 할당 됐는지를 얻거나 설정한다. 지원된 값은 8 또는 16 이다. 다른 값을 입력 하면 16 이 사용된다. **IsInputEnabled** 가 거짓일 때 설정 가능하다.

IsInputEnabled – 경우의 상태를 얻거나 설정한다 (시작 또는 멈춘다). 이 속성을 참으로 설정하는 것은 Start 메소드를 부르는 것과 똑같다. 거짓은 Stop 메소드를 부르는 것과 똑같다.

IsStereo – 두 채널 (스테리오) 또는 한개 (모노)를 사용할지 얻거나 설정한다. **IsInputEnabled** 가 false 일때 설정 가능하다.

LicenseKey – 라이센스 키를 보통 또는 암호화 된 형식으로 얻거나 설정한다.

RecordingDevice – 현재 녹음 기계를 얻거나 설정한다. 이는 **IsInputEnabled** 가 false 일 때 설정 가능하다. 이 속성을 null 로 설정함으로 윈도우 기본 녹음 기계가 사용 된다.

SamplesPerSecond – 샘플링 회수를 얻거나 설정한다. **IsInputEnabled** 가 false 일 때 설정 가능하다.

ThreadInvoking – 메인 UI 쓰레드에 경우의 이벤트가 자동 동기화 되는지를 얻거나 설정한다. 이는 부르는 쪽에서 **Control.Invoke** 메소드를 부를 필요를 제거한다.

Volume – 볼륨을 얻거나 설정한다. 범위는 0 에서 100 이다. **IsInputEnabled** 가 false 일 때 설정 가능하다.

19.2 메소드

GetRecordingDevices – 이 정적 메소드를 이용해 사용 가능한 윈도우 녹음 장치의 목록을 불러와라.

RequestStop – **AudioInput** 경우를 멈추라고 신호 보낸다. 이 메소드를 종료후 바로 멈추지 않는다. **Stopped** 이벤트에 구독함으로 부르는 사람은 모든 것이 멈췄을 때 통지 받는다.

Start – 선택 녹음 장치에서 오디오 읽기를 시작한다. 내부 쓰레드가 시작하기 전에 Started 이벤트가 트리거 된다.

19.3 이벤트

DataGenerated – 새로운 오디오 데이터 세트가 생성 되었을 때 일어난다. 데이터 및 이의 첫 샘플의 타임 스탬프는 **DataGeneratedEventArgs** 객체가 매게 변수로 제공 되었을 때 읽을 수 있다.

Started – 오디오 입력이 시작 되었을 때 실행 된다. **StartedEventArgs** 객체가 매개 변수로 제공 되었을 때 세개의 공공 필드를 갖고 있다: **BitsPerSample**, **ChannelCount** 및 **SamplesPerSecond.**

Stopped – 오디오 입력이 멈춰졌을 때 일어난다.

19.4 사용 (윈폼)

이 장은 윈폼의 **AudioInput** 클라스 버전의 사용에 대해 설명한다. WPF 버전은 20.5 장에서 다룰 것이다.

19.4.1 생성

새로운 **AudioInput** 경우를 소스 코드에 수동으로 생성하거나 비주얼 스튜디오의 툴박스에서 표로 그래그 드롭으로 생성하라.

GUI 를 보여줄 필요가 없거든 (자신의 GUI 를 사용 중이거나 **AudioInput** 객체를 소스 코드에서 제어하고 있을 때) **Visible** 속성을 거짓으로 설정하라. **Parent** 속성은 항상 설정하는 것을 추천한다. 이는 부모 제어가 제거 되면 **AudioInput** 경우가 자동으로 제거 된다. 부모가 없으면

AudioInput 경우가 더 이상 필요 없을 때 **Dispose** 메소드를 불러야 한다. 비주얼 스튜디오의 툴박스를 통해 새로운 **AudioInput** 경우가 생성되면 **Parent** 속성이 자동으로 설정 된다.

AudioInput 경우가 윈도우 기재에서 찾기 보다 명백한 라이센스 키를 사용하기 위해 **LicenseKey** 속성을 설정하는 것을 추천한다. 트라이얼 버전 또는 라이센스가 사용 됐다면 **LicenseKey** 속성은 기본 값으로 둬도 된다.

19.4.2 이벤트 핸들링

AudioInput 경우에서 새로운 샘플을 얻기 위해서 사용자는 적어도 **DataGenerated** 이벤트에 구독 해야 한다. **DataGenerated** 이벤트가 트리거 되면 새로운 샘플 및 첫 샘플의 타임 스탬프는 **DataGeneratedEventArgs** 에서 매개 변수로 제공 된다.

Started 이벤트에 구독하여 **AudioInput** 경우가 오디오 샘플링 작업을 언제 시작하는지 알 수 있다. **StartedEventArgs** 객체가 **AudioInput** 의 매개 변수로 정보를 제공한다. 예를 들어 샘플당 비트 수, 스트림 오디오가 모노인지 스테리오인지, 또는 초당 몇개의 샘플이 생성되는지 등을 알 수 있다.

Stopped 이벤트에 구독하여 **AudioInput** 경우가 언제 멈췄는지를 알수 있다. 이 이벤트는 매개 변수가 없고 사용자에게 모든게 언제 멈췄는지 알리는 용도 밖에 없다.

19.4.3 구성

ThreadInvoking = true 로 설정해 **AudioInput** 경우를 자동으로 메인 UI 쓰레드에 이벤트를 동기화 하게 하는 동시에 **AudioInput** 경우가 유효 부모 제어가 있는지를 확인한다. **ThreadInvoking** 은 기본으로 false 로 설정 되었으니 GUI 를 **DataGenerated** 이벤트 핸들러에 업데이트 할 때 **Control.Invoke** 메소드를 부르는 것을 잊지 마라.

RecordingDevice 속성을 설정하여 기본 장치 외 기타 윈도우 녹을 장치의 사용을 허용한다. 사용 가능한 모든 녹음 장치를 **AudioInput** 의 정적 메소드 **GetRecordingDevices** 로 불러오라.

Volume 속성으로 볼륨을 제어 가능하다. 유효 값은 0 에서 100 이며 0 은 음소거고 100 은 최대 볼륨이다. 볼륨은 **AudioInput** 경우가 활성화 되었을 때도 설정 가능하다 (예를 들어 샘플 생성 할 때).

SamplesPerSecond 속성을 설정해서 기본과 다른 샘플링 율을 사용하라 (기본율은 44100 Hz). **AudioInput** 경우가 활성화 되었을 때 이 속성을 설정 하는 것은 아무 효과가 없다.

스테레오 (기본 설정) 대신 모노 오디오를 사용하기 위해서는 **IsStereo** 를 거짓으로 설정하라. **AudioInput** 경우가 활성화 되었을 때 이 속성을 설정 하는 것은 아무 효과가 없다.

16 (기본 설정) 보다 샘플당 8 비트가 선호되면 **BitsPerSample** 속성을 8 로 설정하라. 유요 값은 8 과 16 (기본 설정) 이다. 이 제한은 PCM 웨이브형에서 온다. **AudioInput** 경우가 활성화 되었을 때 이 속성을 설정 하는 것은 아무 효과가 없다.

19.4.4 시작

AudioInput 을 시작하기 위해 **IsInputEnabled** 속성을 참으로 설정하거나 **Start** 메소드를 불러라. **DataGenerated** 이벤트가 새로운 오디오 샘플을 생성한다. 이는 라이트닝차트 얼티밋 경우를 이용해 렌더링 가능하다.

19.4.5 멈추기

AudioInput 경우를 멈추기 위해 **IsInputEnabled** 를 거짓으로 설정하거나 **RequestStop** 메소드를 불러라. **RequestStop** 메소드는 즉시 멈추지 않는다. 대신 이는 **AudioInput** 경우를 최대한 빨리 멈추라고 신호를 보낸다. **Stopped** 이벤트에 구독해 **AudioInput** 경우가 멈췄는지 알 수 있다.

19.5 사용 (WPF)

이 장은 WPF 버전의 **AudioInput** 클라스의 사용을 설명한다. WPF 버전의 **AudioInput** 은 윈폼의 버전과 대부분 똑같이 작용한다. 하지만 WPF 사용자가 유의해야 할 몇가지가 있다.

19.5.1 생성

새로운 **AudioInput** 경우를 코드-비하인드로 수동으로 생성하거나 비주얼 스튜디오의 툴박스에서 창으로 드래그 드롭해서 생성하라.

GUI 를 보여줄 필요가 없으면 (개인 GUI 또는 **AudioInput** 객체가 소스코드에서 제어 될 때) **Arction.WPF.SignalTools** 네임스페이스에서 **AudioInput** 를 이용하라. 이 크라스는 FrameworkElement 에서 오며 이의 모든 속성은 바인딩 가능하다. 편리함을 위해 라이트닝차트 .NET SDK 를 설치 후 **Arction.WPF.SignalTools.AudioInput** 이 비주얼 스튜디오 툴박스에서도 찾을 수 있다. 이는 창으로 드래그 드롭 후 SAML 코드로 필요할 때 움직여 사용 가능하다. 필요한 XML 네임스페이스는 이런 방식으로 자동으로 추가된다.

AudioInput 을 위한 바로 사용 가능한 GUI 가 있다. 이는 **Arction.WPF.SignalTools.GUI** 네임스페이스에서 찾을 수 있다. 비주얼 스튜디오의 툴박스에서도 LightningChart® .NET SDK 설치 후 찾을 수 있다. 이는 **Arction.WPF.SignalTools.AudioInput** 클라스만을 위한 GUI 이지만 **Arction.WPF.SignalTools.AudioInput** 클라스의 경우가 있어 Input 속성으로 접근 가능하다. 다른 말로는 **Arction.WPF.SignalTools.AudioInput** 경우를 따로 생성할 필요가 없다는 말이다.

AudioInput 경우가 윈도우 기재에서 찾기 보다 명백한 라이센스 키를 사용하기 위해 **LicenseKey** 속성을 설정하는 것을 추천한다. 트라이얼 버전 또는 라이센스가 사용 됐다면 **LicenseKey** 속성은 기본 값으로 둬도 된다.

20. AudioOutput 구성 요소

데모 예시: 시그널리더 -> 스피커; 시그널생성기 -> 스피커

AudioOutput 구성 요소는 사용자가 System.Double 시그널 데이터를 스피커로 재생 또는 음향 장치 라인 아웃 연결 오디오 스트림으로 변경하는 것을 허용한다.

20.1 속성

Balance – 오디오 재생 밸런스를 얻거나 설정한다. 유효 값은 -100 에서 100. -100 은 오디오를 왼쪽 스피커에서만 재생. 0 은 두 스피커 모두 오디오 출력 함. 100 은 오디오를 오른쪽 스피커로만 재생.

BitsPerSample – 샘플당 몇 비트가 할당 되었는지를 얻거나 설정한다. 지원되는 값은 8 과 16 이다. 다른 값이 사용되면 16 이 대신 사용된다. **IsOutputEnabled** 가 거짓일 때 설정 가능하다.

IsOutputEnabled – 이 경우의 상태를 얻거나 설정한다 (시작하거나 멈춘다). 이 속성을 참으로 설정하는 것은 Start 메소드를 부르는 것과 같고 false 로 설정은 Stop 메소드를 부르는 것과 같다.

IsStereo – 채널 두개 (스트레오) 또는 한개 (모노)를 사용할지 알거나 설정한다. **IsOutputEnabled** 가 거짓일 때 설정 가능하다.

LicenseKey – 라이센스 키 스트링을 보통 또는 암호화 형식으로 얻거나 설정된다.

PlaybackDevice – 현재 재생 장비를 얻거나 설정한다. **IsOutputEnabled** 가 거짓일 때 설정 가능하다. 이 속성은 null 로 설정하면 윈도우 기본 재생 장비가 사용된다.

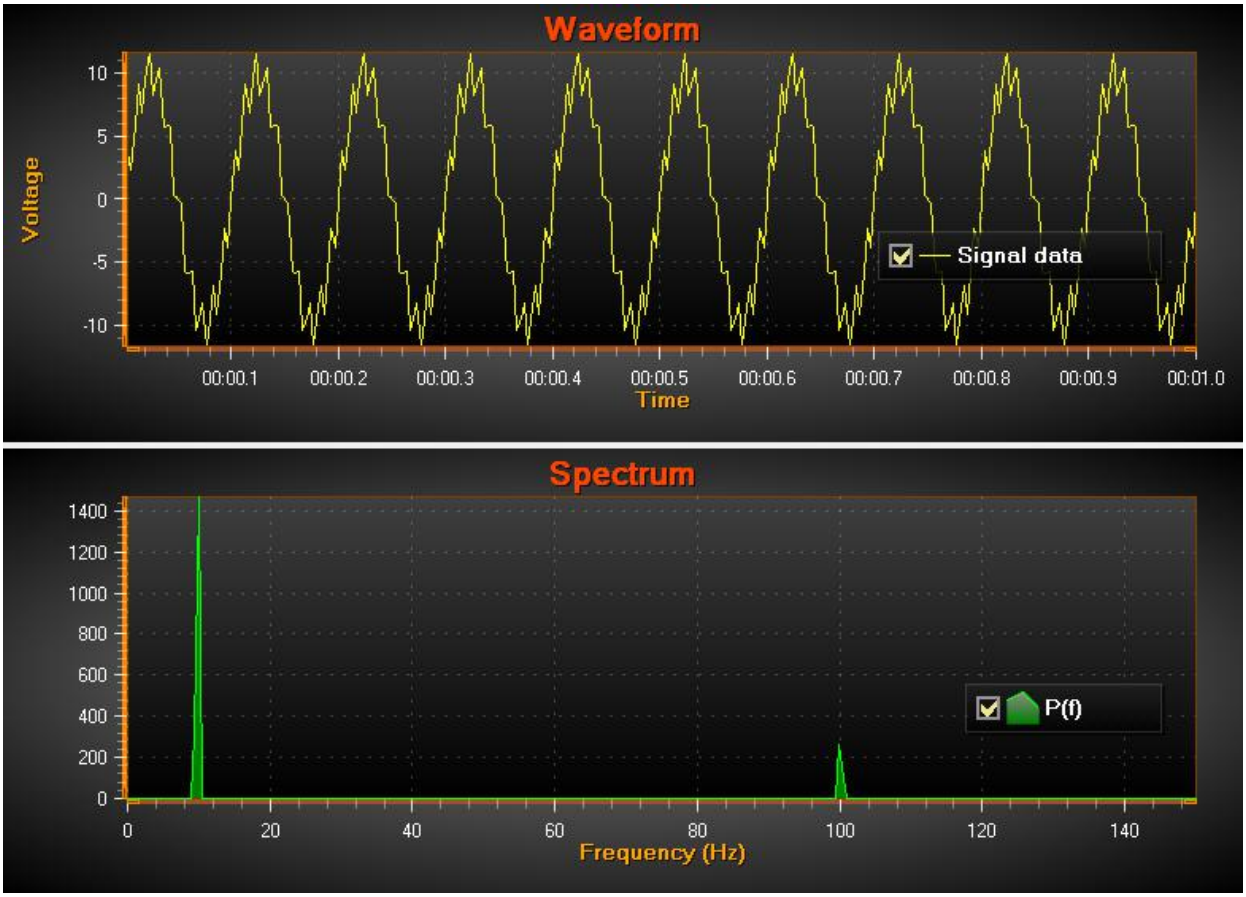
SamplesPerSecond – 샘플링 회수를 얻거나 설정한다. **IsOutputEnabled** 가 거짓일 때 설정 가능하다.

Volume – 볼륨을 얻거나 설정한다 (0-100). **IsOutputEnabled** 가 거짓일 때 설정 가능하다.

21. SpectrumCalculator 구성 요소

데모 예시: 웨이브형 및 스펙트럼

SpectrumCalculator 구성 요소는 시간 도메인 및 회수 도메인 사이 변경을 허용한다.



보기 22-1. 소스 시그널 데이터 (위) 가 회수 도메인 (아래)로 변경 된 예시다. 시그널 샘플링 회수 = 300 Hz 이다. 회수 스케일은 고로 300/2 = 150 Hz 이다. 강한 사인 베이스라인은 10Hz (초당 10 사이클)이다. 작은 100Hz 의 시그널은 노이즈로 추가 됐다. 파워 스펙트럼에서 두 스파이크를 찾을 수 있다.

다음과 같은 공공 메소드가 사용 가능하다:

- **CalculateForward**(double[] samples, out double[] fftData**) –** 타임 도메인 시그널 데이터를 회수 도메인으로 FFT 를 이용해 변경한다. 출력 fftData 는 음수 값도 포함한다. 입력 및 출력 데이터

배열은 같은 길이여야 한다. 길이는 데이터의 해상도다. 0 Hz 에서 샘플링 회수 / 2 로 출력 값 사이 같은 회수 간격이 있다.

- **CalculateForward**(float[] samples, out float[] fftData) – 이전 메소드와 비슷하지만 싱글 정확 플로트 프인트 값을 위해 사용된다.
- **CalculateBackward**(double[] fftData, out double[] samples) – 회수 도메인을 타임 도메인으로 변경한다. FFT 데이터에서 시그널 샘플을 생성한다. 샘플 카운트가 fftData 길이와 같다.
- **CalculateBackward**(float[] fftData, out float[] samples) – 이전 메소드와 비슷하지만 싱글 정확 플로트 포인트 값을 위해 사용된다.

- **PowerSpectrum**(double[] samples, out double[] fftData) – 시그널 데이터의 파워 스펙트럼을 계산한다. **CalculateForward** 와 같지만 절대 출력 값을 사용한다.

- **PowerSpectrum**(float[] samples, out float[] fftData) – 이전 메소드와 비슷하지만 싱글 정확 플로트 포인트 값을 위해 사용된다.

- **PowerSpectrumOverlapped**(double[] samples, int fftWindowLength, double overlapPercent, out double[] fftData, out int processedSampleCount) – 계산 창을 소스 시그널 샘플 데이터 내로 움직여 파워 스펙트럼을 계산한다. 시그널 데이터는 주어진 FFT 창 길이 보다 길어야 한다. 출력 FFT 데이터가 fftWindowLength 의 길이다. 이는 소스 데이터의 길이와 일치 하지 않을 수도 있다. 출력 데이터는 절대 값을 갖고 있다.

22. 헤드리스 모드

헤드리스 모드는 GUI (그래픽 유저 인터페이스) 에 접근할 수 없는 기계에 작업을 할 수 있는 소프트웨어 기능이다. “헤드리스”란 소프트웨어가 주변 장치 (디스플레이, 키보드, 마우스 등)이 필요 없을 때 사용 되는 단어다. 주변 장치가 없음은 초기화 또는 실행의 실패를 일으키지 않는다. 하지만 이런 경우에는 소프트웨어가 입력을 받고 출력을 제공을 네트워크 또는 시리얼 포트 등 기타 통신 인터페이스를 통해 한다.

22.1.1 헤드리스 렌더링

헤드리스 구성은 라이트닝차트를 헤드리스/서버 환경에서 실행을 가능케 한다. 기대 시나리오는 유저 인터페이스 (UI) 없이 소프트웨어 어플리케이션의 배경 렌더링 및 차트 내용에서 비트맵 이비지 생성 등이 있다. 이 이미지를 후에 헤드풀 시스템으로 넘겨 더 많은 렌더링을 할 수 있다.

기본 사용:

```
var chart = new LightningChartUltimate(new RenderingSettings()
{
    HeadlessMode = true
});
```

헤드리스 모드는 **HeadlessMode** 플래그를 true 로 설정 하여 활성화 할 수 있다. 이 속성은 (WPF 에서는) **chart.ChartRenderOptions** 으로 또는 (윈폼에서는) **chart.RenderOptions** 로 접근 가능하다. 라이트닝차트는 윈도우 서비스 종류 어플리케이션에서 사용을 감지해서 모드를 지정할 필요가 없다.

22.1.1.1 추가 초기화 옵션

UI 가 없고 비주얼 부모가 없는 라이트닝차트의 초기화 경우는 렌더링 요청을 받지 않는다. 이는 레이아웃 크기 변경 또는 프레임 렌더링 등을 포함한다. 윈폼이 이 생성 시간에 엔진 초기화를 하는 도중 WPF 차트는 이 시그널을 이용해 렌더링 엔진을 초기화 한다. 그럼으로 다음 연산 및 구성을 차트에 사용자가 적용해야 한다:

- 크기를 **chart.Width** 및 **chart.Height** 속성들로 정의.
- (WPF 에만 해당) **chart.InitializeRenderingDevice(true)** 를 불러 렌더링 엔진 초기화 요청.
- **chart.AfterRendering** 에 구독해 이미지 수출 로직을 적용.

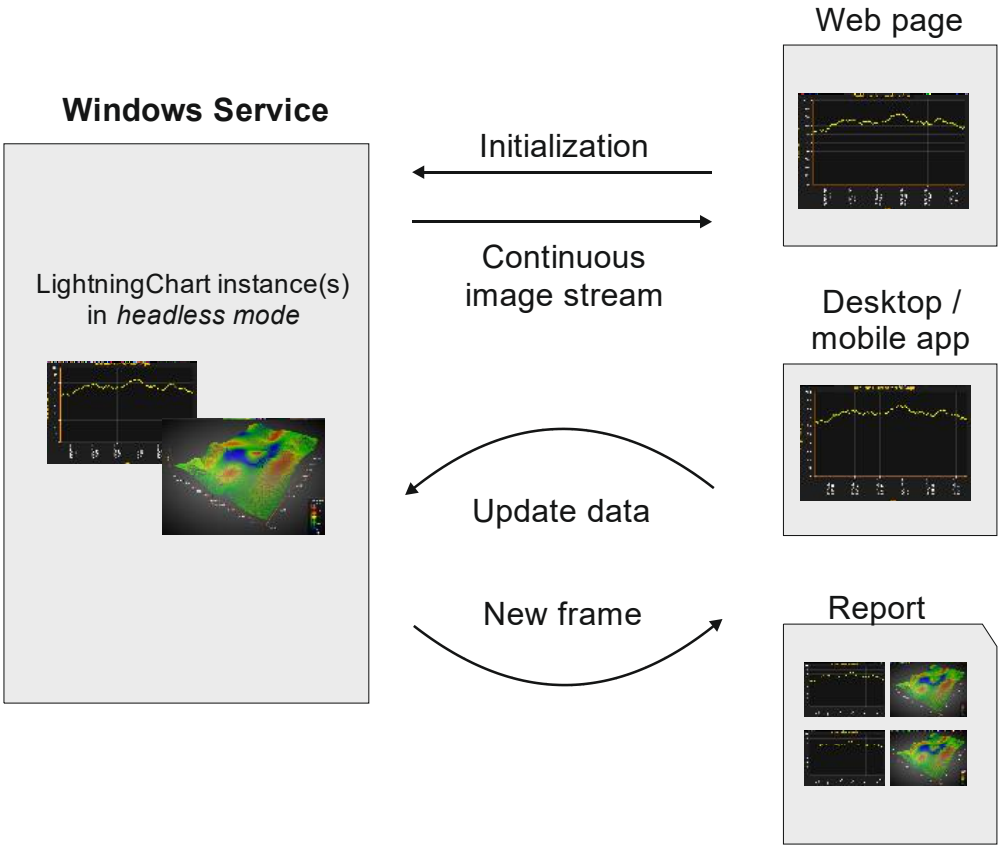
차트는 이 속성 변화에 반응을 한다. 새로운 프레임의 렌더링은 **BeginUpdate()** 및 **EndUpdate()** 을 불러 필요한대로 요청 가능하다.

22.1.1.2 이미지 캡쳐

렌더링 된 프레임은 여러 방식 대로 수출 가능하다 (14 장을 보라):

- **OutputStream** 속성
- **SaveToStream** 메소드
- **CopyToClipboard** 메소드
- **CaptureToByArray** 메소드
- **SaveToFile** 메소드

기본적으로 미트맵 스트림이 선호 된다. 또한 ViewXY 차트는 헤드리스 모드에서 **SaveToStream** 및 **SaveToFile** 메소드로 EMF, WMF, SVG 를 지원한다.



보기 23- 1. 예시 사용 표.

22.1.2 제한 및 요구 사항

22.1.2.1 쓰레드

헤드리스 구성은 배경 작업을 위해 라이트닝차트를 비주얼 부모 안에 넣지 않고 전경 스레드(GUI 쓰레드)에서 차트에게 접근 없이 하는 것을 허용한다. 라이트닝차트 경우의 생성 도중 차트의 속성이 차트를 생성한 같은 스레드 내에서 업데이트 되어야 한다.

- “아파트”라 불리는 COM 스레딩 모델은 MTA (멀티 스레드 아파트)가 아닌 STE (싱글 스레드 아파트)여야 한다. 보통 UI 어플리케이션에는 STA 가 기본 모델이지만 윈도우 서비스에는 MTA 상태가 기본 선택이다.
- 모든 접근 (업데이트, 생성 및 제거)는 그 스레드로 생성 되어야 한다. UI 는 GUI 스레드에서만 만져져야 한다. 그럼으로 대화식 연산이 필요하면 차트 스레드에서 GUI 스레드로 먼저 옮겨야 한다. 주의! 라이트닝차트는 GUI 스레드로 실행 가능하다.
- 스레드는 유효하고 활동적인 메시지 큐 펌프가 있어야 한다. 예를 들어 **Application.Run** 를 스레드에서 실행하라.

22.1.2.2 차트 업데이트

라이트닝차트는 렌더링에 싱글 버퍼를 사용해서 이전 이미지의 수출이 끝난 후에 새로운 이미지의 수출이 시작 된다. 이 동기화적인 구성 (**ChartUpdateType.Sync**)은 차트 속성을 업데이트 하는 요청을 받은 직후 이미지의 렌더링을 제공한다. 싱크 모드를 활성화하여 헤드리스 모드로 더욱 빠르고 방해 없는 성능을 활성화 하라.

22.1.2.3 엔진 지원

DirectX 9 & 11 엔진 둘다 헤드리스 모드에서 실행 된다. 하지만 DirectX 11 만이 윈도우 서비스 종류 어플리케이션에서 사용 가능하다. 이는 MS DirectX 의 제한들 때문이다.

22.1.2.4 라이센스

기본적으로, 윈도우 서비스는 사용자 시스템 사용자 계정의 보안 내용 내에서 실행된다. **트라이얼 및 개발 라이센스의 설치는 불가능하다.** 이 이유 때문어 서비스 어플리케이션은 유효한 **배포 키**를 가져야만 하거나 유효 라이센스 사용자의 신원으로 실행해야만 한다 (실험/개발).

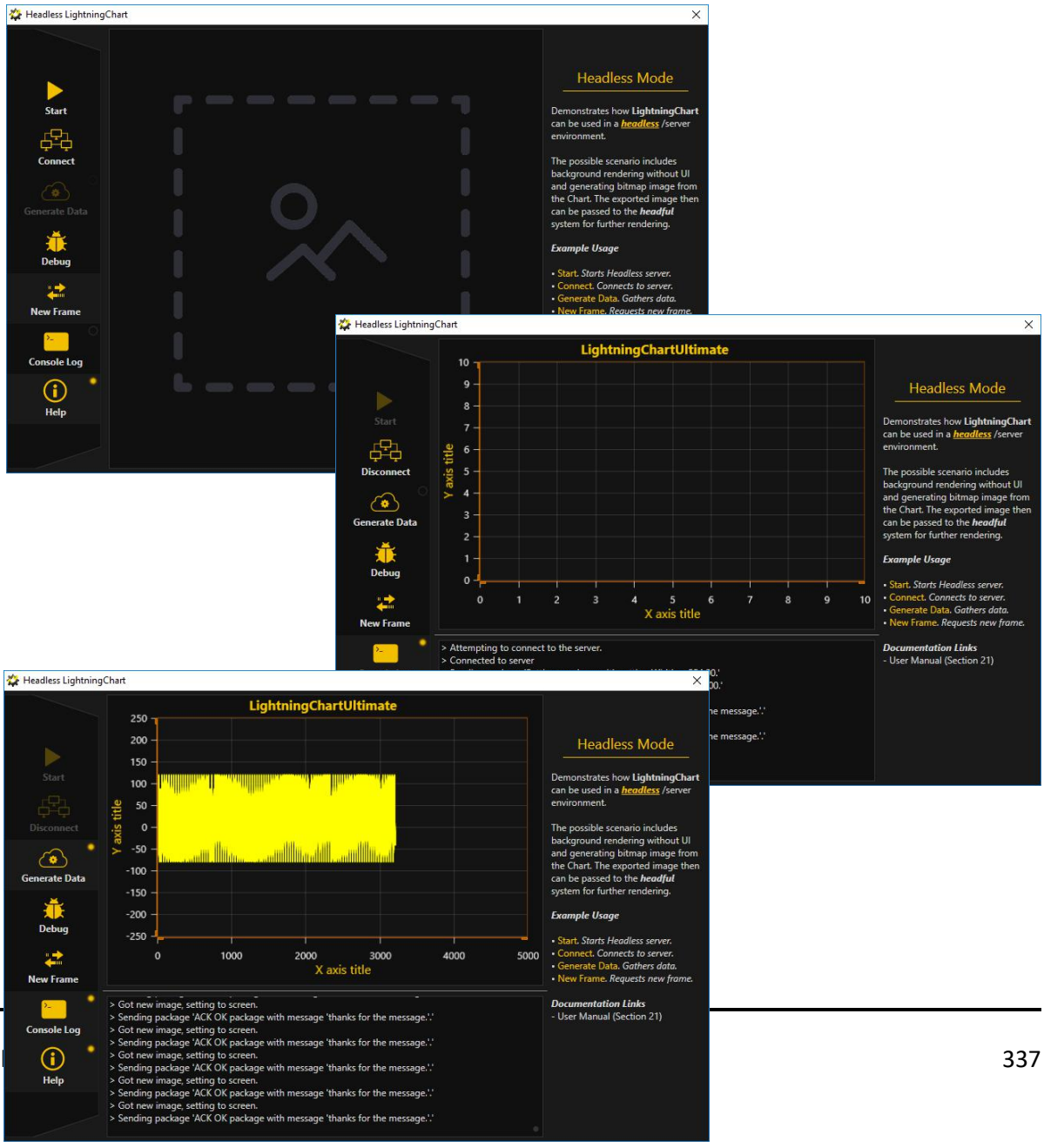
22.1.3 해결 예시

라이트닝차트 SDK 는 다음을 포함한 비주얼 스튜디오 솔루션의 예시와 같이 온다 **(DemoService.sln).**

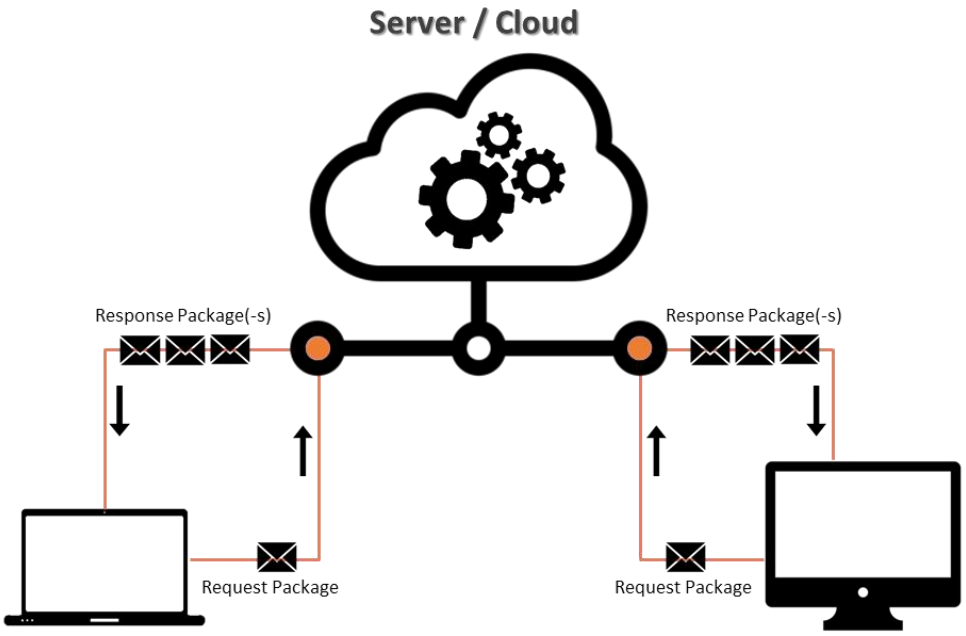
- 서비스
- 콘솔 어플리케이션
- WPF 위한 클라이언트 어플리케이션

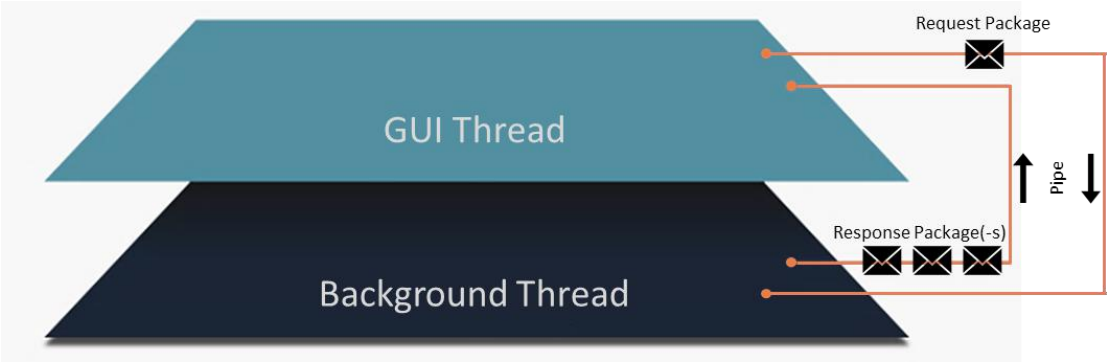
DemoService.sln 는 **C:\ProgramData\Arction\LightningChart .NET SDK v.8\DemoService** 폴더에서 찾을 수 있다.

WPF 클라이언트를 실행할 때 이는 프레임 컨테이너를 창 가운데에 표시한다.



보기 23- 2. WPF 클라이언트 앱. Start, Connect, Generate Data 이후 이는 이미지 스트림을 계속해서 업데이트 하는 것을 보인다.





보기 23- 3. 헤드리스 데모 서비스 – 클라이언트 내부 이름 있는 파이프 와 배경 스레드가 표현된 메시지 운영

23. WPF 어플리케이션에서 윈도우 폼 차트 사용

라이트닝차트는 3 개의 WPF API 를 사용 가능하다. 특수한 경우에만 윈폼 차트 API 를 WPF 어플리케이션에서 사용하는 것을 고려하라.

23.1 에서 아크션 윈도우 폼 제어 사용은?

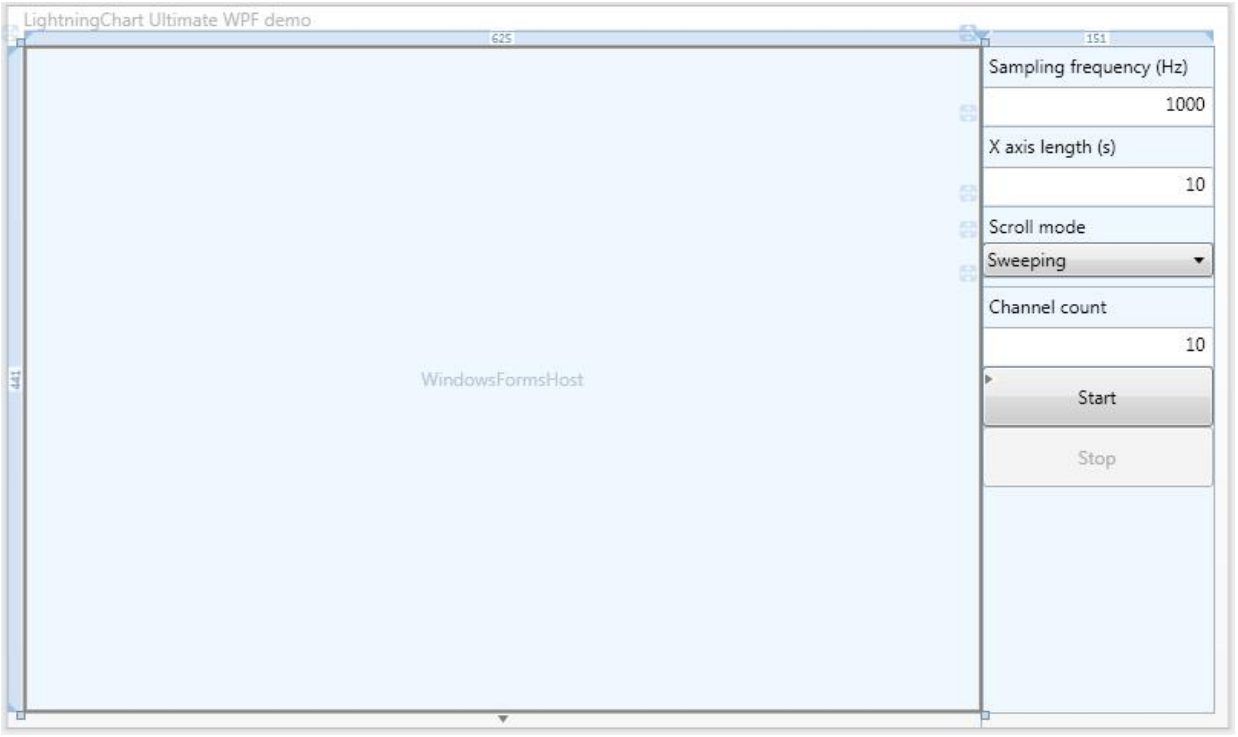
WPF 에서는 윈도우 폼 구성 요소를 **Arction.WinForms.Charting.LightningChartUltimate.dll 및 Arction.WinForms.SignalProcessing.SignalTools.dll** 을 프로젝트에 잠조로 추가하여 사용 가능하다. 추가 후 코드로 생성해야 한다. 라이트닝차트 얼티밋 제어 및 대부분의 다른 제어들은 빌트인 UI 를 갖고 있다. **WindowsFormsHost** 를 이들의 부모 컨테이너로 사용하라. 이 컨트롤은 UI 없이 이들의 메소드 및 속성으로 사용 가능하다.

23.2 WPF 에서 Arction.WinForms.LightningChartUltimate 을 사용할까?

WPF 차트 어셈블리를 사용하는 것이 WPF 어플리케이션에서 윈폼 차트를 사용하는 것 보다 추천된다. 이는 **WindowsFormsHost** 컨트롤이 필요없어서 **WindowsFormsHost** 컨트롤의 일반적인 "공적" 문제를 갖고 있지 않다. 또 다른 장점은 WPF 차트는 투명 배경을 가질 수 있고 차트를 겹쳐서 놓을 수 있다.

윈폼 차트 컨트롤과 **WindowsFormsHost** 컨트롤 사용은 절대 최대 성능이 필요할 때 사용 가능하다. **WindowsFormsHost** + WinForms 차트 렌더링이 살짝 더 빠르다.

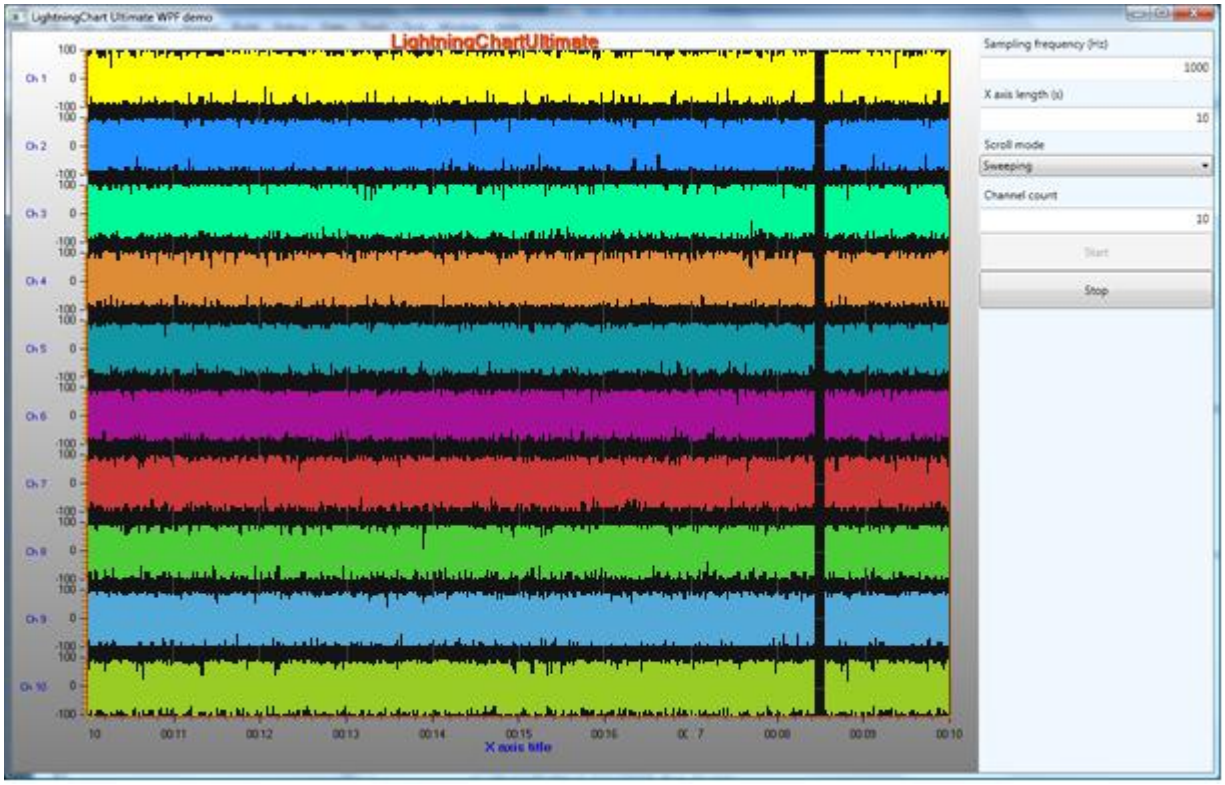
사용자가 WPF 어플리케이션에서 윈폼 차트 사용을 선택한다면 이는 **WindowsFormsHost** 컨트롤 내에 놓여야 한다 **WindowsFormsHost** 컨트롤을 WPF 폼에 추가하라 (비주얼 스튜디오 WPF 툴박스에서 찾을 수 있다).



보기 24- 1. 디자이너에서 WPF 어플리케이션. WindowsFormsHost 는 라이트닝차트 얼티밋 객체를 어플리케이션이 실행될 때 안에 유지한다.

라이트닝차트 얼티밋 객체를 생성해 **WindowsFormsHost** 객체 내에 코드로 넣어라. 폼 xaml.cs 파일을 열어 폼 생성자에 차트를 생성하라:

```
public WindowMain()
{
   InitializeComponent();

   CreateChart();
}

private LightningChartUltimate m_chart = null;

void CreateChart()
{
        m_chart = new LightningChartUltimate();

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



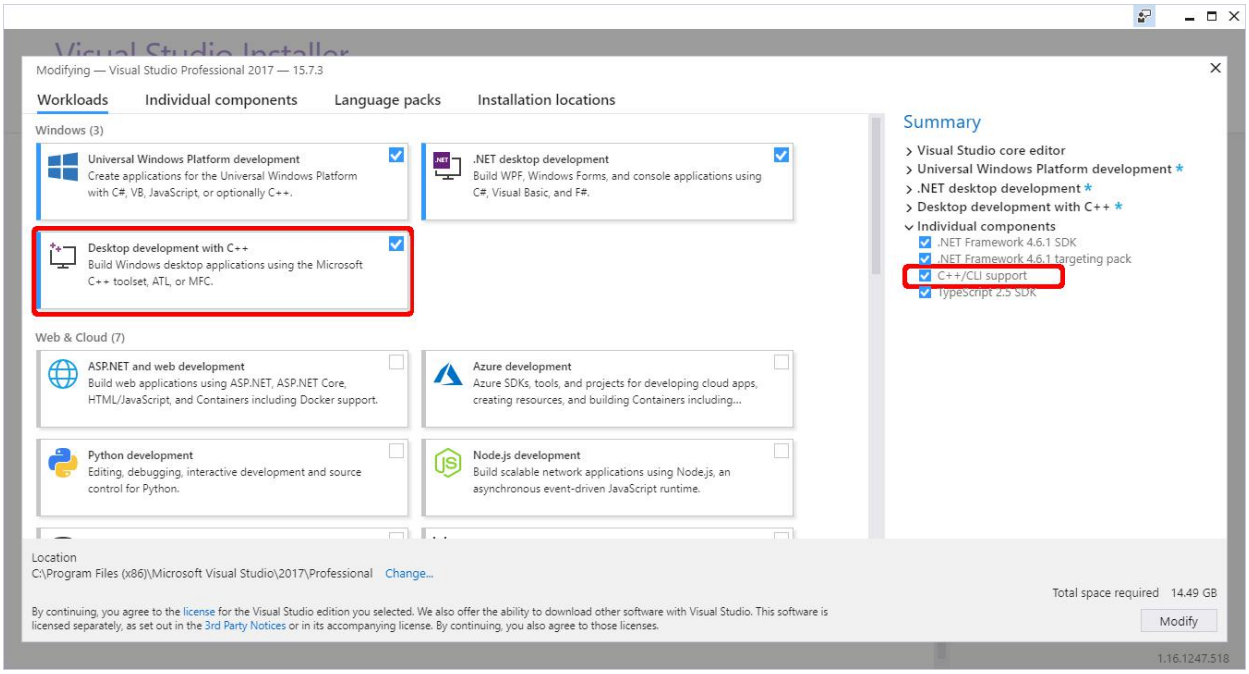
보기 24- 2. WPF 어플리케이션 안에 있는 윈폼 차트

24. C++ 어플리케이션에서 라이트닝차트 사용

라이트닝차트는 .NET 라이브러리다. 이는 C# 및 VB.NET 언어와 유창하다. 하지만 라이트닝차트를 C++ Win32 어플리케이션에서도 사용 가능하다. 이는 MFC 어플리케이션도 포함한다. 라이트닝차트를 사용하는 어플리케이션은 **커먼 랭귀지 런타임 서포트 (/clr)** 옵션으로 컴파일 되어야 한다. 윈도우 폼 프로젝트를 C++를 사용해 생성할 때 다음 튜토리얼을 꼭 참고하라.

24.1 필요 C++/CLR 패키지 설치

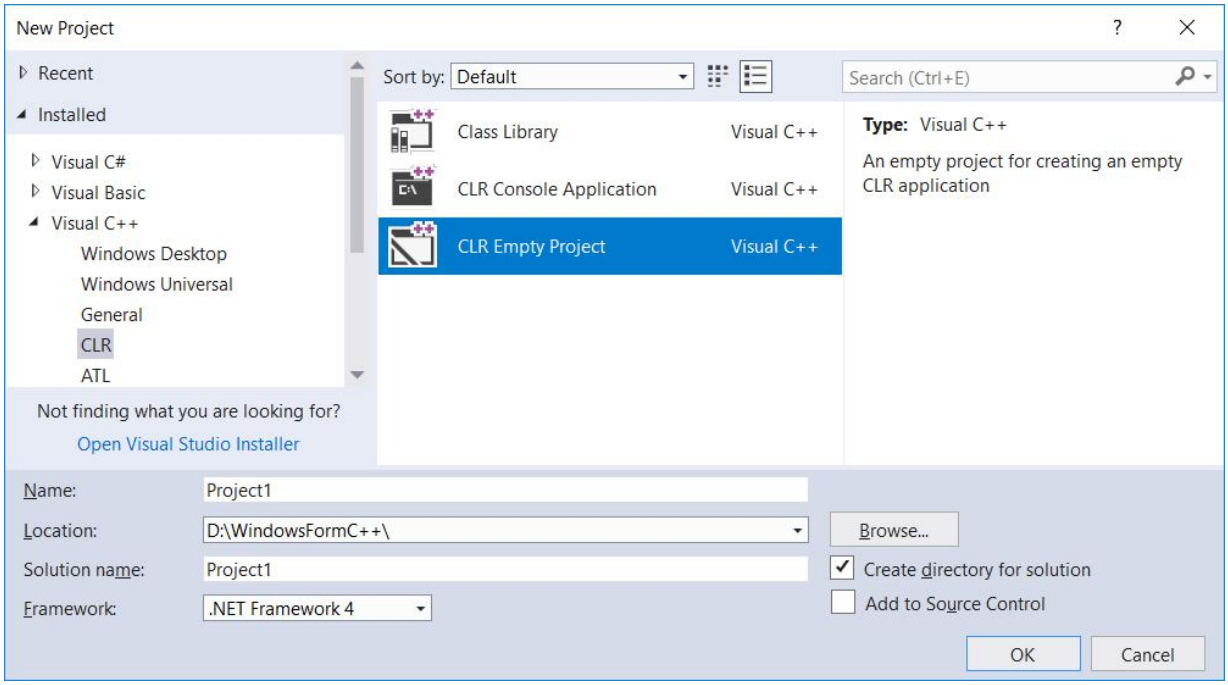
비주얼 스튜디오가 C++ 패키지를 C++/CLR 로 설치 한 것을 확인하라. 예를 들어 비주얼 스튜디오 (2017) 설치자를 실행해 업데이트/변경 버튼을 선택하라. 개별 구성 요소에서 **C++/CLI 지원** 을 선택하라. 워크로드에서는 **C++로 데스크탑 개발**을 선택하라.



보기 25- 1. C++ 프로젝트를 위한 비주얼 스튜디오 설치자 옵션

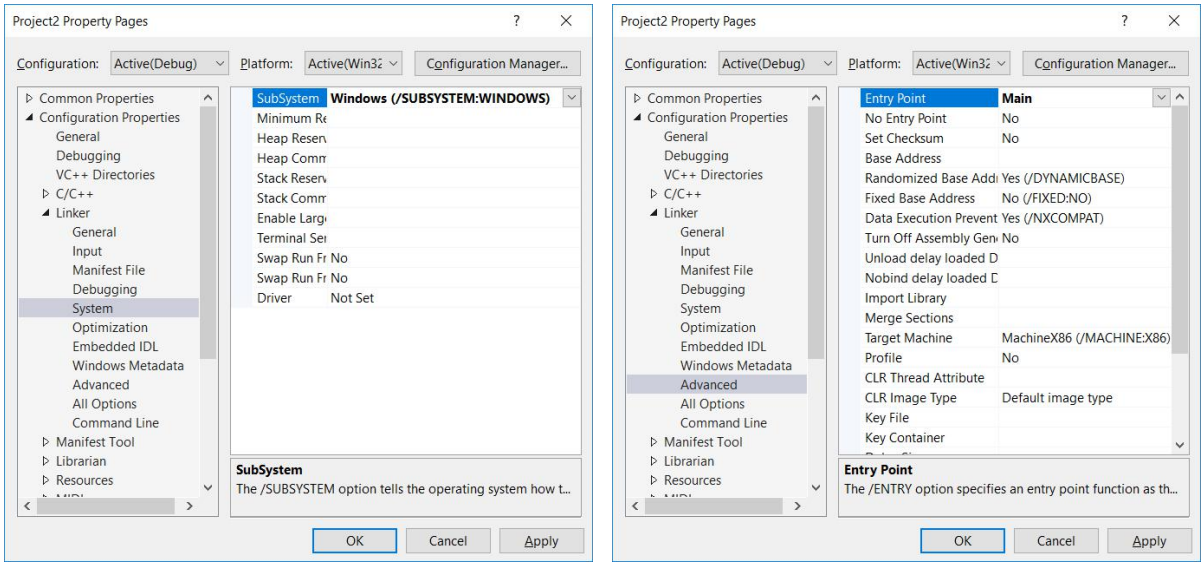
24.2 비주얼 스튜디오 프로젝트 설정

비주얼 스튜디오 2017 을 열어 새로운 프로젝트를 생성하라. 위에 설명 된 모든 필요한 패키지 및 구성 요소가 설치 되었으면 새로운 프로젝트를 생성할 때 다음 선택권이 생긴다 (템플레이트 -> 비주얼 C++ -> CLR -> CLR 공프로젝트).



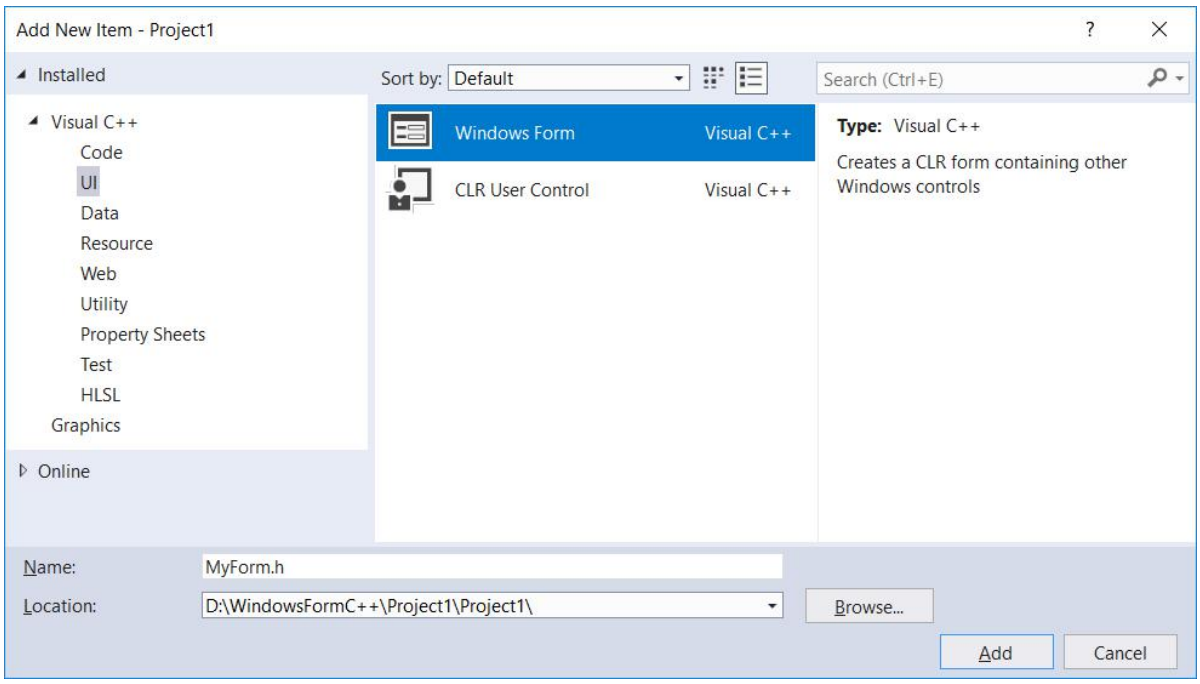
보기 25- 2. 윈도우 폼 C++ 프로젝트 템플레이트.

생성된 프로젝트를 우클릭하고 속성 옵션을 선택하라. 보기 24-3 에 보여진 대로 **Configuration Properties -> Linker -> System -> SubSystem** 및 **Linker -> Advanced -> Entry point** 을 변경하라.



보기 25- 3. C++ 프로젝트 속성 페이지 .

윈도우 폼 아이템을 프로젝트에 추가하라: 프로젝트 우클릭 후 **Add -> New Item...** 선택. 보기 24-4 에 보여진 대로 윈도우 폼을 선택하라. 에러 메시지가 나올 수 있다: **The data necessary to complete this operation is not yet available. (Exception from HRESULT: 0x8000000A)**. 이는 무시해도 된다. 닫고 다음 단계로 넘어가라.



보기 25- 4. C++ 프로젝트에 새로운 윈도우 폼 아이템 추가

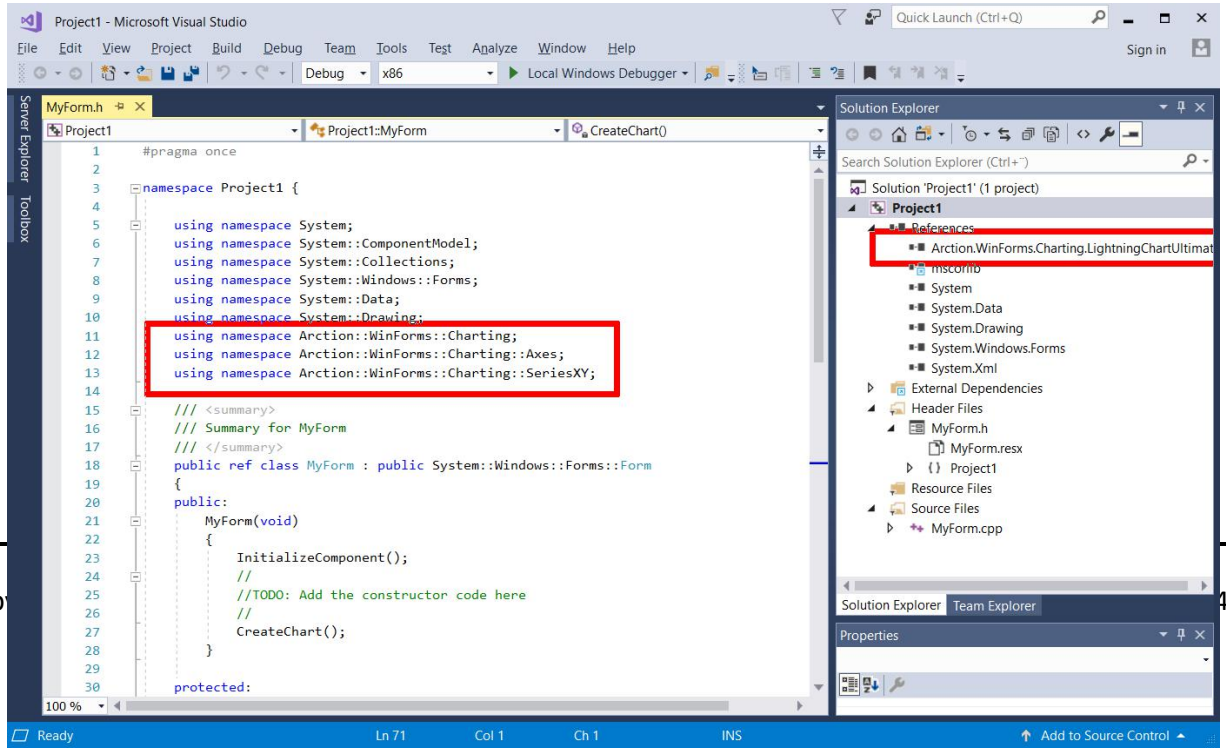
생성된 폼에 (이 예시에는 MyForm.cpp) 다음과 같은 코드를 추가하라. 저장 후 비주얼 스튜디오 2017 을 닫아라.

```
#include "MyForm.h"

using namespace System;
using namespace System::Windows::Forms;

[STAThreadAttribute]
void Main(array<String^>^ args) {
        Application::EnableVisualStyles();
        Application::SetCompatibleTextRenderingDefault(false);
        Project1::MyForm form;
        Application::Run(%form);
    }
```

프로젝트가 처음으로 빌드 가능하다. 프로젝트를 다시 열어 **Build -> Rebuild Solution** 을 선택하라. 프로젝트가 실행될 때 비어 있는 위도우 폼이 보여야 한다.

24.3 C++ 프로젝트에서 라이트닝차트 어플리케이션 생성

MyForm.h 파일을 수정해 폼에 구성 요소를 이제 추가할 수 있다. 밑에는 **PointLineSeries** 가 있는 차트를 어떻게 생성할 수 있는 간단한 설명이다. 아크션의 WinForms DLL 을 참조 목록에 포함하고 해당 네임스페이스를 **MyForm.h** 코드 파일에 추가하라.



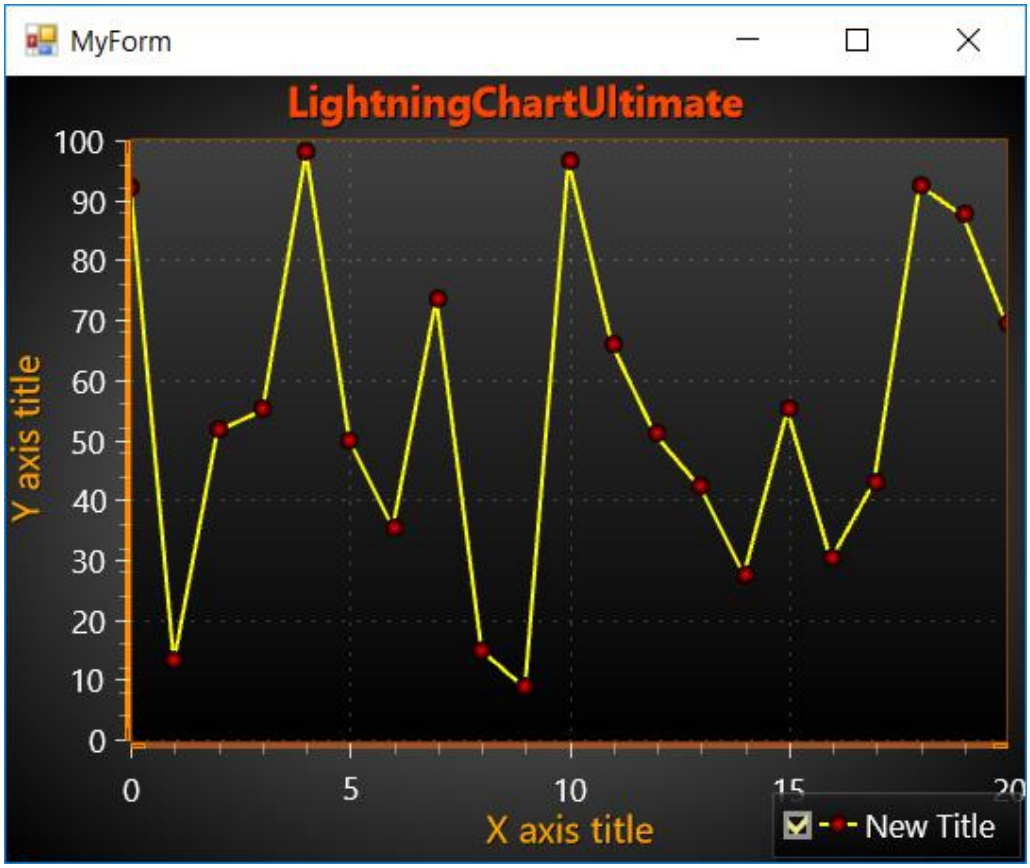
보기 25- 5. Arction.WinForms.Charting.LightningChartUltimate.dll 을 참조 목록에 포함하고 해당 네임스페이스를 프로젝트에 추가하라.

'차트' 변수를 선언하고 속성을 설정하라. 아래에 차트 생성 방법의 예시가 있다.

```
protected:
LightningChartUltimate ^ _chart;

void CreateChart()
{
	_chart = gcnew LightningChartUltimate();

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

	//Set parent window by window handle
	_chart->Parent = this;

	//Fill the form area
	_chart->Dock = DockStyle::Fill;

	_chart->ActiveView = ActiveView::ViewXY;

	// Configure x-axis.
	AxisX^ axisX = _chart->ViewXY->XAxes[0];
	axisX->SetRange(0, 20);
	axisX->ScrollMode = XAxisScrollMode::None;
	axisX->ValueType = AxisValueType::Number;

	// Configure y-axis.
	AxisY^ axisY = _chart->ViewXY->YAxes[0];
	axisY->SetRange(0, 100);

	PointLineSeries^ pls1 = gcnew PointLineSeries(_chart->ViewXY, axisX, axisY);
	pls1->LineStyle->Color = Color::Yellow;
	pls1->Title->Text = "New Title";
	pls1->PointsVisible = true;
	pls1->LineVisible = true;
	_chart->ViewXY->PointLineSeries->Add(pls1);

	// Generate random data.
	Random rand;
	int pointCount = 21;

	array<SeriesPoint> ^ points = gcnew array<SeriesPoint>(pointCount);
	for (int point = 0; point < pointCount; point++)
	{
		points[point].X = (double)point;
		points[point].Y = 100.0 * rand.NextDouble();
	}

	pls1->Points = points;
```

```
            // Allow chart rendering.
            _chart->EndUpdate();
}
```

컴파일 및 실행 되는 결과 어플리케이션



보기 25- 6. 예시 어플리케이션 실행.

25. 제거 패턴

25.1 차트 제거

차트가 코드로 생성 되었고 더이상 필요가 없을 때 **chart.Dispose()** 를 불러라. 이는 차트와 이의 모든 객체들을 (시리즈, 마커, 팔레뜨 단계 등) 저장 공간에서 비운다.

25.2 객체 제거

객체를 임의로 생성했고 어플리케이션 종료 전 혹은 차트를 **chart.Dispose()** 로 제거 전 메모리를 비워야 한다면, 객체가 추가된 컬렉션에서 부터 지우고 객체를 위해 **Dispose()** 를 불러라.

예를 들어 **chart.ViewXY.PointLineSeries** 컬렉션에서 시리즈를 제거하기:

```
//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();
```

라이트닝 차트의 객체가 더이상 필요가 없을 때 메모리 리킹을 방지하기 위해 제거하는 것이 좋은 버릇이다.

26. 객체 모델 노트

26.1 다른 객체와 객체 공유

라이트닝차트 객체 모델은 트리-기반이다. 각 클라스는 그의 부모 객체 및 자녀 객체 목록이 있다. 이 트리 모델은 자녀 객체가 부모 객체에게 변화를 알려 부모가 반응할 수 있게 한다. 반대로 부모가 이의 부모에 알림을 보내어 루트 노드인 라이트닝차트 얼티밋 자체에 도달돼 이에 맞게 새로 고침을 언제하는지 안다.

차트는 주어진 모든 객체의 주인이 되어 더 이상 필요가 없을때 제거한다. 이는 차트에 주어진 이전 객체를 새로운 객체로 교체해야 하는데 객체의 부모가 제거 됐을 때 경우도 얘기한다. 사용자는 이를 알아야 한다. 그러지 않으면 제거된 객체를 사용할 수가 있다.

객체가 다른 .NET 구성요소와 라이트닝차트 사이 공유 되고 라이트닝차트가 객체를 제거하면 .NET 구성 요소는 제거된 객체와 남게 된다. 라이트닝차트는 자신과 다른 구성 요소 사이 부모 공유를 감지할 수 없다.

같은 차트 내 또는 다른 차트 경우와 다른 객체와 객체 공유는 허용되지 않는다.

잘못된 사용의 예시 1:

```
AnnotationXY annotation1 = new AnnotationXY();
chart.ViewXY.Annotations.Add(annotation1);

AnnotationXY annotation2 = new AnnotationXY();
annotation2.Fill = annotation1.Fill;
chart.ViewXY.Annotations.Add(annotation2);
```

문제: 같은 **Fill** 객체를 여러 객체 사이 공유할 수 없다.

제대로 된 방식: ValueType 속성들만 복사하라 (Integer, Double, Color 등)

잘못된 사용의 예시 2:

```
SeriesEventMarker marker = new SeriesEventMarker();

chart.ViewXY.PointLineSeries[0].SeriesEventMarkers.Add(marker);
chart.ViewXY.PointLineSeries[1].SeriesEventMarkers.Add(marker);
```

문제점: 같은 객체가 여러 컬렉션의 컬렉션에 추가 되어선 안된다.

제대로 된 방식: 여러 마커를 시리즈당 한개를 생성하라.

ChartMessage 이벤트 핸들러에 구독하는 것을 기억하라. 대부분의 경우에는 무효한 객체 공유 에러를 알린다 (16 장을 보라).

27. 라이트닝차트 어셈블리의 배포

27.1 참조된 어셈블리

실행 가능한 폴더와 아크션 dll 파일들을 전달하라. 폴더 옆에 글로벌 어셈블리 캐시 또는 .NET 어셈블리 해결 시스템이 찾을 수 있는 다른 폴더 옆에 놓아라. 라이트닝차트는 **ClickOnce** 배포를 지원한다.

윈폼:

- 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

SignalTools 을 사용 시

- Arction.WinForms.SignalProcessing.SignalTools.dll
- Arction.MathCore.dll

WPF:

- Arction.Wpf.Charting.LightningChartUltimate.dll **(for Non-bindable WPF chart)**
- Arction.Wpf.SemibindableCharting.LightningChartUltimate.dll **(for semi-bindable WPF chart)**
- Arction.Wpf.BindableCharting.LightningChartUltimate.dll **(for 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

SignalTools 사용 시

- Arction.Wpf.SignalProcessing.SignalTools.dll
- Arction.MathCore.dll

27.2 라이센스 키

모든 구성 요소에 정적 **SetDeploymentKey** 메소드를 사용하는 것을 기억하라. 그러지 않으면 차트는 트라이얼 모드에 들어가며 30 일 동안만 작동한다. 이 기간동안 업그레이드 하라고 괴롭힌다. 배포키 생성 및 라이센스 키 관리에 대한 상세 정보를 위해서는 4 장을 보라.

27.3 어플리케이션 코드 난독화

어플리케이션 코드의 난독화는 .NET 디서셈블러 도구에 라이트닝차트 라이센스 키가 보이지 않게 **필수**이다. 새는 라이센스 키는 라이센스 해제로 이어질 수 있으며, 법적 조치 및 손실 청구로도 이어질 수 있다.

27.4 라이트닝차트 코드 난독화

라이트닝차트 소스 코드 구독자는 라이트닝차트 라이브러리 소스 코드에 접근이 허용된다. 라이트닝차트 소스 코드로 빌드 된 어셈블리를 난독화 하는 것은 필수이다. 이는 아크션의 지적 재산권 및 코드가 새는 것을 방지 하기 위함이다. 비난독화 라이트닝차트 라이브러리의 배포는 EULA 의 위반이며 라이센스 해제, 법적 조치 및 손실 청구로 이어질 수 있다.

27.5 아크션 어셈블리의 XML 파일

이 XML 파일의 배포는 금지다.

- Arction.WinForms.Charting.LightningChartUltimate.xml
- Arction.Wpf.BindableCharting.LightningChartUltimate.xml
- Arction.Wpf.Charting.LightningChartUltimate.xml

- Arction.Wpf.SemibindableCharting.LightningChartUltimate.xml
- Arction.Wpf.SignalProcessing.SignalTools.xml
- Arction.WinForms.SignalProcessing.SignalTools.xml

아크션으로 제공된 파일들은 어플리케이션 개발에 도움을 주기 위해서다. 이들은 코드 매개 변수 및 속성에 대한 도움을 주기 위해서다. 소스 코드에서 라이트닝차트 어셈블리를 재건 할 때 위에 나와있는 XML 파일들이 배포 되지 않게 확인하라. **이들의 배포는 절대 금지다.** 이는 .NET 디서셈블러와 리버스 엔지니어링 어플리케이션에게 너무 많은 정보를 밝히기 때문이다.

28. 문제 해결

28.1 구 버전에서 업데이트

라이트닝차트 구성 요소 API 는 이전 버전에서 변했을 수 있다. 이런 경우 프로젝트가 로드 되지 않거나 새로운 버전을 자동을 사용하지 않을 수 있다. 다음 설명은 새로운 버전 어셈블리를 프로젝트의 참조로 설정하고 비주얼 스튜디오 폼 에디터에서 디시리얼화가 안되었던 속성들을 고치는 방법을 알려준다.

차트를 업데이트 하기 위해 이전 버전의 참조는 제거하고 새로운 버전에 참조를 추가해야 한다. 어떤 경우에는 *.Designer.cs 및 *.resx 파일들을 고쳐야 할 수도 있다. 이 파일들에는 바이너리 호환 되지 않는 속성들이 있을 수 있기 때문이다.

프로젝트 레퍼런스에서 옛날 참조 제거

1. 솔루션 익스플로러에 가라.
2. 레퍼런스 폴더를 열어라.
3. 아크션 어셈블리를 선택 후 삭제 버튼 또는 우클릭 후 제거 선택으로 제거하라.

새로운 버전에 참조 추가

1. 솔루션 익스플로러에 가라.
2. 레퍼런스 폴더를 열어라.

3. 새로운 차트에 참조를 추가하라. 레퍼런스 폴더에 우클릭하라. 참조 추가…를 선택하고 새로운 아크션 DLL 파일을 선택하라.

API 가 변경 되었을 수도 있기 때문에 변경된 속성의 소스 코드도 고쳐야 할 수도 있다.

차트가 완전히 비호환 되면 (비주얼 스튜디오가 폼 에디터에서 UI 를 로딩 못한다 등) 라이트닝차트 속성 설정자들이 *.Designer.cs 및 *.resx 파일들에서 제거 되어야 한다.

***.Designer.cs 파일에서 속성 설정자 제거**

1. *.Designer.cs 파일을 텍스트 에디터에서 열어라 (또는 비주얼 스튜디오 이외 에디터를 사용하라)
2. 라이트닝차트 얼티밋의 설정자가 있는 행을 찾아 삭제하라. 예를 들어:
   this.m_chart.Background =
   ((Arction.LightningChartUltimate.Fill)(resources.GetObject("m_chart.Background")));

상속 속성을 제거할 필요는 없다. 이들은 위의 메소드 같은 걸로 리소스에서 읽혀진다 "NN = ((...)(resources.GetObject("...")));".

***.resx 파일에서 시리얼화 아이템 제거**

1. *.resx 파일을 텍스트 에디터에서 열어라.
2. Xml 태그 달린 아크션 객체를 찾아라 (차트 멤버 이름으로 구별 가능하다, 예를 들어 "m_chart" 또는 "lightningChartUltimate1").
3. Xml 객체 끝에 <data> 태그가 붙은 라인들을 제거하라 (</data> 태그). 예를 들어 차트 배경이 다음과 같은 xml 객체로 시리어화 되었으면 이후 모든 라인들을 *.resx 파일에서 제거해야 한다.

```
<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>
```

어느 객체들은 매우 클 수도 있다. 예를 들어 제목 열 카운트가 약 200 줄이 될 수 있다. 뷰는 보통 이보다도 훨씬 크다 (View3D 는 약 2000 줄이 있다)
여러 차트가 있을 경우에는 모든 시리얼화 된 속성들을 제거 해야 한다. 에디터 서치는 차트 객체를 찾기 위해 유용한 도구이다.
객체가 모두 *.resx 파일에서 제거 되고 관련 설정자들이 *.Designer.cs 파일에서 제거 되었을 때 프로젝트를 비주얼 스튜디오 폼 에디터에서 성공적으로 열 수 있을 것이다.

28.2 웹 지원

지원 옵션을 위해 www.arction.com/support 을 보아라.

http://www.arction.com/forum 에서 토론 포럼에 참가하라.

28.3 가상 기기 플랫폼에서 실행

라이트닝차트는 그래픽 하드웨어에 액세스 허용이 안되는 시스템을 위해 DirectX10/11 WARP 렌더링과 같이 온다. WARP 렌더링은 CPU 에서 일어나기 때문에 하드웨어 렌더링에 비해 성능이 저하될 것이다. 이는 DirectX11 가 지원되는 운영 체제가 필요하다.

DirectX11 가 지원되지 않는 시스템에는 라이트닝차트는 DirectX9 Reference Rasterizer 모드로 폴백한다. 성능은 WARP 성능의 일부 밖에 되지 않아 매우 안좋다. WARP 및 DirectX9 로 자동 폴백을 하기 위해 RenderDevice 를 **Auto**, **AutoPreferD9 또는 AutoPreferD11** 로 설정을 유지하라 (5.10 장).

29. 크레딧

29.1 인텔 수학 커널 라이브러리

LightningChart® .NET SDK 는 어느 부분에서 인텔 수학 커널 라이브러리를 사용한다. 예를 들어 Fast Fourier Transform 메소드 등이 있다. 아크션 어셈블리의 몇 기본 DLL 파일들은 이 라이브러리로 빌드 되었다. (주) 아크션은 인텔 수할 커널 라이브러리를 합법적으로 사용할 수 있다.

29.2 오픈 소스 프로젝트

다음 오픈 소스 프로젝트 및 기재 제공자들에게 감사의 인사를 전하고 싶다:

.NET 를 위한 DirectX 라이브러리

라이트닝차트는 SharpDX 유래된 아크션 확장이 있는 DirectX .NET DLL 을 사용한다.
http://www.sharpdx.org/

지도 소스

LightningChart® .NET 지도들은 다음 지도 제공자로 부터 불러왔다:

세계, 북미, 유럽:	내츄럴 어스, http://www.naturalearthdata.com/
호주:	호주 통계국, http://www.abs.gov.au/
미국의 길:	미국의 국제 아틀라스, http://www.nationalatlas.gov

스케일 가능 벡터 그래픽 산출

라이트닝차트 SVG 수출은 주 리스크케어의 SvgNet 프로젝트 코드의 부분을 사용했다

다항식 회귀

다항식 회귀 계산 코드는 Math.Net 라이브러리의 부분을 사용했다, http://www.mathdotnet.com/

변경된 소스 코드 부분들은 아크션 지원 (support@arction.com) 에게 무료 문의 요청 가능하다.

오픈소스 프로젝트의 저작권 고지를 위해서는 라이트닝차트 SDK 설치 폴더에 있는 **LightningChart .NET Readme.txt** 을 보아라