닷넷 차트 컨트롤
히포 차트

차례

1장 히포차트 기본 개념 익히기

2장 실전 히포차트 개발하기(기초편)

3장 실전 히포차트 개발하기(기능편)

히포차트 기본 개념 익히기

히포차트 시리즈(Series)의 개념 이해하기

2007년 배포했던 테스트 버전에서 베타버전으로 업그레이드되면서 가장 크게 변화한 부분이 있다면 바로 시리즈의 개념을 도입한 내용일 것입니다. 히포차트의 구조를 이해하기 위해서 가장 중요한 부분이면서 기본이 되는 부분이므로 반드시 숙지하시고 넘어가야겠습니다.

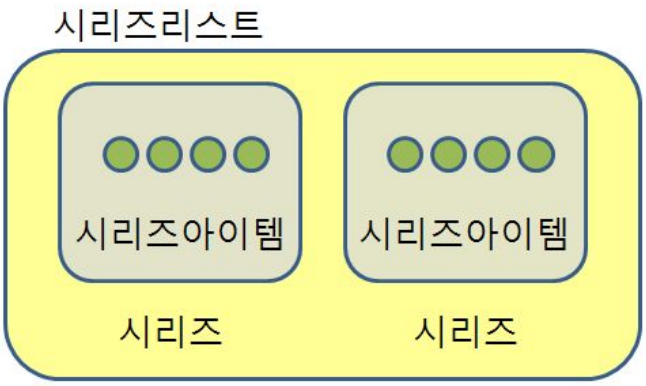
시리즈(Series)란?

Series라는 영어 단어를 사전에 찾아보면 아래와 같이 나옵니다. 이 중에서 히포차트에서 사용하는 의미라면 '일련의 연속된 수 집합' 정도로 이해하시면 되겠습니다.

se·ries [ ] n. (pl. series)
1 [a series] 일련, (…의) 연속 《of》
a series of victories[misfortunes] (연전) 연승[잇단 불행]
2 시리즈, 연속물, 연속 출판물, 총서(叢書), 제 …집(集);(라디오·TV·영화 등의) 연속 프로;연속 강의
a series on African wildlife 아프리카 야생 생물에 대한 연속 프로
[유의어] series 한 작품 한 작품이 완결되면서 계속되는 것 serial 클라이맥스로 끝나고 다음으로 이어지는 연속물

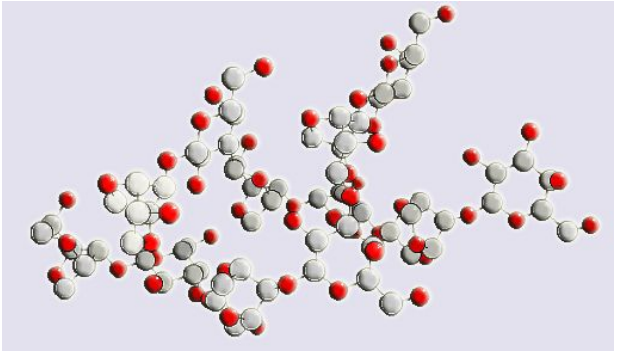
시리즈의 구조

여타의 다른 외국 차트들에도 역시 시리즈의 개념이 들어가 있는데 비슷한 부분도 있고 완전히 다른 부분도 있어 별도로 이해를 하셔야겠습니다. 히포차트의 시리즈 구조는 크게 시리즈리스트, 시리즈, 시리즈아이템의 계층적인 분류할 수 있는데 좀 더 이해하기 쉽게 아래 그림을 그려보았습니다.



이 구조는 마치

화합물 >> 분자들의 집합 >> 분자 >> 원자



처럼 하나의 화합물을 생성하기 위해 수많은 원자들이 모여 분자를 이루고, 그 분자들의 집합 구조로써 화합물이 이루어지듯이 히포차트도 많은 시리즈아이템들로 구성된 시리즈들의 집합들이 보여 시리즈리스트를 이루고 그것이 결국 히포차트를 구성하게 됩니다.

히포차트 >> 시리즈리스트 >> 시리즈 >> 시리즈아이템

그럼 하나씩 자세히 알아보겠습니다.

시리즈리스트(SeriesList)

시리즈리스트란, 하나의 차트군을 그리기 위한 가장 기본적인 단위라고 정의할 수 있습니다. 그럼 차트군이란 무엇인가? 차트군이란, 같은 부류의 차트들의 무리를 의미하는데 그 분류는 아래와 같습니다.

차트군	차트
좌표 특성의 차트	LineChart ColumnChart CircularChart ConeChart SplineChart ScatterChart BarChart GanttChart ImageBarChart KagiChart StackedBarChart* StackedColumnChart* StackedLineChart* StackedSplineChart* StackedCircularChart*
비좌표 특성의 차트	PieChart PylamidChart GaugeChart RadarChart*

(표1 - 히포차트 차트군)

이 표는 시리즈리스트 하나를 생성하고 라인차트를 설정한 시리즈와 파이차트를 설정한 시리즈를 같이 추가할 수 없다는 것을 말해줍니다. 같은 부류의 차트가 아니기 때문이죠.

하지만 여기에도 몇 가지 **예외사항***이 있는데 아주 특수한 차트인 레이다차트인 경우에는 시리즈리스트 레벨에서만 차트 타입을 설정할 수 있습니다. 구체적인 차트별 내용은 뒤에서 다시 알아보겠습니다.

1. 시리즈 리스트 속성

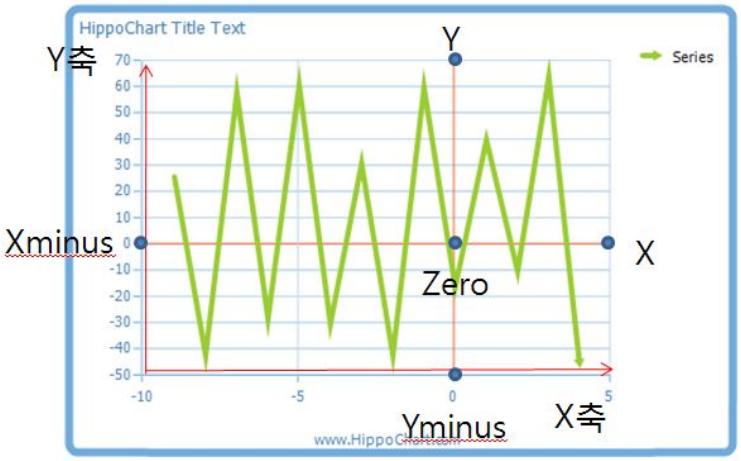
속성	설명
AxisFactor*	시리즈리스트의 좌표 영역에 대한 특징을 설정합니다. 가장 중요한 속성 클래스 중 하나입니다.
ChartType	시리즈리스트의 전체 차트군의 차트타입을 정의합니다. 이 속성은 반드시 설정해야합니다.
CoorcinateType	차트군을 구분하는 속성으로 차트타입 설정 후 내부에서 자동으로 설정합니다.
CoordinativeDirection	차트 좌표의 근본적인 수직, 수평 방향을 정의하는 속성으로 내부에서 자동으로 설정합니다.
Description	좌표 영역 내 빈 공간에 간단한 메모를 할 수 있는 속성입니다.
Gantt	간트차트를 그리기 위한 속성들을 정의하는 속성입니다.
GraphArea*	시리즈리스트내의 실제 그래프가 그려지는 영역을 의미하는 속성으로 배경색, 그리드, 보더라인색 등을 설정할 수 있습니다. 아래서 다시 자세히 다룹니다.
GraphTitle	시리즈리스트에 그려지는 전체 차트군의 타이틀을 설정합니다. 시리즈리스트가 여러 개 있는 차트일 경우에 각각의 제목을 붙일 수 있습니다.
IsShow3D	전체 차트군에 3D 입체를 반영합니다. 입체는 반드시 차트군 전체에 반영되어야합니다.
Legend	이너 레전드(그래프 내부에 범례)를 표시할 경우 sList.Legend.Visible = true; 와 같이 사용합니다. 이너 레전드는 기존 범례와 별도로 동작하므로 바깥쪽 범례를 안보이게 설정해주는 등의 추가 설정이 필요합니다.
Margin*	각 시리즈리스트에 할당된 영역과 GraphArea 사이의 공간을 정의합니다. 아래서 그림으로 알아봅니다. 마진은 정수 단위로 설정이 가능하고 기본값은 0, 최대 100 까지만 설정 가능합니다.
Radar	방사형차트(레이다차트)의 특징들을 정의하는 속성입니다. 앞서 설명한대로 레이다차트는 조금 특수한 차트여서 시리즈리스트 레벨에서 기본적인 정의들을

	마칩니다. 단, 시리즈레벨에서도 세부 설정을 할 수 있습니다. 뒤에서 다시 알아봅니다.
SeriesCollection	시리즈리스트에 추가될 시리즈들을 관리하는 컬렉션 속성입니다.
StackType	차트의 스택형을 정의하는 속성으로 사용자는 100% Full 스택형일 경우만 설정합니다.
Transparency	전체 차트군의 투명도를 조절합니다.

(표2 - 시리즈리스트 속성)

(1)축 요소(AxisFactors)

히포차트에서 가장 중요한 속성 중에 하나인 축 요소 속성에 대해 알아보겠습니다. 우선 좌표 영역의 구성은 아래 그림과 같이 나타낼 수 있습니다.



속성	설명
AxisItems	추가된 축 리스트를 관리합니다. 멀티축에 대해선 뒤에서 다시 자세히 알아봅니다.
CoordinateRectangle	실질적인 좌표를 그리는 사각형을 의미합니다. 사용자가 설정 사항은 아닙니다.
DisplayItemType	수치표시시 표시되는 내용을 설정하고 가져온다. Figure 수치 Name 이름 Figure_Name 수치 + 이름

Orientation	X, Y 축의 방향을 설정합니다. BottomLeft TopLeft BottomRight TopRight
ThreeDstepDepth	차트를 입체로 설정할 경우 그 넓이를 설정합니다. 넓이란 입체의 세로 간격을 의미합니다.
X	x축의 양의 방향 끝 포인트로서 자동설정됩니다.
XAxis*	좌표의 x축(이름축)을 설정합니다. 아래에서 자세히 합니다.
XMinus	x축의 음의 방향 끝 포인트로서 자동설정됩니다.
Y	y축의 양의 방향 끝 포인트로서 자동설정됩니다.
YAxis*	좌표의 y축(값축)을 설정합니다. 아래에서 자세히 합니다.
YMinus	y축 음의 방향 끝 포인트로서 자동설정됩니다.
Zero	좌표축의 가운데 (0.0) 포인트로서 자동설정됩니다.

(표3 - axisfactor 속성)

Axis

아래는 축을 나타내는 Axis 클래스의 속성들입니다. 축 요소 클래스의 두 가지 중요한 속성인 YAxis와 XAxis는 이 클래스의 객체로서 실질적인 좌표축의 설정을 담당합니다.

AnalysisItems	y좌표가 있는 반대편에 각종 통계 아이템을 표시할 수 있습니다. *AnalysisCategory Max 최대값 Min 최소값 Average 평균 Sum 총합계 Variance 분산 StandardDeviation 표준편차 Median 중앙값
Areas	축 영역 마커를 관리하는 컬렉션 객체입니다. 추가하고자하는 AxisArea 객체를 이 속성에 추가해서 반영

	합니다.
AutoScaleBreak	스케일브레이크를 자동으로 설정할 것인가 여부를 설정합니다. (이 속성은 비활성화 되있습니다.)
AxisScaleRanges	ScaleRange 객체를 관리하는 컬렉션 객체입니다. 추가하고자하는 ScaleRange 객체를 이 속성에 추가해서 반영합니다.
AxisType	축의 종류(이 속성은 사용자 설정사항이 아닙니다.) Xtype x축 타입 Ytype y축 타입
AxisUnitText	단위를 나타내는 문자열(예: 원, km, $)을 설정합니다. 이 값은 IsShowUnitText 값이 true일 때만 반영되며 설정 시 Y축의 최상단 단위 값이 보이지 않게 됩니다. (단, 이 기능은 Y축에만 적용됩니다.)
CultureInfo	축의 문화권 정보를 반영합니다. 기본값은 ko-KR입니다.
DataType	축에 표시되는 데이터의 타입을 설정합니다. y축일 경우는 무조건 대부분 number 타입이며, 간트차트일 경우 datetime이 될 수 있습니다. 또한 X축을 number 형으로 설정할 경우 분산형, 함수의 그래프 등을 표현할 수 있으며 각종 마커들과 수동 조절기능을 활용할 수 있습니다. Number 숫자 타입(Y축 기본값) DateTime 날짜 타입 String 문자열 타입(X축 기본값)
DateTimeFormat	축의 라벨에 표시되는 날짜형 데이터의 포맷을 설정합니다.(예, yyyy-MM-dd, m, yy-MM-dd) 단, 이 속성은 간트차트일 경우에만 반영됩니다.
Decimalpoint	축의 라벨에 표시되는 숫자의 소수점 자리수를 설정합니다. (예, 2이면 소수 둘째자리까지 표시)
Direction	축의 방향을 설정합니다. (이 속성은 사용자 설정사항이 아닙니다.) Left 왼쪽 Right 오른쪽

	Top 위 Bottom 아래
ExtraTicks	추가 AxisTick 객체를 관리하는 컬렉션 객체입니다. 이 객체는 축에 표시되는 각 단위값을 사이에 눈금을 더 표시하기를 원할 경우 사용하면 유용합니다.
FigureFormat	축의 숫자 포맷을 설정합니다. 이 속성은 시리즈리스트의 CultureInfo 객체와 연관하여 사용할 것을 권장합니다. None 없음 Number 숫자형태 Percent 퍼센트형태 Currency 통화형태 FixedPoint 소수점형태 Exponential 지수형태 Custom 사용자지정
Font	축 라벨리스트들에 공통으로 적용될 Font 를 설정합니다. ※ 라벨리스트 전체에 적용되므로 개별 설정을 할 수 없습니다. 이럴 경우는 AxisTick을 이용하여 눈금을 재 디자인할 수 있습니다.
ForeColor	축 라벨리스트들에 공통으로 적용될 글자색상을 설정합니다. ※ 라벨리스트 전체에 적용되므로 개별 설정을 할 수 없습니다. 이럴 경우는 AxisTick을 이용하여 눈금을 재 디자인할 수 있습니다.
Interval	축의 눈금 간격을 설정합니다. x축의 경우는 직접 설정이 가능하고 y축의 경우는 IsAutoSetting = false 일 경우만 반영됩니다.
IsAutoSetting	축의 단위값을 자동으로 설정할 것인가 여부를 설정합니다. y축의 경우는 직접 설정이 가능하나 x축일 경우는 datatype에 number 일 경우만 반영됩니다. ※ 이 기능은 주의를 요합니다.
IsShowTick	축에 보이는 눈금들을 보일 것인가 여부를 설정합니

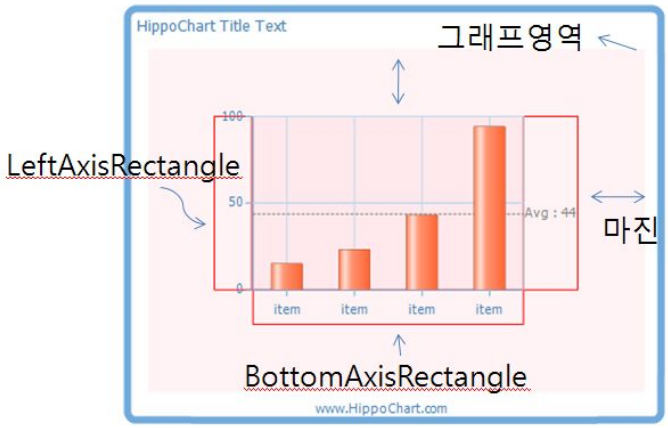
	다.
IsShowUnitText	y축의 단위 문자열을 표시할 것인가 여부를 설정합니다.
IsVisibleFigures	축의 라벨리스트를 표시할 것인가 여부를 설정합니다.
itemCount	축의 총 아이템 개수를 가져옵니다.(읽기전용)
LabelFormat	(사용자 설정사항 아님)
LabelItems	(사용자 설정사항 아님)
Line	축 라인에 대해 설정합니다. 각 축의 라인을 굵게 한다거나 다른 색상으로 수정할 수 있습니다. http://hippochart.tistory.com/166 강좌와 연계하면 좌표축의 디자인에 대해 더욱 쉽게 접근할 수 있습니다.
Markers	AxisMaker 객체를 관리하는 컬렉션 객체입니다. 원하는 마커 생성 후 이 객체에 추가하여 반영합니다.
MaxUnitDateTimeValue	날짜형 Y축을 가진 간트차트일 경우의 최대단위값을 설정합니다. (IsAutoSetting = false 일 경우만 반영됩니다.)
MaxUnitValue	일반 숫자형 Y축 차트의 최대단위값을 설정합니다. (IsAutoSetting = false 일 경우만 반영됩니다.)
MinUnitDateTimeValue	날짜형 Y축을 가진 간트차트일 경우의 최소단위값을 설정합니다. (IsAutoSetting = false 일 경우만 반영됩니다.)
MinUnitValue	일반 숫자형 Y축 차트의 최소단위값을 설정합니다. (IsAutoSetting = false 일 경우만 반영됩니다.)
Points	(사용자 설정사항 아님)
SmallInterval	(미반영)
Ticks	AxisTick 객체를 관리하는 컬렉션 객체입니다. 원하는 눈금 설정 후 이 객체에 추가해서 반영합니다.
TitleFormat	축 타이틀 포맷을 설정하고 가져옵니다.
TitleLabel	축의 타이틀 라벨을 설정하고 가져옵니다.
UnitPixel	축 한 단계 길이를 설정하고 가져옵니다. (읽기전용)
Visible	축 자체를 보일 것 인가 여부를 설정하고 가져온다.
X	x축의 양의 방향 끝 포인트로서 자동설정됩니다.
XMinus	x축의 음의 방향 끝 포인트로서 자동설정됩니다.
Y	y축의 양의 방향 끝 포인트로서 자동설정됩니다.
YMinus	y축 음의 방향 끝 포인트로서 자동설정됩니다.
Width	축이 그려지는 공간의 짧은 넓이를 가져옵니다.(읽기전용)

Zero	좌표축의 가운데 (0.0) 포인트로서 자동설정됩니다.

(표4 - Axis 클래스 속성)

※축의 기능은 위와 같이 상당히 복잡하고 다양하므로 실전 편에서 다시 자세히 알아봅니다.

(2)그래프 영역(GraphArea)와 마진(Margin)



위 이미지는 그래프 영역과 마진과의 관계와 그래프영역의 세부 공간을 보여줍니다. 코드작성 시 핵심적인 사항은 아니지만 시리즈리스트 내부 공간의 구조를 알고 있다면 보다 많은 응용이 가능할 것입니다.

GraphArea

속성	설명
BackColor	그래프 영역의 배경색
BorderColor	그래프 영역의 외곽 라인 색
BottomAxisRectangle	그래프 영역 하단 축 영역. 사용자 설정사항 아님
GraphRectangle	그래프 영역 가운데 영역. 사용자 설정사항 아님
Grid*	그래프 영역 그리드의 특징을 설정합니다.
LeftAxisRectangle	그래프 영역 왼쪽 축 영역. 사용자 설정사항 아님
RightAxisRectangle	그래프 영역 오른쪽 축 영역. 사용자 설정사항 아님
TopAxisRectangle	그래프 영역 상단 축 영역. 사용자 설정사항 아님

(표5 - 그래프영역 속성)

Grid*

속성	설명
GridDirection	그리드의 방향을 설정합니다. Both 양쪽(기본값) Horizontal 가로방향 Vertical 세로방향 None 그리드 없음
GridLine	그리드의 라인 객체를 설정합니다. Line 객체는 히포차트를 구성하는 기초 클래스로서 뒤에서 다시 자세히 알아보겠습니다.
Interval	그리드의 라인을 그리는 간격을 조절합니다. 이 값은 1차적으로 x축의 interval을 상속받아서 그리게 되어 있는데, 이 값을 사용자가 설정할 시에는 x축 간격 값에 우선하게 됩니다.
ScaleBreakLine	스케일 브레이크의 라인 객체를 설정합니다. 스케일 브레이크부분에서 다시 자세히 합니다.
ZeroLineColor	좌표에 음수가 있을 경우 나타나는 제로 라인의 색상을 정의합니다. 기본값은 Red입니다.

(표6 - 그리드 속성)

2. 시리즈리스트 이벤트

시리즈리스트에는 4가지 이벤트를 지원하는데 특정 속성이 변경 될 경우 발생하는 이벤트를 통해 적절한 로직을 삽입할 수 있습니다.

이벤트	설명
Chart3DShowChanged	3D입체 여부가 변경될 경우 발생하는 이벤트입니다.
ChartMarginChanged	마진이 수정될 경우 발생하는 이벤트입니다.
ChartTransParentChanged	투명도가 변경될 경우 발생하는 이벤트입니다.
ChartTypeChanged	차트 종류가 변경될 경우 발생하는 이벤트입니다.

(표7 - 시리즈리스트 이벤트)

이벤트의 사용 방법은 시리즈리스트가 생성되는 load 이벤트 혹은 사용자 생성 이니셜라이징 메소드 등에서 해당 이벤트 객체에 핸들러를 등록해 주시고 생성된 이벤트 핸들러에 적절한 코드를 삽입해주시면 되겠습니다.

```
private void GenerateEvent()
{
    SeriesList slist = new SeriesList();

    slist.Chart3DShowChanged += new EventHandler(slist_Chart3DShowChanged);
    slist.ChartMarginChanged += new EventHandler(slist_ChartMarginChanged);
    slist.ChartTransParentChanged += new EventHandler(slist_ChartTransParentChanged);
    slist.ChartTypeChanged += new EventHandler(slist_ChartTypeChanged);
}

 void slist_ChartTypeChanged(object sender, EventArgs e)
 {

 }

 void slist_ChartTransParentChanged(object sender, EventArgs e)
 {

 }

 void slist_ChartMarginChanged(object sender, EventArgs e)
 {

 }

 void slist_Chart3DShowChanged(object sender, EventArgs e)
 {

 }
```

TIP.

이벤트 핸들러를 쉽게 등록하는 방법은 이벤트에서 "+=" 을 작성하면 아래와 같이 탭키를 누르라는 메시지가 나오는데 TAB 키를 두 번 클릭해서 자동으로 위와 같은 이벤트 핸들러를 삽입할 수 있다.

```
slist.ChartTypeChanged +=
                        new EventHandler(slist_ChartTypeChanged);   (<TAB> 키를 눌러 삽입합니다.)
```

3. 시리즈리스트의 추가, 제거

제 아무리 멋지게 시리즈리스트를 설정했다 하더라도 차트 객체에 추가하지 않으면 차트를 그릴 수 없겠죠?

시리즈리스트는 차트 객체의 SeriesListDictionary라는 컬렉션 객체에서 관리합니다.

C#

```
SeriesList sList = new SeriesList();

this.hHippoChart1.SeriesListDictionary.Add(sList);
this.hHippoChart1.DrawChart();
```

VB

```
Dim sList As New SeriesList()

Me.HHippoChart1.SeriesListDictionary.Add(sList)
Me.HHippoChart1.DrawChart()
```

일반적인 방법으로는 위와 같은 방식으로 추가를 할 수 있습니다. 차후 시리즈리스트를 모두 제거하여 차트를 초기화해야할 경우나 계속적으로 호출되는 메소드를 구성할 경우 아래와 같은 방식으로 처리할 수 있습니다.

```
this.hHippoChart1.SeriesListDictionary.Clear();
```

또는, 시리즈리스트가 1개만 추가가 되는 경우라면 아래와 같이 간단하게 호출할 수도 있습니다. 아래 방법은 DrawChart 메소드의 오버로드를 이용한 방식으로 내부적으로는 위와 동일합니다. 단, 이 방법은 내부적으로 제거를 하고 추가하므로 반복 호출 메소드를 만들어 사용 시 별도로 제거할 필요가 없습니다.

```
SeriesList sList = new SeriesList();
this.hHippoChart1.DrawChart(sList);
```

시리즈(Series)

시리즈란, 하나의 차트를 그리기 위한 최소한의 단위라고 정의할 수 있습니다. 라인차트라면 하나의 일련의 포인트를 연결한 라인, 파이차트라면 한 개의 원을 의미합니다. 실질적으로 차트를 그리는 요소이므로 Line, Column, Points 등의 기초클래스들과 시리즈색상 그리고 비좌표 요소들(UnAxisFactor, Gauge 등)이 포함되어 있습니다.

1. 시리즈 속성

속성	설명
AxisIndex	시리즈가 사용하는 축을 나타내는 인덱스 값을 설정합니다. 기본 축을 사용할 경우 설정을 하지 않거나 0으로 설정하면 되고 추가된 축으로 설정할 경우 추가된 첫 번째 축부터 1, 2 ... 순으로 설정합니다. 멀티축 사용에 관해서는 뒤에서 다시 자세히 합니다.
ChartType	시리즈 차트 종류를 설정합니다. 시리즈레벨에서 차트타입을 설정하면 시리즈리스트 레벨에서 설정한 차트 타입을 오버라이드합니다. 반대로 설정을 안 할 시에는 시리즈리스트에서 설정한 값을 그대로 사용합니다. 단, 이 설정에서 주의할 점은 반드시 시리즈리스트에서 설정한 차트타입과 같은 차트군의 차트를 설정해야한다는 점입니다. (차트군에 대해선 앞에서 설명)
Column	대표적인 기초 클래스 중에 하나입니다. 컬럼 차트류(컬럼, 원기둥, 콘, 바차트)의 기둥 디자인을 관리합니다. DesignType 컬럼 디자인 종류를 가져오고 설정합니다. ImagesBarPath 컬럼 이미지를 가져오고 설정합니다. Width 컬럼 넓이를 가져오고 설정합니다. WidthType 컬럼의 넓이를 가져오고 설정합니다.

FigureStringFormatFlags	그래프에서 수치를 표시할 때 그 모양을 수직으로 할 것인가 수평으로 할 것인가를 설정합니다. 기본값은 수평으로 설정하지 않으면 되고 공간이 좁거나 하여 수직으로 표시해야할 경우는 sr.FigureStringFormatFlags = StringFormatFlags.DirectionVertical; 와 같이 표시하면 되겠습니다.
FigureTextLocation	그래프 상에서 수치가 표시되는 상대적 위치를 설정합니다. 이 속성을 조절함으로써 여러 가지 이유로 겹치는 수치들을 효과적으로 모두 보이게 설정할 수 있습니다. Default 기본값 Top 기본값 보다 위 Bottom 기본값 보다 아래 Left 기본값 보다 왼쪽 Right 기본값 보다 오른쪽
ForeColor	시리즈의 글자색상으로 범례에 보이는 시리즈이름 표시 색상을 설정합니다.
Gauge	게이지차트의 다양한 특징들을 정의합니다. BackColor 게이지 배경색을 설정하고 가져옵니다. BorderColor 게이지 외곽 라인 색상을 설정하고 가져옵니다. Font 게이지 수치 폰트를 설정하고 가져옵니다. GaugeType 게이지 타입을 설정하고 가져옵니다. Grid 게이지 그리드를 설정하고 가져옵니다. InnerBackColor 게이지 내부 그리드 배경색을 설정하고 가져옵니다. IsFillGrid 게이지 그리드를 색칠할 것인가 여부를 설정하고 가져옵니다. IsShowBorderLine 게이지의 외곽라인을 그릴 것인지 여부를 설정하고 가져옵니다. Margin 게이지 마진을 설정하고 가져옵니다. ShapeType 게이지 모양 종류를 설정하고 가져옵니다.
items	SeriesItem 을 관리하는 컬렉션 객체입니다.
Label	시리즈의 라벨을 설정합니다. 시리즈에서 라벨은 비좌표차트에만 적용되는데 현재 레이다차트와 게이

	지차트의 라벨에 사용됩니다.
LegendVisible	시리즈를 범례에 표시할 것인지 여부를 설정합니다.
Line	대표적인 기초 클래스 중의 하나로 시리즈의 차트 타입이 라인류(라인, 스플라인, 카기) 차트일 경우 그 라인의 특징을 설정합니다.
Name	시리즈의 이름으로 범례에 표시됩니다.
Points	대표적인 기초 클래스 중의 하나입니다. 포인트가 표시되는 차트들(라인, 스플라인, 스케터) 인 경우 그 포인트 모양과 크기 등을 설정합니다.
RadarType	방사형차트(레이다차트)의 각 시리즈별 그려지는 모양을 설정합니다. Point 포인트 Line 포인트와 그 연결선 Area 포인트와 그 면적
SeriesColor	시리즈의 그래프 색상을 설정합니다. 이 색상은 좌표 성질의 차트에만 적용됩니다. (비좌표류 차트는 시리즈아이템 레벨에서 설정 가능합니다)
Transparency	시리즈의 그래프 색상의 투명도를 조절합니다. 설정하지 않을 시에는 시리즈리스트 레벨의 투명도를 그대로 사용합니다.
AreaType	시리즈리스트 전체를 영역형 차트들로 그릴 것인가 여부를 나타냅니다. 히포차트에서 영역형이란 라인 영역형을 의미하는 것으로 라인 차트에만 적용되는 속성입니다. (3.1 버전부터 추가)
UnAxisFactor*	비좌표류 차트들의 특징들을 관리하는 속성입니다. 다음 페이지에서 자세히 알아봅니다.

(표8 - 시리즈 속성)

UnAxisFactor

속성	설명
Angles	파이차트를 그리는 각도들을 관리하는 컬렉션. 사용자 설정 사항 아님.
Axis	비좌표 차트에 존재하는 축을 설정합니다. 게이지차

	트와 방사형 차트에 적용됩니다.
Decimalpoint	비좌표 차트의 표시되는 수치의 소수점을 설정합니다.
InnerRectangles	비좌표 차트가 그려지는 내부 공간을 의미합니다. 사용자 설정 사항은 아닙니다.
ItemNames	비좌표 차트의 각 시리즈아이템 이름을 관리하는 컬렉션입니다. 사용자 설정사항 아님.
Percentages	비좌표 차트의 각 시리즈아이템의 값에 대한 퍼센트 값을 관리하는 컬렉션입니다. 사용자 설정사항 아님.
PercentLineDisplayValue	그래프에 수치를 표시할 때 보이는 한계 퍼센트 값을 지정합니다. 예를 들어 2라고 설정할 경우 1.9%인 값은 수치보이기를 해도 그래프 상에 표시되지 않습니다. 파이, 피라미드차트에 적용됩니다.
Pie3DDepth	파이차트의 입체 두께를 조절합니다. (기본값 35)
PieFigures	비좌표 차트의 실제 값을 관리하는 컬렉션. 사용자 설정사항 아님
PieTypes	파이차트의 모양을 정의합니다. (미지원) Normal 일반 형태 TwoDemention 2D 형태 Slice 각 파이가 쪼개진 형태
TextLocation	파이, 피라미드 차트의 수치 표시 위치를 설정합니다. Auto 자동으로 계산해서 표시 Inner 그래프 안쪽에 표시 Outer 그래프 바깥으로 라인과 연결해서 표시
UnAxisDisplayItemType	파이, 피라미드 차트에 표시되는 수치의 내용을 설정합니다. Percent 퍼센트 Figure 수치 Name 이름 Angle 각도 Figure_Percent 수치(퍼센트) Percent_Name 퍼센트(이름)

	Figure_Name 수치(이름) Angle_Name 각도(이름)

(표9 - UnAxisFactor 속성)

2. 시리즈 이벤트

이벤트	설명
ChartTransParentChanged	시리즈의 투명도가 변경될 경우 발생합니다.
ChartTypeChanged	시리즈의 차트 종류가 변경될 경우 발생합니다.

(표10 - 시리즈 이벤트)

이벤트의 사용 방법은 시리즈가 생성되는 이벤트 혹은 사용자 생성 메소드 등에서 해당 이벤트 객체에 핸들러를 등록해 주시고 생성된 이벤트 핸들러에 적절한 코드를 삽입해주시면 되겠습니다.

```
private void GenerateEvent()
{
    Series sr = new Series();
    sr.ChartTransParentChanged += new EventHandler(sr_ChartTransParentChanged);
    sr.ChartTypeChanged += new EventHandler(sr_ChartTypeChanged);
}

void sr_ChartTypeChanged(object sender, EventArgs e)
{

}

void sr_ChartTransParentChanged(object sender, EventArgs e)
{

}
```

3. 시리즈 추가, 제거

시리즈는 앞서 설명했듯이 시리즈리스트의 SeriesCollection라는 컬렉션으로 관리합니다.

```
SeriesList sList = new SeriesList();

Series sr = new Series();
sList.SeriesCollection.Add(sr);

this.hHippoChart1.DrawChart();
```

만약 시리즈리스트를 먼저 추가했다면 아래와 같이 SeriesListDictionary의 인덱스로 접근하여 추가할 수도 있습니다.

C#

```
SeriesList sList = new SeriesList();

Series sr = new Series();

this.hHippoChart1.SeriesListDictionary.Add(sList);

this.hHippoChart1.SeriesListDictionary[0].SeriesCollection.Add(sr);
this.hHippoChart1.DrawChart();
```

VB

```
Dim sList As New SeriesList()

Dim sr As New Series()

Me.HHippoChart1.SeriesListDictionary.Add(sList)

Me.HHippoChart1.SeriesListDictionary(0).SeriesCollection.Add(sr)
Me.HHippoChart1.DrawChart()
```

시리즈아이템(SeriesItem)

시리즈아이템이란, 차트를 구성하는 가장 작은 단위의 객체라고 정의할 수 있습니다. 라인, 스플라인, 스케터, 방사형 차트일 경우에는 한 점을 의미할 것이고, 바, 간트, 컬럼, 원기둥 차트일 경우는 하나의 막대(기둥)을 의미할 것입니다. 또

한, 파이 차트 같은 경우는 한 조각의 부채꼴을 의미하겠습니다.

1. 시리즈아이템 속성

시리즈아이템은 앞서 화합물과의 비유에서 말했듯이 차트를 구성하는 더 이상 쪼개지지 않는 '원자'와 같습니다. 그리고 하나의 점을 의미하므로 points 라는 객체를 소유할 것이고, 하나의 막대 기둥을 의미하므로 column 이라는 객체를 소유할 것입니다. 또한, 수치를 표시할 것이므로 Label이라는 객체를 가지고 있으며 풍선도움말과 같은 객체도 가지고 있습니다.

시리즈아이템을 다루는데 주의해야할 점이 있다면 모든 속성의 값들이 비어있다는 것입니다. 이는 시리즈가 기본적으로 시리즈리스트의 값을 상속받듯이 시리즈아이템 역시 시리즈의 상속을 받기 위함입니다. Points, Column, Balloon 같은 클래스 속성들이 모두 null로 초기화가 되어 있으므로 특정 시리즈아이템을 설정하고자 한다면 반드시 인스턴스를 생성한 후에 작업하셔야 됩니다.

속성	설명
Balloon	시리즈아이템에 풍선 모양의 텍스트 공간을 만들어 줍니다. 이 속성은 반드시 인스턴스를 생성해야합니다. BackColor 풍선의 배경색을 설정하고 가져옵니다. BalloonType 풍선 도움말 모양을 설정하고 가져옵니다. Height 풍선의 세로길이을 설정하고 가져옵니다. HeightType 풍선도움말을 표시할 위치를 설정하고 가져옵니다. IsShadow 풍선의 그림자를 표시할것인가 여부를 설정하고 가져옵니다. Label 풍선 라벨을 설정하고 가져옵니다. Line 풍선 라인을 설정하고 가져옵니다. TextLocation 풍선 내부 글자의 정렬을 설정하고 가져옵니다. Width 풍선의 가로길이을 설정하고 가져옵니다. X 풍선의 Left위치을 설정하고 가져옵니다. Y 풍선의 top 위치을 설정하고 가져옵니다.
Column	차트 타입이 컬럼, 원기둥, 콘, 바, 간트 차트일 경우 해당하며 사격형의 기둥을 디자인합니다. 이 속성은 반드시 인스턴스를 생성해야합니다.
FigurePoint	수치가 표시되는 위치 값을 가져옵니다. 이 속성은

	사용자 설정사항이 아닙니다. ※이 속성을 통해 위치값을 알고자 한다면 DrawChart() 메소드를 호출한 후에 그 값을 가져올 수 있습니다.
IsShowFigureText	시리즈아이템의 수치 값을 그래프에 표시할 지 여부를 설정합니다. (기본값 false)
ItemColor	시리즈아이템의 색상을 설정합니다. 설정하지 않을 경우는 ColorSystem 이라는 클래스를 통해 자동 설정됩니다.
Label	시리즈아이템의 표시되는 수치 글자를 나타내는 속성입니다. 글자의 크기, 색상, 폰트 등을 수정할 수 있으며 Text 속성은 적용되지 않습니다.
Name	시리즈아이템의 이름을 설정합니다. 이름이란 좌표 차트일 경우에는 X축에 표시되는 텍스트를 의미하고, 비좌표 차트일 경우에는 파이의 한 조각, 피라미드의 한 칸, 게이지차트의 값 하나를 의미하므로 범례의 이름에 표시되며 비좌표 차트일 경우 이 속성은 반드시 설정해야합니다.
Points	차트 타입이 라인, 스플라인, 스케터, 레이더인 경우에 하나의 점(포인트)를 설정합니다. 포이트는 모양, 굵기 등을 설정할 수 있으며 굵기의 기본값은 1입니다.
XValue	x축이 Number 형일 경우에 입력하는 속성입니다.
YDateTimeValue	차트 타입을 간트차트로 설정했을 경우 종료 날짜를 의미합니다.
YStartDateTimeValue	차트 타입을 간트차트로 설정했을 경우 시작 날짜를 의미합니다.
YStartValue	차트 타입을 간트차트로 설정했을 경우 시작 값을 의미합니다.
YValue	시리즈아이템의 수치 값을 설정합니다. 간트차트일 경우는 종료 값을 의미합니다.

(표11 - 시리즈아이템 속성)

2. 시리즈아이템 추가, 제거

시리즈아이템은 시리즈의 items 라는 컬렉션 속성으로 관리합니다.

C#

```
SeriesList sList = new SeriesList();
Series sr = new Series();
sList.SeriesCollection.Add(sr);

SeriesItem item = new SeriesItem();
sr.items.Add(item);

this.hHippoChart1.SeriesListDictionary.Add(sList);
this.hHippoChart1.DrawChart();
```

VB

```
Dim sList As New SeriesList()
Dim sr As New Series()
sList.SeriesCollection.Add(sr)

Dim item As New SeriesItem()
sr.items.Add(item)

Me.HHippoChart1.SeriesListDictionary.Add(sList)
Me.HHippoChart1.DrawChart()
```

사실 시리즈아이템의 추가는 테스트 개발이 아니라면 위와 같이 일일이 하나씩 추가할 일은 없을 것입니다. DrawChart() 메소드를 이용하거나 히포엔진(HippoEngine) 클래스를 이용해 간편하게 데이터를 시리즈아이템화 할 수 있기 때문입니다.

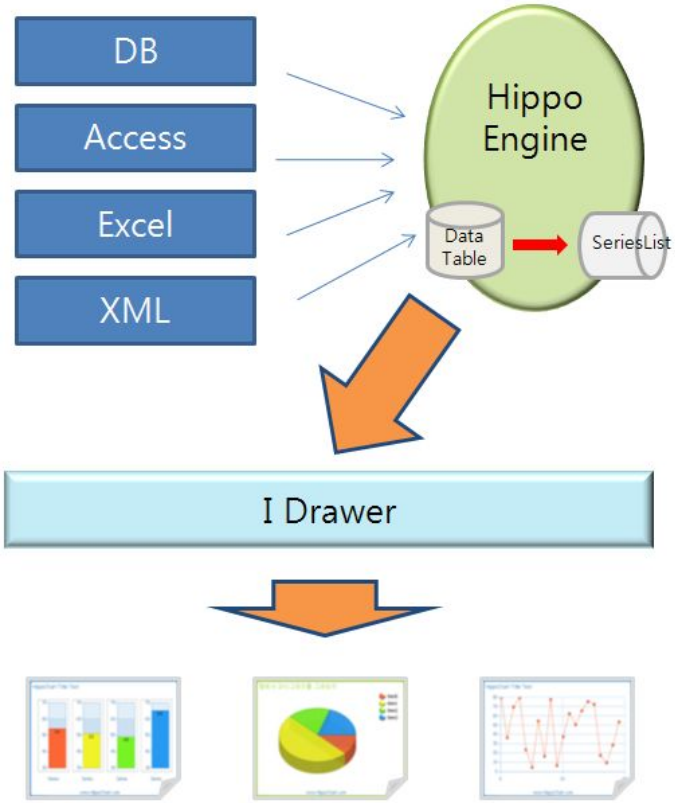
시리즈아이템의 제거는 컬렉션 객체에서의 제거이므로 아래와 같이 동일한 방법입니다.

```
sr.items.Clear();
```

다만, 히포차트에서는 다양한 차트 초기화 메소드들을 제공하므로 위와 같은 코드를 사용하기보다는 ResetData()와 같은 메소드를 사용하면 더욱 편리합니다. 차트 초기화에 대해서는 뒤에서 다시 알아보겠습니다.

히포엔진을 사용하여 차트 데이터를 만들어 보자.

히포엔진(HippoEngine)은 사용자의 데이터를 차트를 그리기위한 객체로 만들어 주는 Helper class입니다. 아래 그림을 통해 히포엔진의 프로세스를 이해하세요.

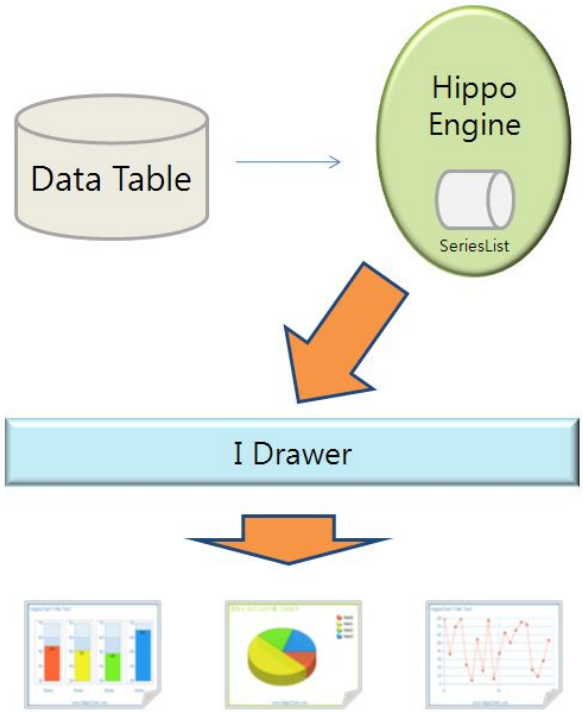


위 그림에서 보이듯이 히포엔진은 데이터베이스, 엑세스, 엑셀, XML 등의 다양한 유저 데이터 소스를 내부적으로 DataTable로 변경한 후 개발자에게 시리즈리스트를 반환해줍니다. 사용자는 그 시리즈리스트를 통해 적절한 디자인을 입히고 차트를 완성하게 됩니다.

유저 데이터 소스와 시리즈리스트 사이에 매개체로 DataTable이 생성된다고 했는데 이는 DataTable을 직접 넘겨도 됨을 의미합니다. 사실 다양한 개발 환경 속에서 히포엔진으로 하여금 데이터베이스를 엑세스하게 한다면 프로젝트의 티어 체계를 무너뜨려 보안에 문제가 될 수 있고 일관성 없는 DB엑세스가 될 수 있으므로 다음 페이지에 나오는 그림과 같은 형태가 가장 많이 사용될 것입니다.

엑셀, 엑세스 등의 다른 데이터 소스들은 히포엔진을 통해 매우 유용하게 사용할 수 있으며 역시 개발자의 별도 엑세스를 통해 DataTable을 넘겨주어도 무방합니다.

히포엔진은 차트 데이터를 만들기 위한 Helper class라고 하였습니다. 이 말은 도와준다는 의미이므로 반드시 써야한다는 의미는 아닙니다. 즉, 다음과 같은 코드를 통해 사용자가 직접 히포엔진이 하는 역할을 대신해도 무방합니다. 히포엔진이 하는 일도, 개발자가 해야하는 일도 결국은 시리즈리스트를 만들어 히포차트 객체에 넘겨주는 일이니까요.



C#

```
DataTable userTAble = Conn.GetData();
SeriesList sList = new SeriesList();
for (int c = 1; c < userTAble.Columns.Count; c++)
{
     Series sr = new Series();

     for (int i = 0; i < userTAble.Rows.Count; i++)
     {
          SeriesItem item = new SeriesItem();
          item.YValue                                                                    =
float.Parse(userTAble.Rows[i][userTAble.Columns[c].ColumnName].ToString());
item.Name = userTAble.Rows[i]["name"].ToString();

           sr.items.Add(item);
      }
      sList.SeriesCollection.Add(sr);
}
```

VB

```
Dim userTAble As DataTable = Conn.GetData()
' 사용자데이터획득
 Dim sList As New SeriesList()

 For c As Integer = 1 To userTAble.Columns.Count - 1
     Dim sr As New Series()
     sr.ChartType = ChartType.Circular

     For i As Integer = 0 To userTAble.Rows.Count - 1
          Dim item As New SeriesItem()
          item.YValue                                                                    =
Single.Parse(userTAble.Rows(i)(userTAble.Columns(c).ColumnName).ToString())
item.Name = userTAble.Rows(i)("name").ToString()

           sr.items.Add(item)
      Next

     sList.SeriesCollection.Add(sr)
 Next
```

이상으로 히포엔진의 기본 개념과 동작 원리를 알아보았습니다. 다음에는 히포

엔진에서 지원하는 다양한 메소드와 속성 들을 알아보겠습니다.

1. HippoEngine 속성

속성	설명
ConnectionString	데이터소스에 대한 연결문자열을 정의합니다. 엑셀, 엑세스, 데이터 베이스에 적용됩니다.
DataType	히포엔진이 엑세스해야하는 데이터 소스의 종류를 설정합니다. Default 기본값(MS SQL) Access Office Access Excel Office Excel Xml Xml Oracle 오라클 MySql MySql
ErrorMessage	에러메세지를 반환합니다.
Query	데이터소스에게 던질 질의를 설정합니다. 데이터베이스, 엑셀, 엑세스에 적용됩니다.
XmlPath	데이터소스가 XML일 경우 그 파일의 경로를 설정합니다.

(표12 - 히포엔진 속성)

2. HippoEngine 메소드

히포엔진에서 두 가지 중요한 메소드를 가지고 있는데 일반적인 데이터를 변환하기 위한 GetSeriesList()와 간트차트를 그릴 때 사용하는 GetSeriesListOfGantt()입니다.

HippoEngine.GetSeriesList Method

이 메소드는 무려 14개의 메소드로 오버로드되어 있습니다. 아래 표를 참고하여 자신에게 맞는 적절한 메소드를 사용할 수 있어야겠습니다.

메소드	설명
GetSeriesList (string name, params string[] values)	연결문자열, 데이터소스타입 등을 속성으로 설정한 후에 호출하는 메소드

	입니다.
GetSeriesList (AxisDataType datatype, string name, params string[] values)	위와 동일하고 AxisDataType가 추가되어 있습니다. 이 메소드는 x축을 숫자형으로 설정하고자 할 때 호출하면 되겠습니다.
GetSeriesList (bool IsShowFigures, string name, params string[] values)	위와 동일하고 수치표시여부 파라미터가 추가되어 있습니다.
GetSeriesList (AxisDataType datatype, bool IsShowFigures, string name, params string[] values)	위와 동일하고 AxisDataType와 수치표시여부 파라미터가 모두 추가되어 있습니다. x축을 숫자형으로 설정하고 수치를 표시하고자 할 경우 사용하면 유용합니다.
GetSeriesList(DataSourceType dataType, string name, params string[] values)	속성 설정으로 다양하게 사용할 수도 있으나 주로 데이터소스가 xml일 경우 사용하면 유용한 메소드입니다.
GetSeriesList (DataSourceType dataType, bool IsShowFigures, string name, params string[] values)	위와 동일하고 수치표시를 할 경우 사용합니다.
GetSeriesList (DataTable source, string name, params string[] values)	가장 많이 사용 빈도가 높을 것으로 보이는 메소드로 데이터소스와 이름, 값 배열을 파라미터로 가집니다.
GetSeriesList (DataTable source, AxisDataType datatype, string name, params string[] values)	위와 동일하며 x축이 숫자형일 경우 사용합니다.
GetSeriesList (DataTable source, bool IsShowFigures, string name, params string[] values)	위와 동일하며 수치를 표시할 경우 사용합니다.
GetSeriesList (DataTable source, bool IsShowFigures, AxisDataType datatype, string name, params string[] values)	위와 동일하며 수치를 표시하고 x축을 숫자형으로 설정할 경우 사용합니다.
GetSeriesList (string[] values)	값 배열만 넘겨 간단하게 차트를 그릴 경우 사용합니다.
GetSeriesList (string[] values, bool IsShowFigures)	위와 동일하고 수치를 표시할 수 있습니다.
GetSeriesList (string[] names, string[] values)	값 배열과 이름 배열을 넘겨 간단하게 차트를 그릴 경우 사용합니다.
GetSeriesList (string[] names, string[] values, bool IsShowFigures)	위와 동일하고 수치를 표시할 수 있습니다.

(표13 - 히포엔진 GetSeriesList 메소드)

※위 표는 같은 맥락의 메소드들을 구분하기 위해 그룹별로 건너서 배경색 처리하였습니다.

※진하게 표시된 부분은 사용 빈도가 높은 중요한 메소드입니다.

HippoEngine.GetSeriesListOfGantt Method

이 메소드는 간트차트를 그릴 경우 사용하는 메소드로 4개로 오버로드되어 있습니다.

메소드
GetSeriesListOfGantt (AxisDataType dataFormat, string name, params string[] values)
GetSeriesListOfGantt (DataSourceType dataType, AxisDataType dataFormat, string name, params string[] values)
GetSeriesListOfGantt (DataTable source, AxisDataType dataFormat, string name, params string[] values)
GetSeriesListOfGantt (DataTable source, bool IsShowFigures, AxisDataType dataFormat, string name, params string[] values)

(표14 - 히포엔진 GetSeriesListOfGantt 메소드)

3. 히포엔진 사용 예

다음은 이론을 바탕으로 데이터 소스별 사용 예를 알아보겠습니다. 아래 샘플 소스만 이해하시면 히포엔진에 대해서는 무리 없이 사용할 수 있으리라 봅니다.

C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

// 히포 네임스페이스 논리적 참조
using Hippo;
```

```
using Hippo.ChartControl;

namespace HippoBookSamplesCshop
{
    public partial class Form1 : Form
    {
        HippoEngine en;
        SeriesList sList;

        public Form1()
        {
            InitializeComponent();

            this.InitChart();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            //DrawDataBase();
            //DrawAccess();
            //DrawExcel();
            //DrawXML();
            DrawArray();
        }

        /// <summary>
        /// 차트 초기 설정
        /// </summary>
        private void InitChart()
        {
            sList = new SeriesList();
            en = new HippoEngine();
        }

        /// <summary>
        /// 그리기
        /// </summary>
        private void Draw()
        {
            // 전체 초기화
            this.hHippoChart1.ResetAll();

            this.hHippoChart1.SeriesListDictionary.Add(sList);
```

```
            this.hHippoChart1.DrawChart();

            // 또는
            //this.hHippoChart1.DrawChart(sList);
        }

        /* ==========================================================
         *  히포 엔진 샘플
         *
         *  ========================================================== */
        #region
        /// <summary>
        /// (1)데이터베이스(Database)
        /// </summary>
        private void DrawDataBase()
        {
            en.ConnectionString = "server=localhost; uid=sa; pwd=1; database=pubs";
            en.Query = " SELECT * FROM sales";

            sList = en.GetSeriesList("stor_id", "qty");

            this.Draw();
        }

        /// <summary>
        /// (2)엑세스(Access)
        /// </summary>
        private void DrawAccess()
        {
            en.ConnectionString
=       @"Provider=Microsoft.ACE.OLEDB.12.0;Data       Source='C:₩Documents       and
Settings₩Administrator₩My Documents₩My Pictures₩book₩HippoDB.accdb';";

            en.Query = "select * from hippotable";

            sList = en.GetSeriesList(DataSourceType.Access, "name", "values");

            this.Draw();
        }

        /// <summary>
        /// (3)엑셀(Excel)
        /// </summary>
```

```
        private void DrawExcel()
        {
            en.ConnectionString
= @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source='F:₩MY Docu₩2007연령별성별인
구.xls';Extended Properties='Excel 12.0;HDR=YES'";

            en.Query = " SELECT * FROM [Sheet1$] ";

            sList = en.GetSeriesList(DataSourceType.Excel, "연령", "남자(명)", "여자(명)");
            sList.ChartType = ChartType.Bar;

            this.Draw();
        }

        /// <summary>
        /// (4)XML
        /// </summary>
        private void DrawXML()
        {
            en.XmlPath = @"C:₩Documents and Settings₩Administrator₩My
Documents₩My Pictures₩book₩source1.xml";

            sList = en.GetSeriesList(DataSourceType.Xml, "name", "values");

            this.Draw();
        }

        /// <summary>
        /// (5)배열(Array)
        /// </summary>
        private void DrawArray()
        {
            string[] names = new string[4] { "수영", "유리", "윤아", "티파니" };
            string[] values = new string[4] { "45", "78", "12", "99" };

            sList = en.GetSeriesList(names, values);

            this.Draw();
        }
        #endregion

        /// <summary>
        /// 차트 사이즈 변경 이벤트
```

```
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void hHippoChart1_ChartSizeChanged(object sender, EventArgs e)
        {
            this.hHippoChart1.DrawChart();
        }
    }
}
```

VB

```
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms

' 히포 네임스페이스 논리적 참조
Imports Hippo
Imports Hippo.ChartControl

Namespace HippoBookSamplesCshop
    Public Partial Class Form1
        Inherits Form
        Private en As HippoEngine
        Private sList As SeriesList

        Public Sub New()
            InitializeComponent()

            Me.InitChart()
        End Sub

        Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
            'DrawDataBase();
            'DrawAccess();
            'DrawExcel();
            'DrawXML();
            DrawArray()
```

```
        End Sub

        ''' <summary>
        ''' 차트 초기 설정
        ''' </summary>
        Private Sub InitChart()
            sList = New SeriesList()
            en = New HippoEngine()
        End Sub

        ''' <summary>
        ''' 그리기
        ''' </summary>
        Private Sub Draw()
            ' 전체 초기화
            Me.hHippoChart1.ResetAll()

            Me.hHippoChart1.SeriesListDictionary.Add(sList)

                ' 또는
                'this.hHippoChart1.DrawChart(sList);
            Me.hHippoChart1.DrawChart()
        End Sub

' ====================================================================
' * 히포 엔진 샘플
' *
' * ====================================================================

        #Region ""
        ''' <summary>
        ''' (1)데이터베이스(Database)
        ''' </summary>
        Private Sub DrawDataBase()
            en.ConnectionString = "server=localhost; uid=sa; pwd=1; database=pubs"
            en.Query = " SELECT * FROM sales"

            sList = en.GetSeriesList("stor_id", "qty")

            Me.Draw()
        End Sub

        ''' <summary>
```

```
        ''' (2)엑세스(Access)
        ''' </summary>
        Private Sub DrawAccess()
            en.ConnectionString
=       "Provider=Microsoft.ACE.OLEDB.12.0;Data Source='C:₩Documents and
Settings₩Administrator₩My Documents₩My Pictures₩book₩HippoDB.accdb';"

            en.Query = "select * from hippotable"

            sList = en.GetSeriesList(DataSourceType.Access, "name", "values")

            Me.Draw()
        End Sub

        ''' <summary>
        ''' (3)엑셀(Excel)
        ''' </summary>
        Private Sub DrawExcel()
            en.ConnectionString
=  "Provider=Microsoft.ACE.OLEDB.12.0;Data Source='F:₩MY Docu₩2007연령별성별인
구.xls';Extended Properties='Excel 12.0;HDR=YES'"

            en.Query = " SELECT * FROM [Sheet1$] "

            sList = en.GetSeriesList(DataSourceType.Excel, "연령", "남자 (명)", "여자
(명)")
            sList.ChartType = ChartType.Bar

            Me.Draw()
        End Sub

        ''' <summary>
        ''' (4)XML
        ''' </summary>
        Private Sub DrawXML()
            en.XmlPath = "C:₩Documents and Settings₩Administrator₩My
Documents₩My Pictures₩book₩source1.xml"

            sList = en.GetSeriesList(DataSourceType.Xml, "name", "values")

            Me.Draw()
        End Sub
```

```
        ''' <summary>
        ''' (5)배열(Array)
        ''' </summary>
        Private Sub DrawArray()
            Dim names As String() = New String(3) {"수영", "유리", "윤아", "티파니"}
            Dim values As String() = New String(3) {"45", "78", "12", "99"}

            sList = en.GetSeriesList(names, values)

            Me.Draw()
        End Sub
        #End Region

        ''' <summary>
        ''' 차트 사이즈 변경 이벤트
        ''' </summary>
        ''' <param name="sender"></param>
        ''' <param name="e"></param>
        Private Sub hHippoChart1_ChartSizeChanged(ByVal sender As Object, ByVal e
As EventArgs)
            Me.hHippoChart1.DrawChart()
        End Sub
    End Class
End Namespace
```

Hippo Namespace의 기초 클래스들을 이해하자.

앞서 설명한 바대로 히포차트를 구성하는 객체 중에는 몇 가지 기초 클래스들이 있습니다. 시리즈아이템을 '원자'에 비유했다면 이 클래스들은 '쿼크'로 비유한다면 맞을까요?

그럼 아래 표를 통해 자세히 알아보겠습니다. 히포차트를 이미 접해보신 분이라면 그냥 넘어가셔도 좋지만 처음 접하신 분들은 꼼꼼히 읽어보시기 바랍니다.

1. Label

Label 클래스는 히포차트 구성 객체들의 텍스트 영역을 관리하는 클래스입니다.
예를 들어 Titles 라는 객체가 있다면 차트 제목이므로 텍스트 영역이 있을 것입니다. 이럴 경우 Titles는 Label 객체를 가지게 됩니다.

```
this.hHippoChart1.Titles.Label.Text = "차트제목";
this.hHippoChart1.Logo.Label.ForeColor = Color.Red;
```

이런 식으로 기초 클래스들의 역할을 이해하고 중요 차트 구성 객체들을 접근한다면 일일이 매뉴얼을 찾아보지 않고도 충분히 개발을 진행할 수 있을 것입니다.
한 가지 더 예를 들어보면

"Y축의 제목을 넣고 싶다."

라고 생각을 했을 때 우선 축에 관련된 객체이고 Y축이므로

```
sList.AxisFactor.YAxis.
(시리즈리스트의 축 요소 중에 Y축 객체에서...)
```

까지 진행을 합니다. 그 다음에 YAxis가 가지고 있는 속성들 중에 타이틀에 관련된 객체를 찾아서

```
sList.AxisFactor.YAxis.TitleLabel.Text = "Y축입니다";
```

와 같이 설정할 수 있습니다. 이 부분은 단순한 내용이지만 좀 더 복잡한 코드를 통해 차트를 그릴 경우는 이러한 객체 지향적 사고방식은 더욱 필요하겠습니다.

Label 속성

글자를 구성하기 위해서는 폰트와 색상 그리고 내용이 있어야하므로 아래와 같이 구성이 되어 있습니다.

속성	설명
Font	폰트
ForeColor	글자색
Text	텍스트

(표15 - Label 속성)

2. Coumn

Column객체는 '하나의 기둥'을 의미합니다. 반드시 사각형 형태의 막대 기둥을 의미하는 것은 아니고 원기둥, 원뿔 기둥 등 포괄적인 의미입니다.

그럼 어떻게 접근해야 하나?

시리즈 개념을 정리하면서 이미 살펴보았지만 Column이라는 객체를 누가 가지고 있는지 먼저 알아야합니다.

앞서 알아본 바에 의하면 시리즈란 '하나의 차트를 그리기위한 최소한의 단위'라고 했습니다. 즉, 이 말은 시리즈는 차트 하나(일련의 데이터를 이용한)를 그리기위한 모든 요소를 가지고 있어야 한다는 말이 되겠습니다.

컬럼차트를 그리기 위해서 Column, 라인차트를 그리기위해서 Line, 스케터차트를 그리기위해서 Points 등 시리즈(Series)는 모든 기초 클래스들을 가지고 있습니다. 그리고 그 성질들을 상속받는 시리즈아이템 역시 이 기초 클래스들을 모두

가지고 있습니다. 시리즈리스트 레벨에 없는 이유는 시리즈리스트는 하나의 차트를 의미하는 객체가 아니라 동일한 차트군을 이루는 차트들의 무리 전체를 다루는 객체이기 때문입니다.

Column 속성

속성	설명
DesignType	기둥의 디자인 타입을 설정합니다. 본 속성은 차트 별로 적용되지 않는 부분이 있을 수 있습니다.(예, 원기둥일 경우 그라데이션이 적용되지 않습니다.) Default 기본형 Gradation 그라데이션 Dark 짙은 색상 Shadow 그림자효과
ImagesBarPath	기둥을 이미지로 디자인할 경우 설정합니다.(더이상 지원하지 않는 속성입니다.)
Width	기둥의 넓이를 가져옵니다.(읽기전용)
WidthType	기둥의 넓이의 종류를 설정합니다. Default 기본형 Dynamic 동적 변화형 Tight 꽉 찬 형태

(표16 - Column 속성)

3. Line

Line은 수학적 개념의 라인이 아니라 '두 점을 연결한 선'을 의미하는데 히포차트에서 라인은 실로 많은 부분에 등장합니다. 라인차트와 축라인, 그리드라인 등 가장 많이 사용되는 기초 클래스입니다.
(라인이 두 점을 연결한 선을 의미한다고 했으므로 한 개의 시리즈아이템을 그릴 경우 라인으로 그려지지 않습니다. 이는 Line 속성이 SeriesItem 레벨에 없다는 것을 통해 잘 알 수 있습니다.)

- Y축 라인 색상 바꾸기

```
sList.AxisFactor.YAxis.Line.LineColor = Color.Green;
```

● 라인차트를 그릴 때 점선으로 그리기

```
sr.Line.DashStyle = System.Drawing.Drawing2D.DashStyle.Dot;
```

● 좌표 영역 그리드라인 색상 바꾸기

```
sList.GraphArea.Grid.GridLine.LineColor = Color.Blue;
```

● AxisMarker에서의 라인

```
AxisMarker mk = new AxisMarker();
mk.Line.LineWidth = 2;
```

● 통계아이템에서의 라인

```
sList.AxisFactor.YAxis.AnalysisItems[AnalysisCategory.Max].Line.DashStyle =
System.Drawing.Drawing2D.DashStyle.Solid
```

Line 속성

속성	설명
AreaDesignType	라인 면적 디자인을 설정합니다. 면적의 디자인이므로 라인영역차트와 같은 특정 면적이 존재해야 됩니다. 이 속성은 일부 타입에 적용되지 않을 수 있습니다. Default 기본형 Gradation 그라데이션 Dark 짙은 색상 Shadow 그림자효과
DashCap	(※이 속성은 닷넷프레임워크가 지원하는 타입입니다.)
DashStyle	(※이 속성은 닷넷프레임워크가 지원하는 타입입니다.)
EndCap	(※이 속성은 닷넷프레임워크가 지원하는 타입입니다.)
LineColor	라인색상을 설정합니다.
LineWidth	라인굵기를 설정합니다.
StartCap	(※이 속성은 닷넷프레임워크가 지원하는 타입입니다.)

(표17 - Line 속성)

4. Points

Points 클래스는 하나의 포인트(점)을 의미합니다. 전체 차트 구성 객체에서 점은 스케터차트에 존재하며, 라인, 스플라인 차트에서도 나타납니다.

Points 속성

속성	설명
Color	포인트의 색상을 설정합니다.
PointImage	포인트를 사용자 지정 이미지로 사용할 때 그 경로를 지정합니다. ※이 기능은 주의를 요합니다.
PointType	포인트의 모양을 지정합니다. None 포인트 없음 Circle 원 (기본값) Rectangle 사각형 Triangle 삼각형 Star 별모양(현재 지원하지 않음) Image 이미지 FillCircle 원 FillRectangle 사각형 FillTriangle 삼각형
Width	포인트의 굵기를 설정합니다.

(표18 - Points 속성)

실전 히포차트 개발하기(1)

- 기초편 -

앞에서 히포차트의 기초 이론을 알아보았습니다. 이번 장부터는 본격적으로 차트를 개발하는 방법을 알아보겠습니다.

1. 히포차트 디자인

먼저, 히포차트 디자인에 대해 알아보겠습니다. 히포차트는 하늘색 파스텔톤의 인상적인 디자인을 가지고 있지만 사용자 프로그램과 조화를 이루지 못한다면 무용지물의 차트가 될 것입니다. 이번 장에서는 히포차트의 다양한 디자인 설정 요소들을 알아보고 사용자가 원하는 차트를 만들어 봅니다.

배경디자인

히포차트의 배경 디자인은 디자인 타입(DesignType)과 차트 디자이너(Chartdesigner)를 통해 다양하게 설정이 가능합니다. 디자인 타입은 차트의 전체 모양을 관리하는 열거형 타입의 속성이고, 차트 디자이너는 배경 디자인을 설정하는 속성입니다.

(1)디자인 타입(DesignType)

디자인타입	설명
Designable	히포차트의 고유의 배경 디자인입니다. 푸른 바탕에 흰색 종이가 올라가 있는 모양에 왼쪽 상단에 핀이 하나 꼽혀 있고 오른쪽 상단에는 조금 찢어진 모양으로 흰 색 부분에 그래프가 그려지고 푸른 바탕에 범례가 표시되는 형태입니다. 이 값은 디자인타입의 기본 값입니다. this.hHippoChart1.DesignType = ChartDesignType.Designable;
Default	기본 값인 Designable처럼 차트 주변이 라운드 처리가 되어 있고, 핀과 찢어진 모양이 빠진 형태입니다. this.hHippoChart1.DesignType = ChartDesignType.Default;
Classic	위 두 모양과 다르게 각 모서리의 라운드 처리 없이 각지게 표현되고 있습니다. this.hHippoChart1.DesignType = ChartDesignType.Classic;

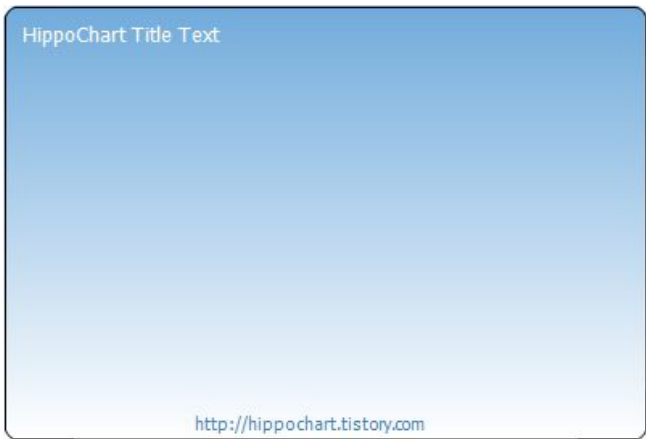
Flat	Default 형태에서 흰 색 도화지(?) 부분이 제거된 형태입니다. 이 타입은 배경색을 조절함으로써 다양한 효과를 줄 수 있어 유용하다 할 수 있습니다. this.hHippoChart1.DesignType = ChartDesignType.Flat;
Simple	베타 2.0 이상부터 추가된 형태이고 히포차트의 고유의 디자인을 조금 탈피한 주변 환경과 보다 잘 어울릴 수 있는 타입이라고 할 수 있습니다. 배경색(아래서 설명)을 조절함으로써 테두리 라인 색상이 변경 가능합니다. 단, 이 타입은 바탕이 흰색이므로 Titles 의 글자 색상(타이틀 텍스의 기본 값은 흰 색)을 변경해주어야 합니다. this.hHippoChart1.DesignType = ChartDesignType.Simple;
None	이 값은 설정사항이 아닙니다. this.hHippoChart1.DesignType = ChartDesignType.None;

(표19 - DesignType 속성)

Designable

Default

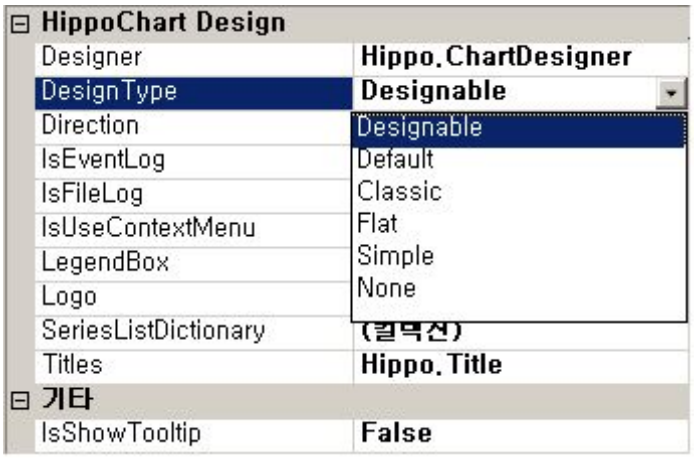
Classic



Flat

Simple

디자인타임에서는 아래와 같이 간단히 설정 가능합니다.



TIP.

Flat 타입과 Simple 타입은 색상을 흰색으로 줄 경우 배경을 완전히 하얀색으로 설정할 수 있다.

(2)차트 디자이너(ChartDesigner)

차트 디자이너는 차트 배경디자인을 담당하고 있는 클래스 속성으로 아래와 같은 속성들을 가지고 있습니다.

속성	설명
BackColor	차트의 배경색을 설정합니다. 디자인 타입이 Simple 타입

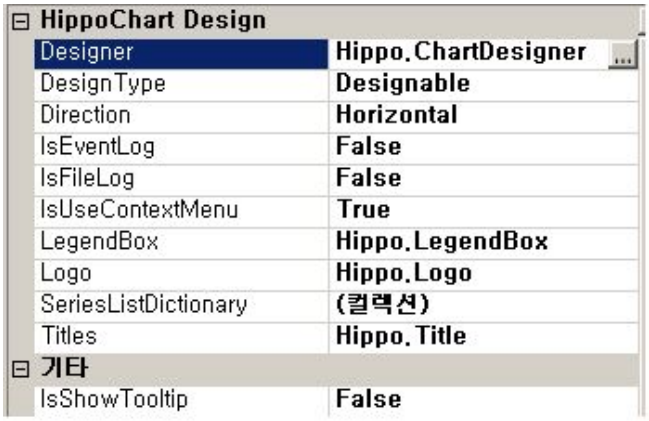
	인 경우는 외곽 라인색을 설정합니다.
BackLineColor	차트의 외곽 라인 색상을 설정합니다.
InnerBackColor	차트 내부 배경색을 설정합니다. 차트의 내부란
IsGradation	차트 배경색에 그라데이션 효과를 줄 것인지의 여부를 설정합니다. 기본 값은 true입니다.
IsShowBorder	차트 배경 라인을 그릴 것인지 여부를 설정합니다. 기본 값은 true입니다.

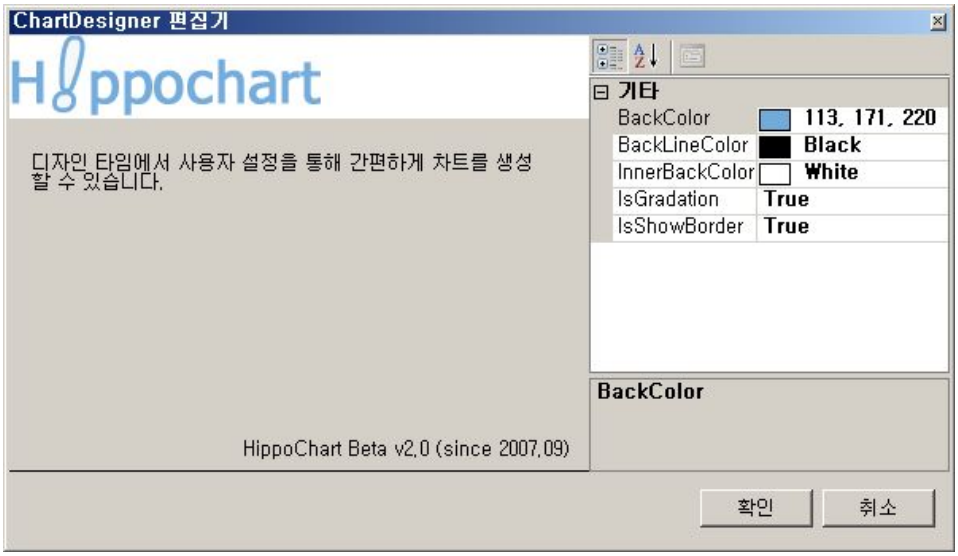
(표20 - 차트디자이너 속성)

차트 디자이너는 아래 코드와 같이 Designer라는 이름을 가지고 있으므로 혼돈이 없으시길 바랍니다.

```
this.hHippoChart1.Designer.BackColor = Color.Pink;
this.hHippoChart1.Designer.BackLineColor = Color.Red;
this.hHippoChart1.Designer.InnerBackColor = Color.White;
this.hHippoChart1.Designer.IsGradation = false;
this.hHippoChart1.Designer.IsShowBorder = true;
```

디자인 타임에서는 아래 그림처럼 접근할 수 있습니다.





TIP.

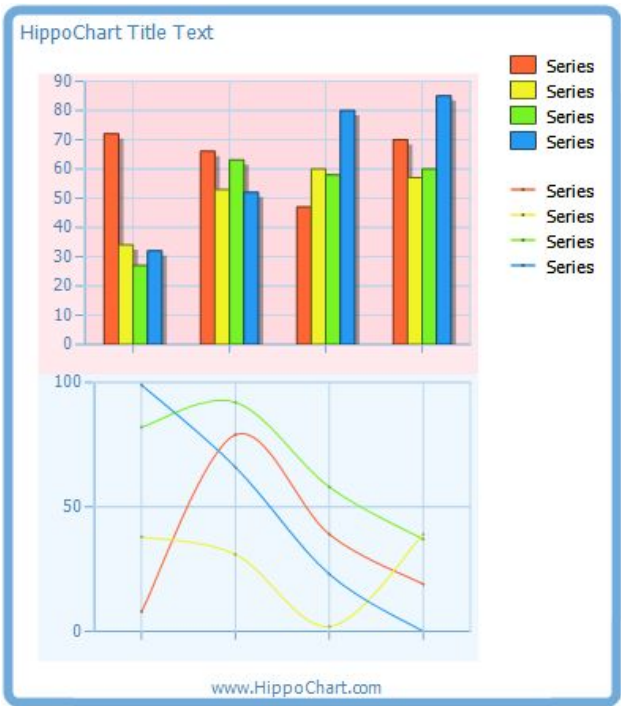
개발을 진행하다 보면 히포차트의 각종 초기 디자인 값을 알아야할 경우가 있는데 한 예로 초기 배경색은 아래와 같이 새 인스턴스를 생성함으로써 다시 가져올 수 있다.

```
ChartDesigner designer = new ChartDesigner();
Color defaultColor = designer.BackColor;
```

그래프 영역 디자인

그래프 영역은 앞에서 시리즈리스트의 속성을 통해 알아보았습니다. 아래 그림에서와 같이 타이틀 영역과 로고 영역, 바깥 범례 영역을 제외하고 기본적으로 띄워져 있는 패딩(Padding)을 제외한 안쪽 영역을 의미합니다.

시리즈리스트 전체를 둘러싸고 있으므로 GraphArea 속성은 시리즈리스트의 속성으로 존재하며 아래와 같이 시리즈리스트를 구분하거나 특별한 배경색이 필요한 경우 유용하겠습니다. (구성 속성은 앞장의 시리즈리스트 부분에서 알아보았습니다)



C#

```
SeriesList sList = new SeriesList();
SeriesList sList2 = new SeriesList();

sList.ChartType = ChartType.Column;
sList2.ChartType = ChartType.Spline;

sList.GraphArea.BackColor = Color.FromArgb(90, Color.Pink);
sList2.GraphArea.BackColor = Color.AliceBlue;

Random r = new Random();

for (int i = 0; i < 4; i++)
{
    Series sr = new Series();
    sr.Column.DesignType = AreaDesignType.Shadow;

    for (int x = 0; x < 4; x++)
    {
        SeriesItem item = new SeriesItem();
```

```
            item.YValue = r.Next(100);

            sr.items.Add(item);
        }
        sList.SeriesCollection.Add(sr);
    }

    for (int i = 0; i < 4; i++)
    {
        Series sr = new Series();

        for (int x = 0; x < 4; x++)
        {
            SeriesItem item = new SeriesItem();
            item.YValue = r.Next(100);

            sr.items.Add(item);
        }
        sList2.SeriesCollection.Add(sr);
    }

    this.hHippoChart1.Titles.Label.ForeColor = Color.SteelBlue;
    this.hHippoChart1.DesignType = ChartDesignType.Simple;
    this.hHippoChart1.Direction = GraphAreaLocation.Vertical;
    this.hHippoChart1.SeriesListDictionary.Add(sList);
    this.hHippoChart1.SeriesListDictionary.Add(sList2);
    this.hHippoChart1.DrawChart();
```

VB

```
Dim sList As New SeriesList()
Dim sList2 As New SeriesList()

sList.ChartType = ChartType.Column
sList2.ChartType = ChartType.Spline

sList.GraphArea.BackColor = Color.FromArgb(90, Color.Pink)
sList2.GraphArea.BackColor = Color.AliceBlue

Dim r As New Random()

For i As Integer = 0 To 3
    Dim sr As New Series()
```

```
    sr.Column.DesignType = AreaDesignType.Shadow

    For x As Integer = 0 To 3
Dim item As New SeriesItem()
item.YValue = r.[Next](100)

sr.items.Add(item)
    Next
    sList.SeriesCollection.Add(sr)
Next

For i As Integer = 0 To 3
    Dim sr As New Series()

    For x As Integer = 0 To 3
Dim item As New SeriesItem()
item.YValue = r.[Next](100)

sr.items.Add(item)
    Next
    sList2.SeriesCollection.Add(sr)
Next

Me.HHippoChart1.Titles.Label.ForeColor = Color.SteelBlue
Me.HHippoChart1.DesignType = ChartDesignType.Simple
Me.HHippoChart1.Direction = GraphAreaLocation.Vertical
Me.HHippoChart1.SeriesListDictionary.Add(sList)
Me.HHippoChart1.SeriesListDictionary.Add(sList2)
Me.HHippoChart1.DrawChart()
```

축 디자인

다음으로 축과 그리드 등 전반적인 좌표 영역의 디자인에 대해 알아보겠습니다. CPU 사용률 차트 등 기존의 특정 차트와 동일하게 설정해야하거나 특별한 색상으로 디자인해야하는 경우 축과 그리드 등의 세부 객체를 이용해 디자인이 가능합니다.

변경해야할 부분은

- X축 라인색
- X축 글자색
- X축 타이틀색
- Y축 라인색
- Y축 글자색
- Y축 타이틀색
- 그리드 라인색

으로 총 7가지 항목입니다. 글자가 들어가지 않는다면 각 축의 라인 색과 그리드 라인 색상이 핵심 사항이 되겠습니다.

```
sList.AxisFactor.XAxis.Line.LineColor = Color.YellowGreen;
sList.AxisFactor.YAxis.Line.LineColor = Color.YellowGreen;
sList.AxisFactor.XAxis.ForeColor = Color.YellowGreen;
sList.AxisFactor.YAxis.ForeColor = Color.YellowGreen;
sList.AxisFactor.XAxis.TitleLabel.ForeColor = Color.YellowGreen;
sList.AxisFactor.YAxis.TitleLabel.ForeColor = Color.YellowGreen;

sList.GraphArea.Grid.GridLine.LineColor = Color.FromArgb(100, Color.YellowGreen);
```

TIP.

사실 개발 시 위와 같이 장황한 코드가 자주 들어간다면 코드가 상당히 길어져 가독성과 모듈화를 저해할 수 있다. 그래서 필자는 아래와 같은 "테마" 메소드를 만들어 사용한다.
(아래 코드는 첨부된 샘플 코드 파일에 있으며 http://hippochart.tistory.com/166를 통해서도 참조 가능하다.)

C#

```
private void SetTheme(ref hHippoChart hippo, int SeriesListIndex, Color ThemeColor)
{
    hippo.Designer.BackColor = ThemeColor;

    hippo.SeriesListDictionary[SeriesListIndex].AxisFactor.XAxis.Line.LineColor = ThemeColor;
    hippo.SeriesListDictionary[SeriesListIndex].AxisFactor.YAxis.Line.LineColor = ThemeColor;
    hippo.SeriesListDictionary[SeriesListIndex].GraphArea.Grid.GridLine.LineColor =
Color.FromArgb(100, ThemeColor);

    int R = ThemeColor.R;
    int G = ThemeColor.G;
    int B = ThemeColor.B;

    R = (int)(R - R * (0.33));
    G = (int)(G - G * (0.33));
```

```
    B = (int)(B - B * (0.33));

    hippo.SeriesListDictionary[SeriesListIndex].AxisFactor.XAxis.ForeColor = Color.FromArgb(R, G, B);
    hippo.SeriesListDictionary[SeriesListIndex].AxisFactor.YAxis.ForeColor = Color.FromArgb(R, G, B);
    hippo.SeriesListDictionary[SeriesListIndex].AxisFactor.XAxis.TitleLabel.ForeColor =
Color.FromArgb(R, G, B);
    hippo.SeriesListDictionary[SeriesListIndex].AxisFactor.YAxis.TitleLabel.ForeColor =
Color.FromArgb(R, G, B);

    hippo.Titles.Label.ForeColor = Color.FromArgb(R, G, B);
    hippo.Logo.Label.ForeColor = Color.FromArgb(R, G, B);
}
```

VB

```
    Private Sub SetTheme(ByRef hippo As hHippoChart, ByVal SeriesListIndex As Integer, ByVal
ThemeColor As Color)
        hippo.Designer.BackColor = ThemeColor

        hippo.SeriesListDictionary(SeriesListIndex).AxisFactor.XAxis.Line.LineColor = ThemeColor
        hippo.SeriesListDictionary(SeriesListIndex).AxisFactor.YAxis.Line.LineColor = ThemeColor
        hippo.SeriesListDictionary(SeriesListIndex).GraphArea.Grid.GridLine.LineColor =
Color.FromArgb(100, ThemeColor)

        Dim R As Integer = ThemeColor.R
        Dim G As Integer = ThemeColor.G
        Dim B As Integer = ThemeColor.B

        R = CInt((R - R * (0.33)))
        G = CInt((G - G * (0.33)))
        B = CInt((B - B * (0.33)))

        hippo.SeriesListDictionary(SeriesListIndex).AxisFactor.XAxis.ForeColor =
Color.FromArgb(R, G, B)
        hippo.SeriesListDictionary(SeriesListIndex).AxisFactor.YAxis.ForeColor =
Color.FromArgb(R, G, B)
        hippo.SeriesListDictionary(SeriesListIndex).AxisFactor.XAxis.TitleLabel.ForeColor =
Color.FromArgb(R, G, B)
        hippo.SeriesListDictionary(SeriesListIndex).AxisFactor.YAxis.TitleLabel.ForeColor =
Color.FromArgb(R, G, B)

        hippo.Titles.Label.ForeColor = Color.FromArgb(R, G, B)
        hippo.Logo.Label.ForeColor = Color.FromArgb(R, G, B)
    End Sub
```

2. 차트 개발 시작하기

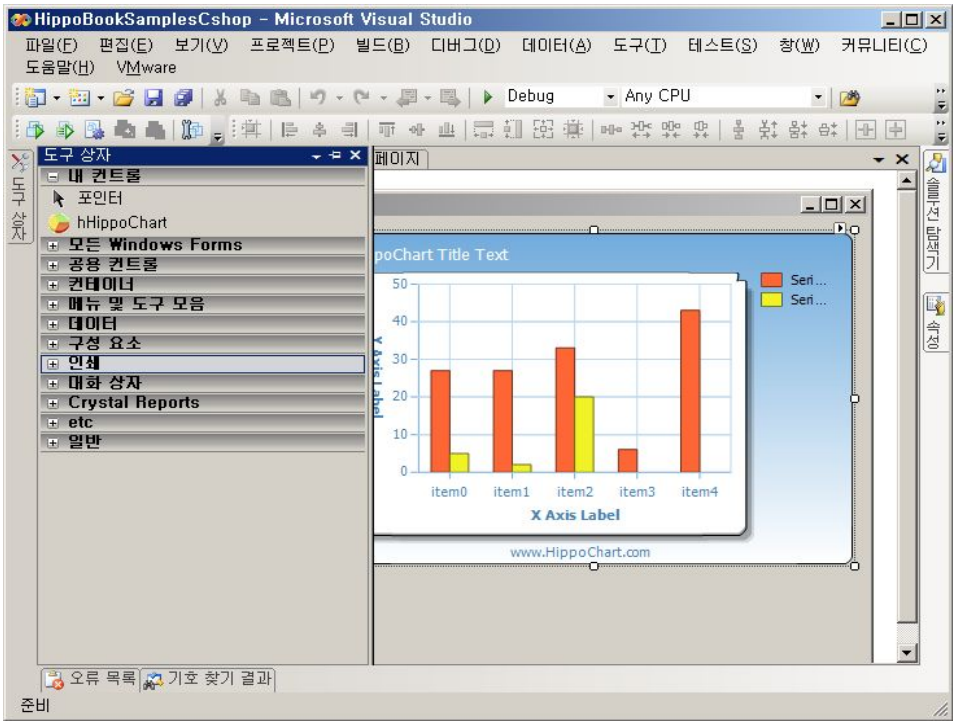
앞에서 히포차트의 기초 이론과 디자인하는 방법에 대해 알아보았습니다. 히포차트의 개발 방법은 대체로 직관적이고 평이하기 때문에 크게 어려운 내용은 없었을 거라 생각합니다. 이제 본격적으로 개발을 해 볼 텐데 처음 접하시는 분들은 머리띠 단단히 동여매시고 따라오시기 바랍니다.

(1) 윈도우즈 폼(Windows Form)

차트 레이아웃

설치가 완료되었으면 비주얼스튜디오를 열고 윈도우즈 응용프로그램 하나를 생성합니다.

다음 도구상자를 열어보면 “내 컨트롤” 이라는 탭에 hHippoChart 라는 이름으로 히포차트 컨트롤이 추가가 되어 있는 것을 볼 수 있습니다. 히포차트 컨트롤을 드래그 앤 드롭으로 폼에 하나 끌어다 놓고 히포차트를 선택한 상태에서 속성(Ctrl + W, P)창을 열어봅니다.

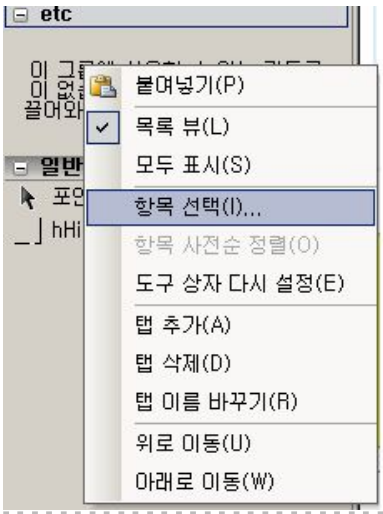


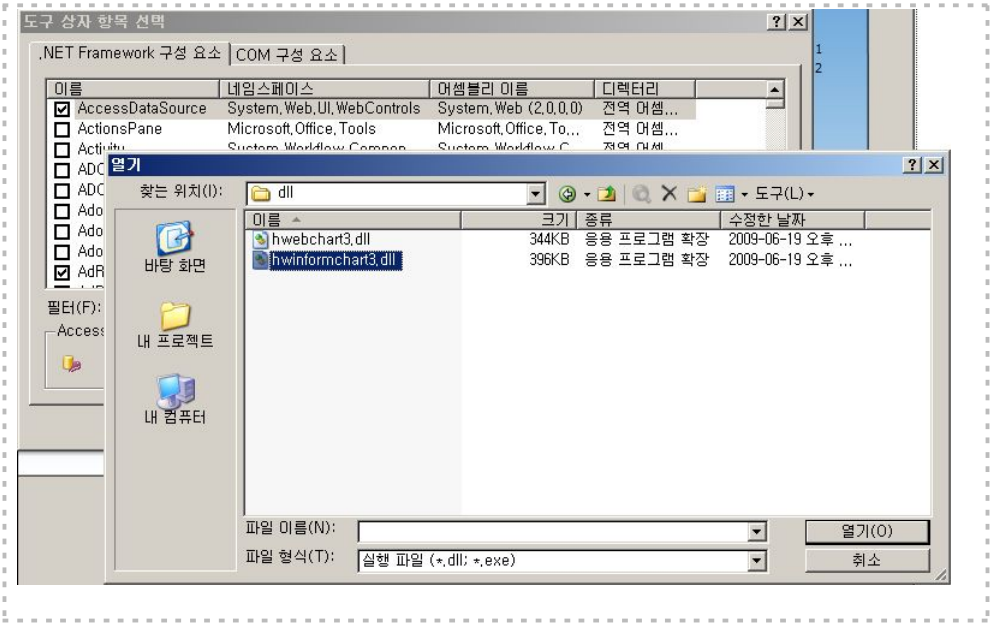
아래와 같은 몇 가지 속성들이 보이는데 하나씩 자세히 알아보겠습니다.

TIP. 도구상자에 컨트롤이 없어요!

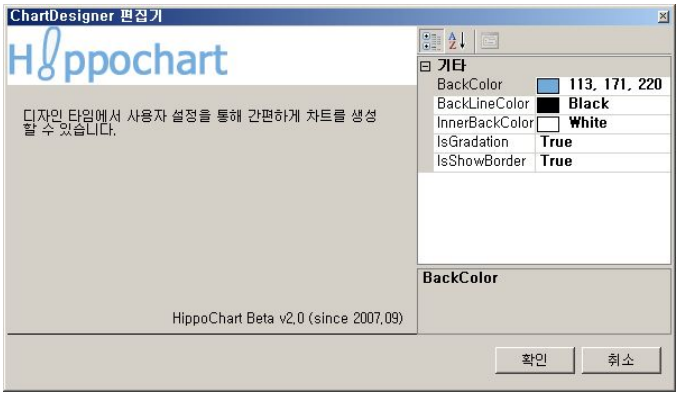
사용자 환경 혹은 비주얼스튜디오 언어 버전 등에 의해 히포차트 컨트롤의 자동 설치가 실패할 경우가 간혹 있는데 이럴 경우 수동으로 설치를 해야 한다. 수동으로 등록하더라도 자동설치된 것과 동일하게 사용 가능하므로 크게 문제될 것은 없다.

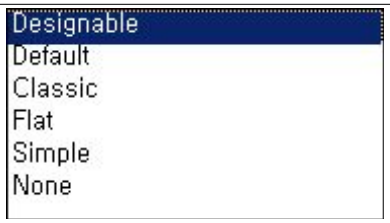
윈도우즈 폼 어플리케이션 하나를 실행하고 왼쪽 도구 상자에서 마우스 오른쪽 버튼을 클릭하면 항목 선택이라는 메뉴가 있다. 그러면 아래와 같이 도구상자 항목 선택이라는 창이 하나 뜨는데 우측 하단의 "찾아보기"를 통해 히포차트 dll를 직접 참조하면 된다.

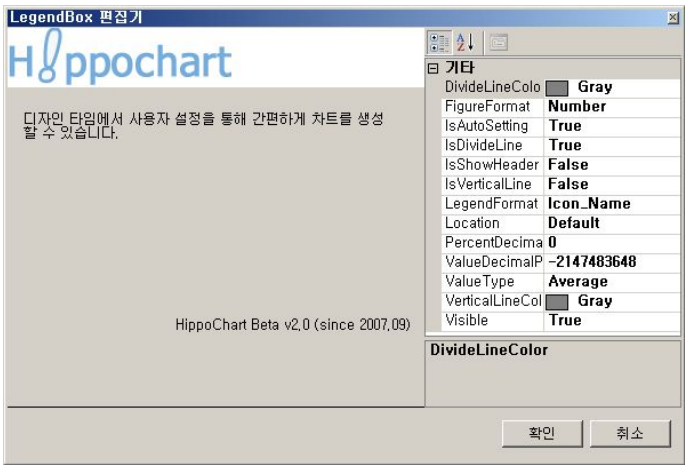
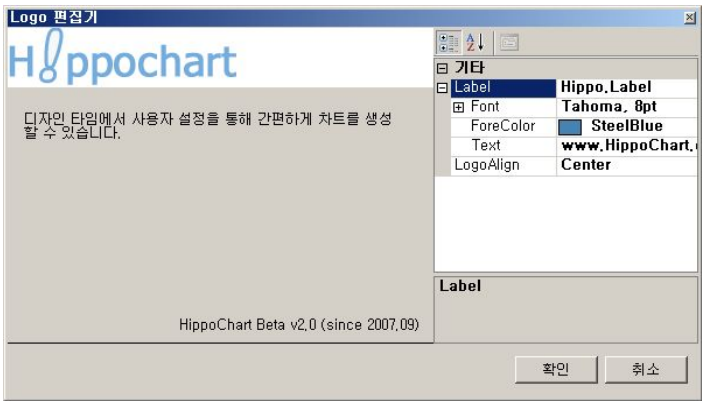


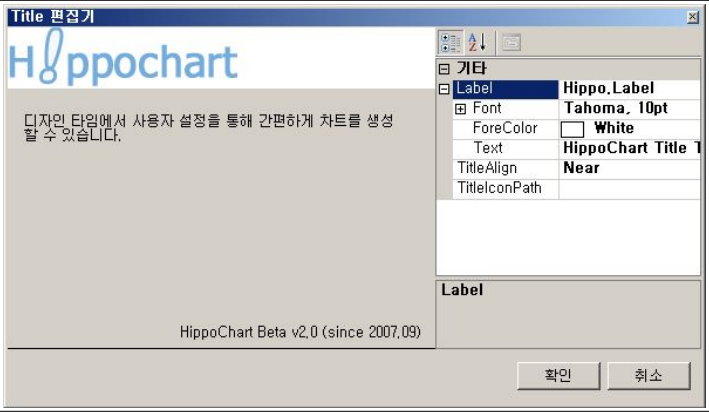


히포차트 윈도우즈 폼 디자이너 속성

속성	설명
Designer	차트 디자이너는 차트의 배경 디자인을 관리합니다. 
DesignType	히포차트의 외곽 배경 디자인을 관리하는 열거형 타입의 속성으로 아래와 같이 5가지 타입과 None으로 구성되어 있습니다.(None은 설정 사항이 아닙니다.)

	
Direction	이 속성은 차트를 다중 시리즈리스트로 구성할 경우 시리즈리스트가 추가되는 방향을 의미합니다.  기본 값은 수평방향입니다.
IsEventLog	개발 시 에러가 발생할 경우 로그를 기록하는 기능 중 하나로 이벤트뷰어에 내용을 등록합니다. 단, 개발 단계에서만 사용이 권장되며 배포 시에는 반드시 false를 해야 계속적인 이벤트 로그를 기록하는 일을 방지할 수 있습니다. (※이 속성은 베타 버전의 테스트를 위한 속성으로 정식 버전 출시 시에는 제거될 수 있습니다.)
IsFileLog	에러 로그 기록 중 하나로 파일시스템을 이용해서 로그를 기록합니다. IsEventLog 보다 사용이 권장되며 배포 초기에 true로 설정해 놓으면 초기 버그를 잡는데 유용합니다. 단, 한 번 그릴 때마다 로그파일을 작성하므로 실시간 차트일 경우 권장되지 않으며 안정화 단계에서는 false로 처리하는 것이 좋습니다. (※이 속성은 베타 버전의 테스트를 위한 속성으로 정식 버전 출시 시에는 제거될 수 있습니다.)
IsUseContextMenu	윈도우즈 폼 차트인 경우에만 제공하는 속성으로 차트 컨트롤의 컨텍스트 메뉴(마우스 오른쪽 버튼 기능)의 사용 유무를 설정합니다. ※주의 : 이 속성은 반드시 디자인타임에서 설정해야 올바르게 동작합니다.
LegendBox	범례(외부 범례)의 디자인타임 속성입니다. 범례에 대한 자세한 내용은 뒤에서(기능편)에서 자세히 알아보겠습니

	다. 
Logo	차트 하단에 표시되는 간단한 문구인 로고의 속성입니다. 역시 기능편에서 자세히 알아보겠습니다. 
SeriesListDictionary	이름에서도 알 수 있듯이 히포차트 컨트롤에 포함되어 있는 전체 시리즈리스트를 관리하는 컬렉션 속성입니다.
Titles	차트 상단에 표시되는 차트의 제목을 나타내는 속성입니다.

	
SeriesAreaRate	한 시리즈리스트 내에 2개 이상의 시리즈가 있을 경우 그 공간의 비율을 설정하는 기능입니다. 설정을 하지 않을 경우는 각 시리즈 별로 동일한 공간이 할당 됩니다.
IsShowTooltip	차트의 한 항목(한 시리즈아이템)에 마우스를 갖다 대었을 때 간단한 수치 정보를 보여주는 툴팁 기능을 사용할 것 인지 여부를 설정합니다. (※ 본 속성은 일부 차트에만 적용되며 실시간 차트, 많은 양의 데이터를 그릴 경우 등에는 권장하지 않습니다.)

(표21 - 윈도우즈 폼 속성)

TIP.

윈도우즈 폼의 디자이너 속성들 중에는 설정 후에도 바로 반영이 되지 않는 경우가 있는데 이는 클래스 속성으로 직렬화되어 리소스에 등록되어 있는 경우에 그렇다. 이럴 경우 아래 그림에서처럼 새로 고침을 해주면 바로 변경이 된다.

히포차트 윈도우즈 폼 이벤트

사이즈 변경 이벤트

```
hHippoChart1_ChartSizeChanged(object sender, EventArgs e)
```

차트의 크기가 변할 경우 발생합니다. 대표적인 사용 예는 아래와 같이 차트 크

기가 변할 경우 다시 그려주는 코드입니다.
이 이벤트는 기본 이벤트로서 차트에서 더블클릭 시 자동 생성됩니다.

C#

```
private void hHippoChart1_ChartSizeChanged(object sender, EventArgs e)
{
    this.hHippoChart1.DrawChart();
}
```

VB

```
Private Sub hHippoChart1_ChartSizeChanged(ByVal sender As Object, ByVal e As EventArgs)
   Me.hHippoChart1.DrawChart()
End Sub
```

컨텍스트 메뉴 생성 이벤트

```
hHippoChart1_ChartContextMenuCreated(object sender, ContextMenuEventArgs e)
```

차트 실행 후 마우스 오른쪽 버튼을 클릭하면 확인할 수 있는 컨텍스트 메뉴를 제어할 수 있는 이벤트입니다. 구현 방법은 아래와 같이 해당 아이템일 경우 구성을 취소하면 됩니다. 아래 예제는 차트 종류와 효과 메뉴를 제거하는 코드입니다.

C#

```
private void hHippoChart1_ChartContextMenuCreated(object sender, ContextMenuEventArgs
e)
{
    if (e.ContextMenuType == ContextMenuType.ChartType ||
        e.ContextMenuType == ContextMenuType.Effect)
    {
         e.Cancel = true
    }
}
```

VB

```
Private Sub hHippoChart1_ChartContextMenuCreated(ByVal sender As Object, ByVal e As ContextMenuEventArgs)
    If e.ContextMenuType = ContextMenuType.ChartType OrElse e.ContextMenuType = ContextMenuType.Effect Then
            e.Cancel = True
    End If
End Sub
```

차트 데이터 바인딩 이벤트

```
hHippoChart1_ChartDataBinging(object sender, EventArgs e)
```

※이 이벤트는 사용되지 않습니다.

차트디자이너 변경 이벤트

```
hHippoChart1_ChartDesignTypeChanged(object sender, EventArgs e)
```

차트의 디자인타입이 변경될 경우 발생합니다. 많이 사용되지는 않겠지만 사용자로 하여금 디자인타입을 변경하도록 코드를 작성한다거나 할 경우 유용합니다.

C#

```
private void hHippoChart1_ChartDesignTypeChanged(object sender, EventArgs e)
{
    if (this.hHippoChart1.DesignType == ChartDesignType.None)
    {
        MessageBox.Show("None은설정사항이아닙니다");
    }
}
```

VB

```
Private Sub hHippoChart1_ChartDesignTypeChanged(ByVal sender As Object, ByVal e As EventArgs)
  If Me.hHippoChart1.DesignType = ChartDesignType.None Then
     MessageBox.Show("None은 설정사항이 아닙니다.")
  End If
End Sub
```

차트 그리기

이제 간단하게 차트를 한 번 그려보겠습니다.

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    StartDraw();
}

private void StartDraw()
{
    this.hHippoChart1.DrawChart();
}
```

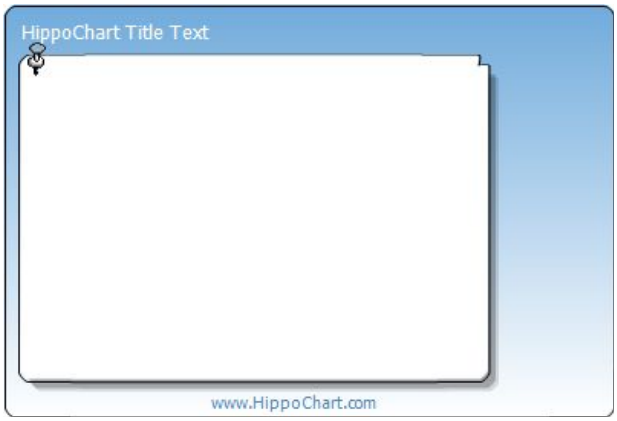
VB

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
    StartDraw()
End Sub

Private Sub StartDraw()
    Me.hHippoChart1.DrawChart()
End Sub
```

우선, 위와 같이 폼 로드 이벤트와 시작 메소드를 하나 만들고 폼 로드 시 호출하도록 코드를 작성합니다. 이벤트는 버튼 클릭 이벤트 등이 될 수도 있겠고 초기 형태는 위와 같을 것입니다.

이 상태에서 실행을 해보면 다음 페이지에 나오는 것과 같이 “빈 차트”가 그려집니다. 아무것도 추가된 것이 없고, 설정된 것이 없기 때문에 Designable의 차

트 타입으로 범례는 오른쪽, 타이틀 텍스트는 “HippoChart Title Text", 로그는 ”www.hippochart.com" 으로 모두 기본 값으로 설정이 되어 있는 것을 볼 수 있습니다.



그럼 하나씩 추가를 해볼까요?

가장 먼저 해야 할 일은 시리즈리스트를 생성하여 추가하는 일입니다. 처음 접하시는 분들은 따라서 한 번 작성해 보시는 것도 좋겠습니다.

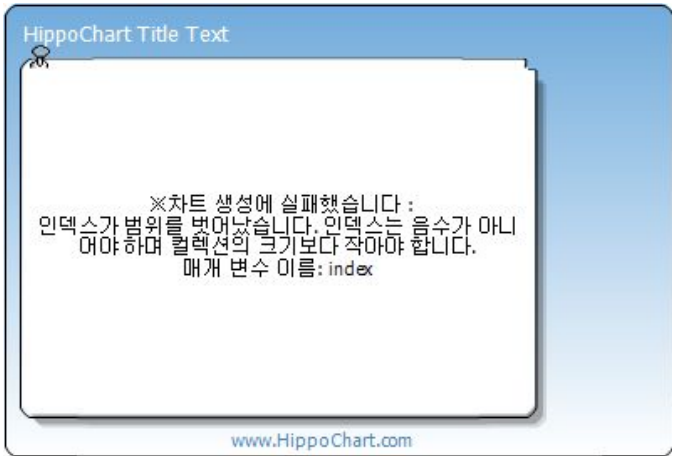
C#

```
private void StartDraw()
{
    SeriesList slist = new SeriesList();

    this.hHippoChart1.SeriesListDictionary.Add(slist);
    this.hHippoChart1.DrawChart();
}
```

VB

```
Private Sub StartDraw()
    Dim slist As New SeriesList()

    Me.hHippoChart1.SeriesListDictionary.Add(slist)
    Me.hHippoChart1.DrawChart()
End Sub
```

위와 같은 형태로 시리즈리스트 하나를 추가해보았습니다. 단 이 상태에서 실행을 해보면 아래와 같은 에러를 발생시키는데 이는 시리즈가 추가되지 않은 시리즈리스트를 추가했을 경우 발생하는 에러인데 첫 번째 시리즈를 찾는 도중에 인덱스 에러를 내는 것이지요.



다음에는 시리즈 한 개와 3개의 시리즈아이템을 추가해서 완성된 차트를 한 번 그려보겠습니다.

C#

```
private void StartDraw()
{
    SeriesList slist = new SeriesList();

    Series sr = new Series();

    SeriesItem item1 = new SeriesItem(55);
    SeriesItem item2 = new SeriesItem(22);
    SeriesItem item3 = new SeriesItem(77);

    sr.items.Add(item1);
    sr.items.Add(item2);
    sr.items.Add(item3);

    slist.SeriesCollection.Add(sr);

    this.hHippoChart1.SeriesListDictionary.Add(slist);
```

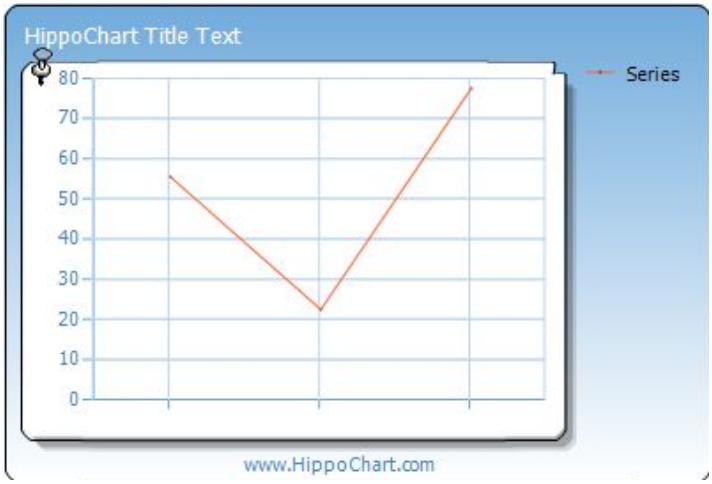
```
    this.hHippoChart1.DrawChart();
}
```

VB

```
Private Sub StartDraw()
    Dim slist As New SeriesList()

    Dim sr As New Series()

    Dim item1 As New SeriesItem(55)
    Dim item2 As New SeriesItem(22)
    Dim item3 As New SeriesItem(77)

    sr.items.Add(item1)
    sr.items.Add(item2)
    sr.items.Add(item3)

    slist.SeriesCollection.Add(sr)

    Me.hHippoChart1.SeriesListDictionary.Add(slist)
    Me.hHippoChart1.DrawChart()
End Sub
```

시리즈 하나를 생성해서 slist 라는 이름의 시리즈리스트에 이를 추가하고 시리즈아이템 3개를 생성해서 시리즈의 items 라는 컬렉션에 추가를 하고 있습니다. 시리즈의 개념에 대해서는 앞장에서 충분히 다루었으므로 크게 어려운 내용은 없을 것입니다.

아래가 그려진 결과 차트 이미지인데 3개의 데이터가 **기본 차트인 라인차트**로 그려지고 각 데이터에 맞게 자동으로 계산된 **y축 단계 수치**들이 보입니다.



그럼 필수적인 몇 가지 설정들을 더 추가해 볼까요?

우선 기본 값으로 되어 있는 타이틀과 로고를 수정해보고 범례에 Series라고 되어 있는 시리즈 이름을 수정해보겠습니다.

C#

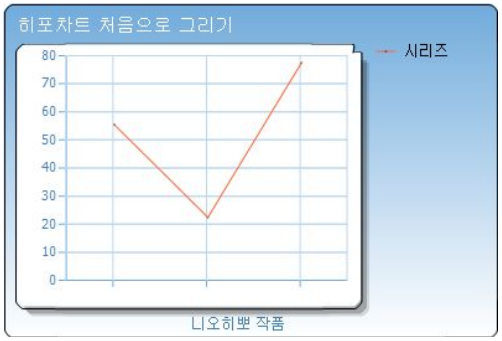
```
private void StartDraw()
{
    SeriesList slist = new SeriesList();

    Series sr = new Series();
    sr.Name = "시리즈";

    SeriesItem item1 = new SeriesItem(55);
    SeriesItem item2 = new SeriesItem(22);
    SeriesItem item3 = new SeriesItem(77);

    sr.items.Add(item1);
    sr.items.Add(item2);
    sr.items.Add(item3);

    slist.SeriesCollection.Add(sr);

    this.hHippoChart1.Titles.Label.Text = "히포차트처음으로그리기";
    this.hHippoChart1.Logo.Label.Text = "니오히뽀작품";
```

```
    this.hHippoChart1.SeriesListDictionary.Add(slist);
    this.hHippoChart1.DrawChart();
}
```

VB

```
Private Sub StartDraw()
    Dim slist As New SeriesList()

    Dim sr As New Series()
    sr.Name = "시리즈"

    Dim item1 As New SeriesItem(55)
    Dim item2 As New SeriesItem(22)
    Dim item3 As New SeriesItem(77)

    sr.items.Add(item1)
    sr.items.Add(item2)
    sr.items.Add(item3)

    slist.SeriesCollection.Add(sr)

    Me.hHippoChart1.Titles.Label.Text = "히포차트 처음으로 그리기"
    Me.hHippoChart1.Logo.Label.Text = "니오히뽀 작품"

    Me.hHippoChart1.SeriesListDictionary.Add(slist)
    Me.hHippoChart1.DrawChart()
End Sub
```

이상 간단하게 중요한 내용들을 추가해보았습니다. 아래 나온 결과 이미지가 가장 기본적인 형태의 차트가 되겠네요. 위 코드는 간단한 내용이지만 기본이 되는 형태이므로 히포 초보 분들은 반드시 숙지하시기 바랍니다.



코드에서의 이미지 저장

다음 단락에서 설명을 하겠지만 차트 작성이 완료된 후에는 런타임 시에 차트 이미지를 얻을 수 있는데 이는 코드에서 사용자가 원하는 시점에 이미지를 저장할 수도 있습니다. 아래와 같은 코드를 통해 간단히 적용이 가능하며 이 코드는 반드시 DrawChart() 메소드 다음에 와야 합니다.

```
this.hHippoChart1.SaveImage(@"C:₩내차트.jpg", System.Drawing.Imaging.ImageFormat.Jpeg);
```

(※이 메소드를 이용하여 이미지를 저장할 경우 여러 가지 형식의 이미지 형식이 지원되지만 히포차트는 png형식을 권장합니다.)

컨텍스트 메뉴

앞에서 잠깐 설명했듯이 히포차트 윈도우즈 폼 버전에서는 다양한 마우스 오른쪽 버튼 메뉴를 지원합니다. 총 큰 분류로 11개의 항목으로 구성되어 있고 그 하위에 상세한 설정 사항들이 나와 있습니다.

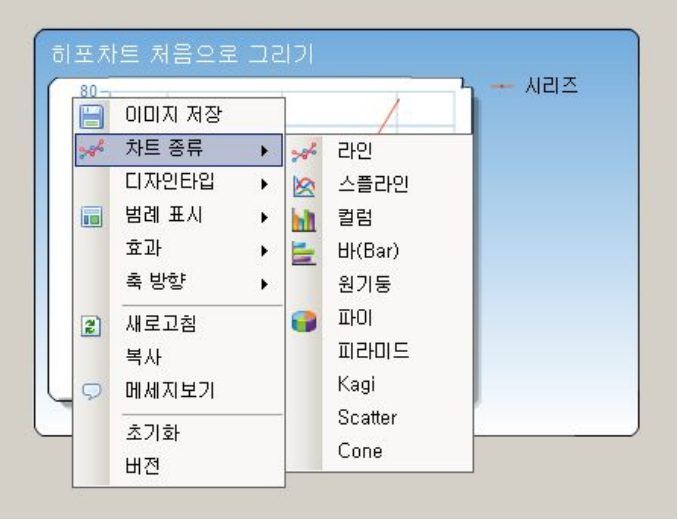
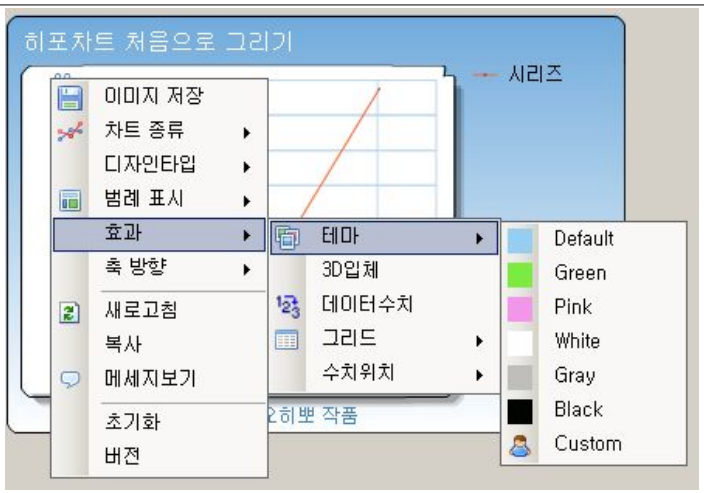
항목	설명
이미지 저장	런타임 시에 작성한 차트 이미지를 파일시스템으로 저장할 수 있습니다. 이 경우는 PNG 파일 형식으로만 저장이 가능합니다.
차트 종류	

	차트 종류 항목에 보면 런타임 시에 변경 가능한 10가지 차트를 확인할 수 있습니다. 히포차트가 지원하는 모든 차트가 보이지 않는 이유는 데이터의 종류에 따라 그릴 수 없는 차트가 있기 때문입니다. 이 항목은 참고를 위한 메뉴입니다.
디자인타입	런타임 시에 디자인 타입을 변경할 수 있습니다.
범례표시	런타임 시에 범례의 표시 유무, 위치, 형태를 변경할 수 있습니다.
효과	 1. 테마: 차트의 전반적인 색상 디자인을 변경 2. 3D입체: 추가된 모든 차트를 3d입체 또는 2d 로 변경 3. 데이터 수치: 그래프의 수치를 표시할 것인 여부 결정 4. 그리드: 그리드의 방향을 4가지로 설정 5. 수치위치: 각 시리즈 별 표시되는 수치의 상대적 위치를 조정
축 방향	좌표 성격의 차트일 경우 4가지 방향으로 차트를 회전시킬 수 있습니다. (단, 일부 차트는 변경이 불가능할 수 있습니다.)
새로고침	현재까지 누적된 설정 사항으로 차트를 다시 한 번 그립니다.
복사	차트 이미지를 클립보드에 복사합니다. (그림판, 포토샵, 각종 문서 등에 바로 첨부할 수 있습니다.)
메시지 보기	차트가 완성된 후 리턴하는 메시지를 확인할 수 있습니다.
초기화	차트의 모든 설정 사항을 초기화 하고 빈 차트를 그립니다.
버전	현재 버전을 확인합니다.

(표22 - 컨텍스트 메뉴)

실시간(Realtime) 차트

히포차트는 기본적으로 가볍고 리소스 관리를 철저히 하기 때문에 계속적인 drawing도 무리 없이 처리를 할 수 있으며 DrawRealTimeChart() 메소드를 지원함으로써 손쉽게 실시간 차트를 그릴 수 있습니다.

실시간 차트는 사용자가 DrawRealTimeChart() 메소드를 이용하지 않고 직접 구현할 수도 있으나 여기서는 해당 메소드를 이용하는 방법을 알아보겠습니다.

DrawRealTimeChart() 메소드는 아래와 같이 4개의 오버로드로 구성되어 있습니다. 파라미터는 데이터와 maxDataCount로 되어 있는데 maxDataCount는 한 차트의 X축 범위 안에 표시될 데이터의 개수를 의미합니다. 데이터는 수치만 넘길 수도 있고 시리즈아이템 객체를 구성하여 넘길 수 있으며 params 키워드를 통해 배열을 넘김으로써 멀티 시리즈 실시간 차트의 구성도 가능합니다.

1. 데이터, 최대스케일 값

```
public void DrawRealTimeChart(double data, int maxDataCount)
```

시리즈아이템의 구성없이 그냥 수치 데이터만으로 그릴 경우 호출합니다. 한 개의 시리즈로 그릴 경우 사용하는 메소드이고 maxDataCount를 통해 가로축 개수를 지정할 수 있습니다.

2. 시리즈아이템, 최대스케일 값

```
public void DrawRealTimeChart(SeriesItem data, int maxDataCount)
```

시리즈아이템을 구성하여 파라미터로 넘김으로써 포인트 굵기, 모양, 풍선도움말 등을 구성할 수 있습니다.

3. 최대스케일 값, 데이터 배열

```
public void DrawRealTimeChart(int maxDataCount, params double[] data)
```

멀티시리즈 실시간 차트를 구성할 수 있는 메소드로 데이터 배열을 넘길때 시

리즈의 순서대로 넘기면 되겠습니다.

4. 최대스케일 값, 시리즈아이템 배열

```
public void DrawRealTimeChart(int maxDataCount, params SeriesItem[] data)
```

멀티시리즈 실시간 차트를 구성할 수 있는 메소드이고 params 키워드의 배열 순서는 추가된 시리즈의 순서입니다.

샘플 코드를 통해 보다 자세히 알아보겠습니다.

아래 샘플은 3개의 시리즈를 실시간으로 그리는 '**멀티 시리즈 실시간 차트**' 예제입니다. 우선 폼 로드에서 기본적인 설정들을 해주어야하는데 아래 코드에서는 시리즈리스트를 생성하여 히포차트 객체에 추가를 하고 있고 3개의 시리즈를 생성하여 시리즈리스트에 추가를 하고 있습니다.

그리고 중요한 사항은 RealTimeList라는 컬렉션 속성입니다. 이 객체는 실시간 차트를 그리기 위해 유지해야하는 여러 값들을 내부적으로 유지하고 있는데 이는 시리즈 개수만큼 필요합니다. 아래 코드를 보면 시리즈 개수만큼 3개가 추가되어 있는 것을 알 수 있습니다.

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    SeriesList sList = new SeriesList();
    sList.SeriesCollection.Add(new Series());
    sList.SeriesCollection.Add(new Series());
    sList.SeriesCollection.Add(new Series());

    this.hHippoChart1.RealTimeList.Add(new Hippo.ChartControl.HippoRealTimeAttribute());
    this.hHippoChart1.RealTimeList.Add(new Hippo.ChartControl.HippoRealTimeAttribute());
    this.hHippoChart1.RealTimeList.Add(new Hippo.ChartControl.HippoRealTimeAttribute());

    this.hHippoChart1.DesignType = ChartDesignType.Simple;
    this.hHippoChart1.SeriesListDictionary.Add(sList);

    timer1.Interval = 10;
    timer1.Start();
```

```
}
```

VB

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
    Dim sList As New SeriesList()
    sList.SeriesCollection.Add(New Series())
    sList.SeriesCollection.Add(New Series())
    sList.SeriesCollection.Add(New Series())

    Me.hHippoChart1.RealTimeList.Add(New Hippo.ChartControl.HippoRealTimeAttribute())
    Me.hHippoChart1.RealTimeList.Add(New Hippo.ChartControl.HippoRealTimeAttribute())
    Me.hHippoChart1.RealTimeList.Add(New Hippo.ChartControl.HippoRealTimeAttribute())

    Me.hHippoChart1.DesignType = ChartDesignType.Simple
    Me.hHippoChart1.SeriesListDictionary.Add(sList)

    timer1.Interval = 10
    timer1.Start()
End Sub
```

위 예제는 Timer 객체를 이용해서 실시간 차트를 그리고 있습니다. 다음은 타이머의 tick 이벤트를 구현해야겠네요.

위 초기 설정에서 3개의 시리즈를 추가했으므로 반복 호출되는 오퍼레이션에서도 반드시 3개의 데이터를 배열로 넘겨야합니다. 반복 호출 메소드를 구성할 경우 **주의할 점**은 필요 없는 코드들을 모두 제거하고 가볍게 구성해야한다는 점입니다. 만약 아래 tick 이벤트에서 시리즈리스트를 생성한다거나 초기설정에서 해야 하는 타이틀, 로고, 범례 등의 설정을 계속적으로 한다면 실시간으로 차트를 그리는데 성능에 지장을 줄 수 있습니다.

C#

```
private void timer1_Tick(object sender, EventArgs e)
{
    Random rr1 = new Random();
    Random rr2 = new Random();

    SeriesItem item = new SeriesItem(rr1.Next(999));

    SeriesItem item2 = new SeriesItem(rr1.Next(555));
```

```
    SeriesItem item3 = new SeriesItem(500);
    item3.Name = DateTime.Now.ToString("mm:ss");

    this.hHippoChart1.DrawRealTimeChart(450, item, item2, item3);
}
```

VB

```
Private Sub timer1_Tick(ByVal sender As Object, ByVal e As EventArgs)
    Dim rr1 As New Random()
    Dim rr2 As New Random()

    Dim item As New SeriesItem(rr1.[Next](999))

    Dim item2 As New SeriesItem(rr1.[Next](555))

    Dim item3 As New SeriesItem(500)
    item3.Name = DateTime.Now.ToString("mm:ss")

    Me.hHippoChart1.DrawRealTimeChart(450, item, item2, item3)
End Sub
```

TIP.

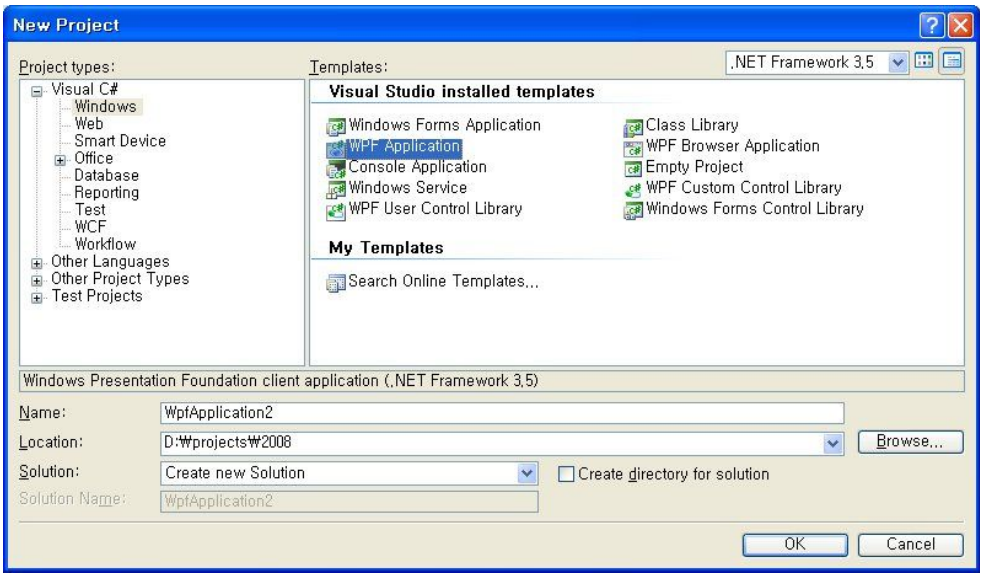
위 코드를 보면 시리즈아이템의 이름은 가장 마지막 item3에만 설정이 되어있다. 이는 같은 시리즈리스트일 경우 가장 데이터 수가 많은 아이템, 가장 마지막으로 추가되는 아이템 순으로 X축 라벨 리스트를 구성하기 때문이다.

WPF에서 히포차트 사용하기

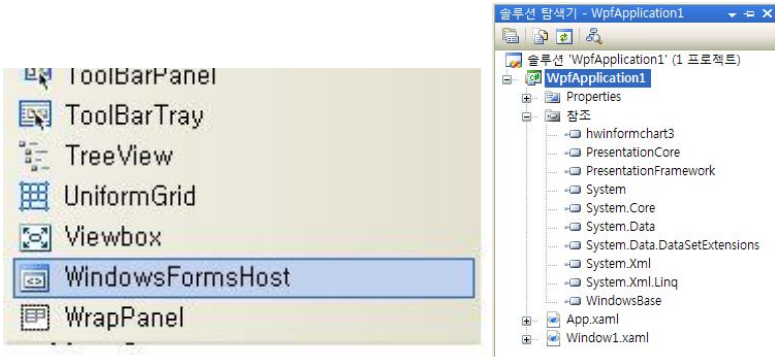
히포차트는 WPF 전용 컨트롤을 지원하지 않지만 윈도우즈 폼 차트 컨트롤을 WPF 환경에 호스팅함으로써 윈도우즈 폼 환경과 동일하게 모든 기능을 사용할 수 있습니다. 다만, 디자인 타임 지원을 사용할 수 없고 런타임 시에 히포차트 객체를 생성하여 추가하는 방법입니다.

시작하기

우선 WPF 어플리케이션을 하나 생성합니다.



다음으로 히포차트를 WPF에 호스트 해줄 컨트롤인 windowsFormsHost 컨트롤을 하나 끌어다 놓습니다.



다음 위 오른쪽 그림처럼 히포차트 윈도우즈 폼 컨트롤을 참조 추가합니다. 자 이제 모든 준비가 끝이 났고 코드를 통해 히포차트를 한 번 그려보겠습니다.

C#

```
hHippoChart hh = new hHippoChart();
```

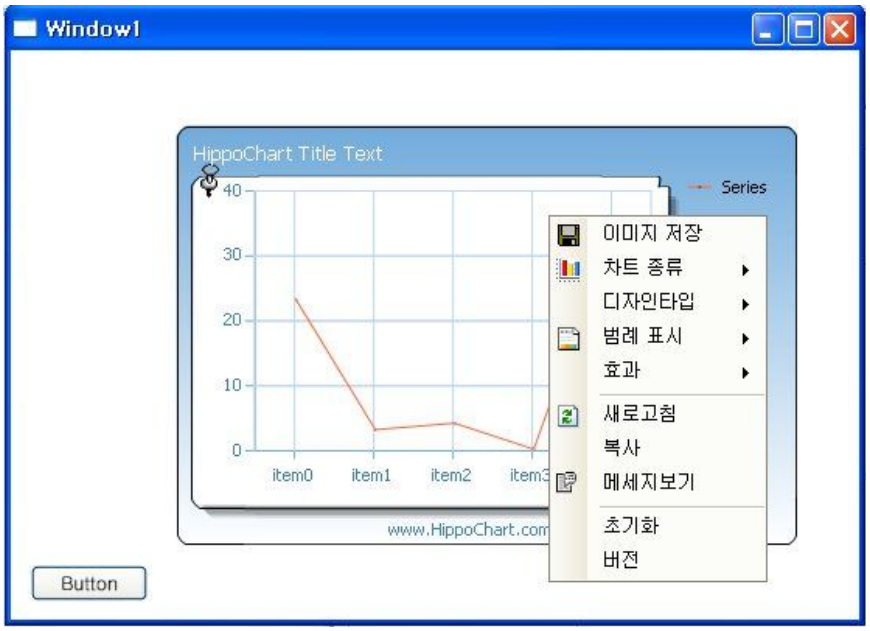
```
hh.Width2 = (float)windowsFormsHost1.Width;
hh.Height2 = (float)windowsFormsHost1.Height;

SeriesList sList = new SeriesList();
sList.SeriesCollection.Add(new Series());

Random R = new Random();

for (int i = 0; i < 5; i++)
{
    SeriesItem item = new SeriesItem();
    item.Name = "item" +  i.ToString();
    item.YValue = R.Next(50);

    sList.SeriesCollection[0].items.Add(item);
}

hh.SeriesListDictionary.Add(sList);
hh.DrawChart();

this.windowsFormsHost1.Child = hh;
```

VB

```
Dim hh As New hHippoChart()
hh.Width2 = CSng(windowsFormsHost1.Width)
hh.Height2 = CSng(windowsFormsHost1.Height)
Dim sList As New SeriesList()
sList.SeriesCollection.Add(New Series())

Dim R As New Random()

For i As Integer = 0 To 4
    Dim item As New SeriesItem()
    item.Name = "item" & i.ToString()
    item.YValue = R.[Next](50)

    sList.SeriesCollection(0).items.Add(item)
Next
```

```
hh.SeriesListDictionary.Add(sList)
hh.DrawChart()

Me.windowsFormsHost1.Child = hh
```

코드에서 보면 디자인 타임에 추가를 할 수 없기 때문에 hHippoChart 라는 윈도우즈 폼 차트 컨트롤 객체를 직접 생성을 하여 차트의 크기를 지정하여 사용하고 있습니다. 나머지 과정은 모두 동일 하며 중요한 부분은 코드의 하단에 windowsFormsHost에 히포차트 객체를 추가해주는 부분이 되겠습니다. 아래는 실행해본 결과입니다.



TIP.

WindowsFormsHost 컨트롤 (MSDN 발췌)

.NET Framework 클래스 라이브러리
WindowsFormsHost 클래스

WPF 페이지에서 Windows Forms 컨트롤을 호스팅할 수 있는 요소입니다.

설명
Windows Forms 컨트롤을 WPF 요소 또는 페이지 내에 배치하려면 WindowsFormsHost 요소를 사용합니다. Windows Forms 컨트롤 또는 폼에서 WPF 요소를 호스팅하려면 ElementHost 컨트롤을 사용합니다.

참고:
WindowsFormsIntegration.dll은 WPF(Windows Presentation Foundation) 어셈블리와 함께 설치됩니다. 어셈블리의 기본 위치는
%programfiles%₩Reference Assemblies₩Microsoft₩Framework₩v3.0₩WindowsFormsIntegration.dll입니다.

WPF 요소에서 Windows Forms 컨트롤을 호스팅하려면 Child 속성에 Windows Forms 컨트롤을 할당해야 합니다.

WindowsFormsHost 요소와 호스팅되는 해당 Windows Forms 컨트롤 사이의 사용자 지정 매핑을 할당하려면 PropertyMap 속성을 사용합니다. 자세한 내용은 Windows Forms 및 WPF 속성 매핑을 참조하십시오.

예제
다음 코드 예제에서는 WindowsFormsHost 요소를 사용하여 System.Windows.Forms..::.MaskedTextBox 컨트롤을 호스팅하는 방법을 보여 줍니다. 자세한 내용은 연습: Windows Presentation Foundation에서 XAML을 사용한 Windows Forms 컨트롤 호스팅을 참조하십시오.

Visual Basic코드

```
<Window x:Class="Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:wf="clr-namespace:System.Windows.Forms;assembly=System.Windows.Forms"
    Title="HostingWfInWpf">
  <Grid>
    <WindowsFormsHost>
      <wf:MaskedTextBox x:Name="mtbDate" Mask="00/00/0000"/>
    </WindowsFormsHost>
  </Grid>
</Window>
```

C#코드

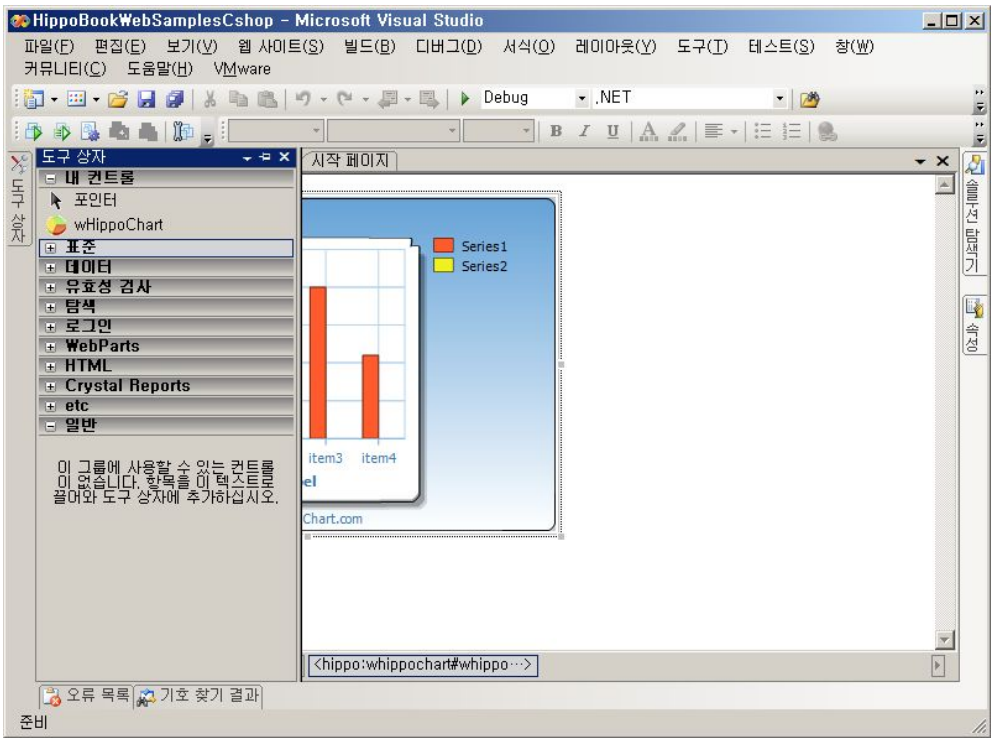
```
<Window x:Class="HostingWfInWpf.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:wf="clr-namespace:System.Windows.Forms;assembly=System.Windows.Forms"
```

```
    Title="HostingWfInWpf">
    <Grid>
        <WindowsFormsHost>
            <wf:MaskedTextBox x:Name="mtbDate" Mask="00/00/0000"/>
        </WindowsFormsHost>
    </Grid>
</Window>
```

(2) 웹 폼(Web Form)

차트 레이아웃

설치가 완료되었으면 비주얼스튜디오를 열고 웹 응용프로그램 혹은 웹사이트 하나를 생성합니다.



다음 도구상자를 열어보면 “내 컨트롤” 이라는 탭에 wHippoChart 라는 이름으로 히포차트 컨트롤이 추가가 되어 있는 것을 볼 수 있습니다. 히포차트 컨트롤을 드래그 앤 드롭으로 폼에 하나 끌어다 놓고 히포차트(이하 히포)를 선택한 상

태에서 속성(Ctrl + W, P)창을 열어봅니다.

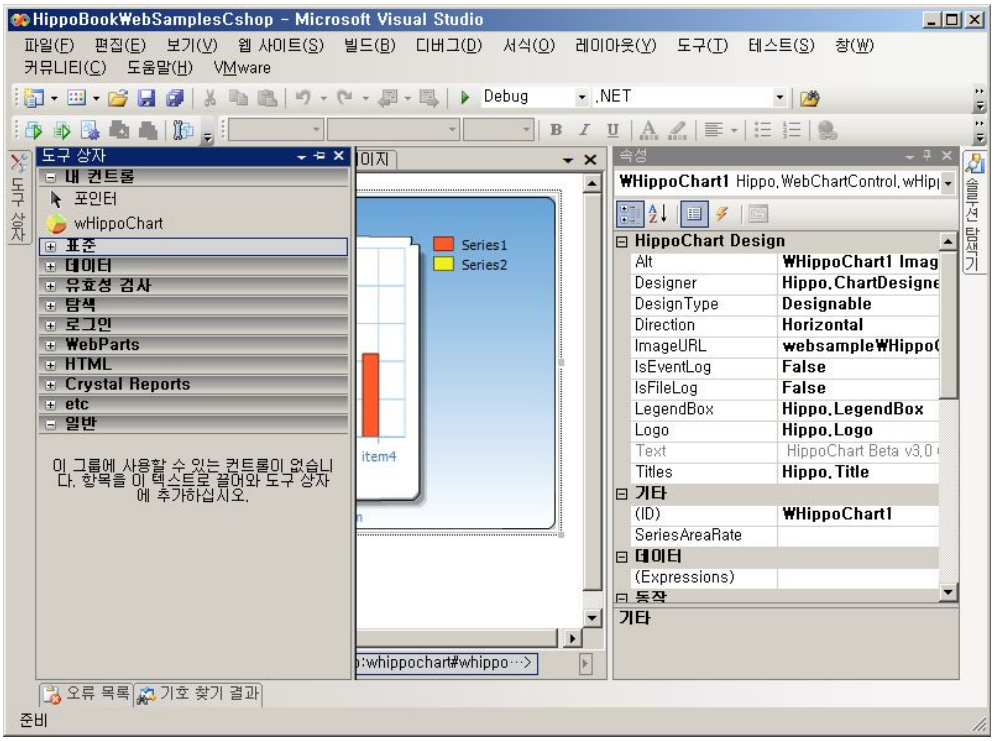
차트를 페이지에 끌어다 놓고 '디자인' 모드를 통해 HTML 부분을 살펴보면 아래와 같이 상단에 차트 컨트롤에 대한 네임스페이스(Hippo.WebChartControl)가 Hippo 라는 이름으로 등록이 되어 있고 wHippoChart 컨트롤 객체가 정의되어 있습니다.

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs" Inherits="_Default" %>

<%@ Register Assembly="hwebchart3" Namespace="Hippo.WebChartControl" TagPrefix="Hippo" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>제목없음</title>
</head>
<body>
    <form id="form1" runat="server">
<Hippo:wHippoChart ID="WHippoChart1" runat="server"></Hippo:wHippoChart>
</form>
</body>
</html>
```

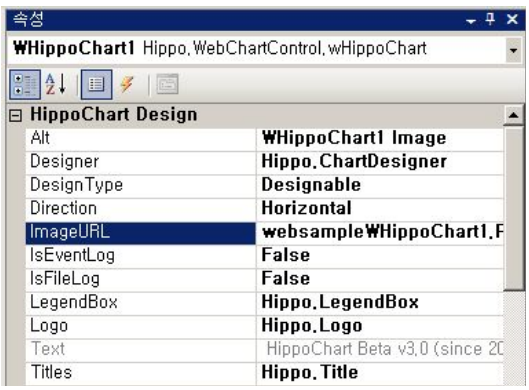
아래 코드는 차트 크기를 변경하거나 몇 가지 속성 변경을 할 경우 내부 속성들이 보이게 되는데 아래 html 코드를 통해서도 간단한 속성 설정을 할 수 있습니다. (단, 시리즈리스트 설정은 html 코드로 하기에 너무나 복잡하므로 제한해 놓았습니다.)

```
<Hippo:wHippoChart ID="WHippoChart1" runat="server" Height="288px" Width="421px">
            <LegendBox DivideLineColor="Gray" FigureFormat="Number" Height="213.239578"
IsAutoSetting="True" IsDivideLine="True" IsShowHeader="False"
IsVerticalLine="False" LegendFormat="Icon_Name"
                Location="Default" PercentDecimalPoint="0"
ValueDecimalPoint="-2147483648" ValueType="Average"
                VerticalLineColor="Gray" Visible="True" Width="95.58289" />
            <Titles TitleAlign="Near">
                <Label Font="Tahoma, 10pt" ForeColor="White" Text="HippoChart Title Text" />
            </Titles>
            <Logo LogoAlign="Center">
                <Label Font="Tahoma, 8pt" ForeColor="SteelBlue" Text="www.HippoChart.com" />
```

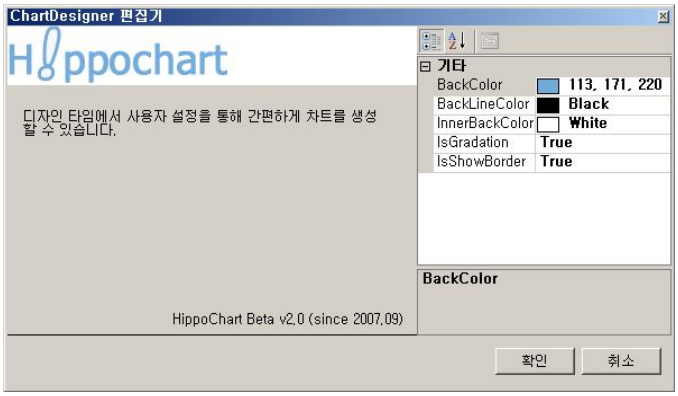
```
        </Logo>
        <Designer BackColor="113, 171, 220" BackLineColor="Black" InnerBackColor="White"
            IsGradation="True" IsShowBorder="True" />
</Hippo:wHippoChart>
```

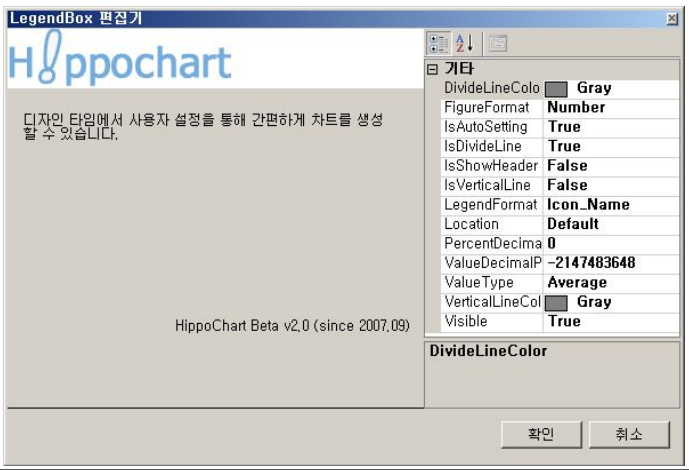


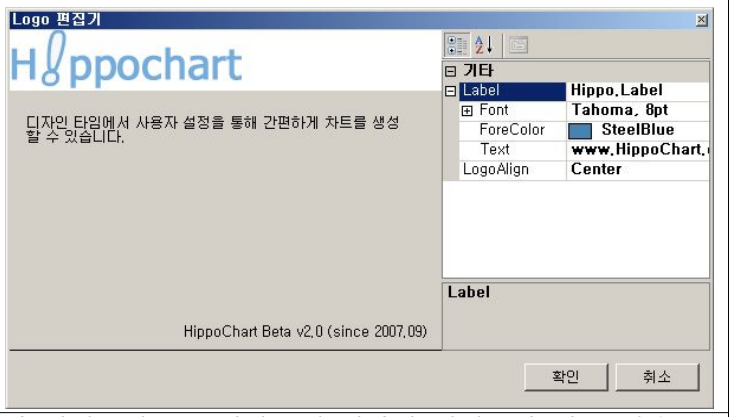
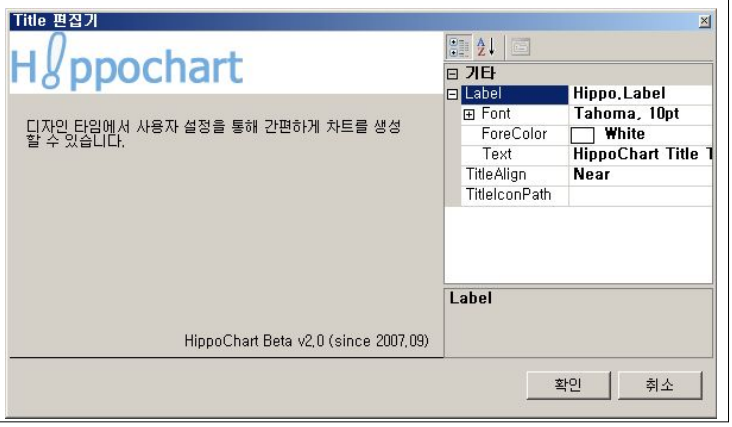
웹 컨트롤의 속성은 윈도우즈 컨트롤과 몇 가지 차이가 있습니다. 하나씩 자세히 알아보겠습니다.



히포차트 웹 폼 디자이너 속성

속성	설명
Alt	이 속성은 웹 차트 컨트롤의 간단한 설명을 달 수 있는 기능입니다. <img> 객체의 alt와 동일합니다.
Designer	차트 디자이너는 차트의 배경 디자인을 관리합니다. 
DesignType	히포차트의 외곽 배경 디자인을 관리하는 열거형 타입의 속성으로 아래와 같이 5가지 타입과 None으로 구성되어 있습니다.(None은 설정 사항이 아닙니다.) 
Direction	이 속성은 차트를 다중 시리즈리스트로 구성할 경우 시리즈리스트가 추가되는 방향을 의미합니다.  기본 값은 수평방향입니다.
ImageURL	이미지 상대 경로를 설정합니다. 기본 값은 websampleWHippoChart1.Png 으로 가상디렉토리의 root에 저장되며 예를 들어 images/chart 하위에 chart1.png 라는 파일로 저장을 하기를 원한다면 “images/chart/chart1.png” 와 같은 형태로 설정하면 되

	겠습니다. ※히포차트는 서버에서 차트를 작성 한 후 이미지를 저장하기 때문에 반드시 경로 설정이 필요합니다.
IsEventLog	개발 시 에러가 발생할 경우 로그를 기록하는 기능 중 하나로 이벤트뷰어에 내용을 등록합니다. 단, 개발 단계에서만 사용이 권장되며 배포 시에는 반드시 false를 해야 계속적인 이벤트 로그를 기록하는 일을 방지할 수 있습니다. (※이 속성은 베타 버전의 테스트를 위한 속성으로 정식 버전 출시 시에는 제거될 수 있습니다.)
IsFileLog	에러 로그 기록 중 하나로 파일시스템을 이용해서 로그를 기록합니다. IsEventLog 보다 사용이 권장되며 배포 초기에 true로 설정해 놓으면 초기 버그를 잡는데 유용합니다. 단, 한 번 그릴 때마다 로그파일을 작성하므로 실시간 차트일 경우 권장되지 않으며 안정화 단계에서는 false로 처리하는 것이 좋습니다. (※이 속성은 베타 버전의 테스트를 위한 속성으로 정식 버전 출시 시에는 제거될 수 있습니다.)
LegendBox	범례(외부 범례)의 디자인타임 속성입니다. 범례에 대한 자세한 내용은 뒤에서(기능편)에서 자세히 알아보겠습니다. 
Logo	차트 하단에 표시되는 간단한 문구인 로고의 속성입니다.

	역시 기능편에서 자세히 알아보겠습니다. 
SeriesAreaRate	한 시리즈리스트 내에 2개 이상의 시리즈가 있을 경우 그 공간의 비율을 설정하는 기능입니다. 설정을 하지 않을 경우는 각 시리즈 별로 동일한 공간이 할당 됩니다.
Text	히포차트의 버전을 표시합니다(읽기전용)
Titles	차트 상단에 표시되는 차트의 제목을 나타내는 속성입니다. 

(표23 - 웹 컨트롤 디자이너 속성)

TIP.

히포차트 웹 컨트롤의 속성들을 디자인 타임에서 수정할 경우 즉시 변경된 모습이 보이지 않을 수 있는데 이럴 경우 아래와 같이 새로고침을 하면 바로 반영된다.

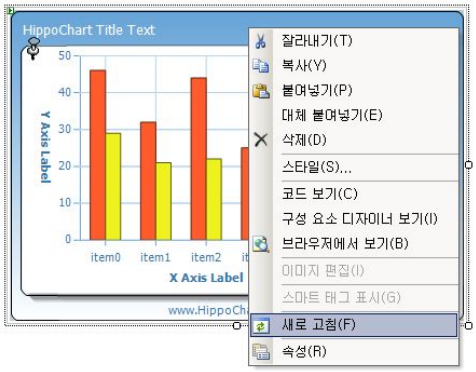


차트 그리기

이제 웹 환경에서 히포차트를 한 번 작성해보겠습니다. 앞서 윈도우즈 폼 코드를 설명하면서 시작하기에 관해 설명을 마쳤으므로 웹에서의 특이사항 중심으로 작성해보겠습니다.

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    DrawChart();
}

private void DrawChart()
{
    SeriesList sList = new SeriesList();
    sList.ChartType = ChartType.Column;

    this.WHippoChart1.SeriesListDictionary.Add(sList);

    this.WHippoChart1.ImageURL = "images/chart1.png";
    this.WHippoChart1.DrawChart();
}
```

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    DrawChart()
End Sub

Private Sub DrawChart()
    Dim sList As New SeriesList()
    sList.ChartType = ChartType.Column

    Me.WHippoChart1.SeriesListDictionary.Add(sList)

    Me.WHippoChart1.ImageURL = "images/chart/chart1.png"
    Me.WHippoChart1.DrawChart()
End Sub
```

우선, 시작을 위와 같이 작성합니다. 윈도우즈 폼의 개발과 다른 점이라면 그려진 차트가 저장될 경로를 지정했다는 점입니다. images/chart 폴더 하위에 chart1.png 라는 이름을 주도록 하고 있는데 실제 프로젝트에서는 아래와 같은 디렉토리 구조를 가지겠습니다.

만약 “일별 사용자 방문 현황” 이라는 차트를 구성한다면 이름을 아래와 같이 런타임 시에 변경되도록 구성할 수도 있습니다.

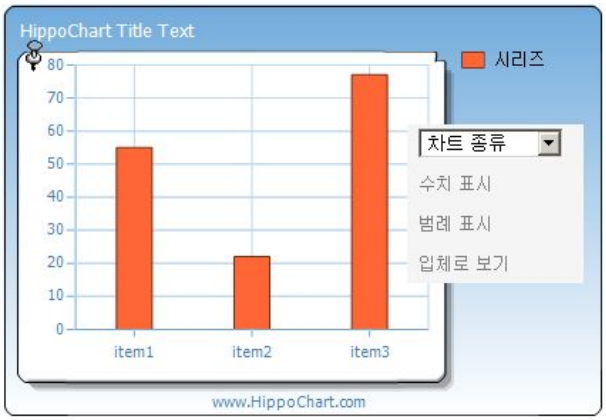
```
this.WHippoChart1.ImageURL = "images/chart/chart" + DateTime.Now.ToString("yyyyMMdd") + ".png";
```

TIP.

이미지 이름 설정 시 매번 그릴 때 마다 이름을 변경한다면 서버에 너무 많은 이미지가 생성될 수 있으니 유의하세요.

자! 그럼 간단한 웹 어플리케이션을 한 번 만들어 볼까요?

만들어 볼 기능은 아래 그림에 잘 나와 있습니다. 웹에서는 윈도우즈 어플리케이션 환경과 달리 많은 제약이 있어 다양한 사용자 기능을 삽입하기가 녹녹하지 않은데 아래와 같이 레이어를 이용해 어느 정도 구현이 가능합니다.



먼저, 아래 코드와 같이 컨텍스트 메뉴처럼 보이게 하는 div 를 구성합니다. 내부에는 table이 있고 차트 종류를 나타내는 드롭다운리스트와 수치, 범례, 입체 보기를 나타내는 링크 버전이 3개가 있습니다.

```
<div style="display:none; position:absolute; left: 374px; top: 179px;" id="conmenu">
  <table style="width: 124px; height: 111px; background-color:#f5f5f5">
   <tr>
<td style="height: 11px">
    <asp:DropDownList ID="drpChartType" runat="server" AutoPostBack="True"
 OnSelectedIndexChanged="drpChartType_SelectedIndexChanged" Width="102px">
       <asp:ListItem Value="-1">차트종류</asp:ListItem>
       <asp:ListItem Value="0">라인차트</asp:ListItem>
       <asp:ListItem Value="1">컬럼차트</asp:ListItem>
       <asp:ListItem Value="2">파이차트</asp:ListItem>
   </asp:DropDownList></td>
</tr>
<tr>
<td>
    <asp:LinkButton ID="lnkFigures" runat="server"
OnClick="lnkFigures_Click" Font-Size="9pt" Font-Underline="False"
```

```
ForeColor="Gray">수치표시</asp:LinkButton></td></tr>
<tr>
   <td>
    <asp:LinkButton ID="lnkLegend" runat="server"
OnClick="lnkLegend_Click"  Font-Size="9pt"  Font-Underline="False"  ForeColor="Gray">범
례표시</asp:LinkButton></td>
</tr>
<tr>
     <td>
    <asp:LinkButton  ID="lnk3D"  runat="server"  OnClick="lnk3D_Click"  Font-Size="9pt"
Font-Underline="False"
ForeColor="Gray">입체로보기</asp:LinkButton></td>
</tr>
</table>
</div>
```

서버 코드를 잠시 살펴보겠습니다. 아래는 폼로드 이벤트인데 차트에 자바스크립트 클릭 이벤트 특징을 주어 컨텍스트메뉴 처럼 활성화를 시켜줍니다. 그리고 아래 유지해야할 변수들을 초기화하고 차트를 그리는 메소드를 호출하고 있습니다.

ASP.NET에서는 '포스트백'이라는 새로운 개념으로 데이터를 submit 하기 때문에 여기서 유지해야할 4가지 변수들(차트 종류, 입체여부, 수치표시 여부, 범례표시여부)은 모두 ViewState를 이용한 프로퍼티로 구성이 되어 있습니다.

```
protected void Page_Load(object sender, EventArgs e)
{
    this.WHippoChart1.Attributes.Add("onclick", "javascript:viewmenu();");

    if (!IsPostBack)
    {
        this.charts = ChartType.Column;
        this.Is3D = false
        this.IsFigure = false
        this.IsShowLegend = false

        DrawChart();
    }
}
```

아래는 div를 보이게 하는 html 소스입니다.

```
<script language="javascript" type="text/javascript">
        function viewmenu()
        {
             var rightmenu = document.getElementById("conmenu");

             rightmenu.style.display = "block";
        }
</script>
```

아래는 '수치표시' 라는 링크 버튼의 이벤트 핸들러인데, 현재 상태를 비교해서 수치가 보이는 상태이면 안보이게, 안 보이는 상태이면 보이게 설정을 하고 있습니다.

```
protected void lnkFigures_Click(object sender, EventArgs e)
{
    if (this.IsFigure)
    {
        this.IsFigure = false
    }
    else
    {
        this.IsFigure = true
    }

    DrawChart();
}
```

아래 전체 소스를 참고하여 더욱 응용하여 사용하시기 바랍니다.

HTML

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs" Inherits="_Default"
%>
<%@ Register Assembly="hwebchart3" Namespace="Hippo.WebChartControl" TagPrefix="Hippo" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>제목없음</title>
  <script language="javascript" type="text/javascript">
        function viewmenu()
        {
            var rightmenu = document.getElementById("conmenu");
            rightmenu.style.display="block"
        }
 </script>
</head>
<body>
    <form id="form1" runat="server">
<table style="width: 578px; height: 401px">
            <tr>
                <td style="width: 745px; background-color:steelblue">
                     <br />
                    <strong><span style="font-size: 16pt; color: white;">  웹차트를
그려보자<br />
                    </span></strong></td>
</tr>
             <tr>
                <td align="center" style="width: 745px">
                <Hippo:wHippoChart ID="WHippoChart1" runat="server" Height="288px"
 Width="421px">
            <LegendBox DivideLineColor="Gray" FigureFormat="Number"
Height="213.239578" IsAutoSetting="True" IsDivideLine="True" IsShowHeader="False"
IsVerticalLine="False" LegendFormat="Icon_Name" Location="Default" PercentDecimalPoint="0"
ValueDecimalPoint="-2147483648" ValueType="Average" VerticalLineColor="Gray" Visible="True"
Width="95.58289" />
            <Titles TitleAlign="Near">
                <Label Font="Tahoma, 10pt" ForeColor="White" Text="HippoChart Title Text" />
            </Titles>
            <Logo LogoAlign="Center">
                <Label Font="Tahoma, 8pt" ForeColor="SteelBlue" Text="www.HippoChart.com" />
            </Logo>
            <Designer BackColor="113, 171, 220" BackLineColor="Black" InnerBackColor="White"
                IsGradation="True" IsShowBorder="True" />
        </Hippo:wHippoChart>
                </td>
</tr>

</table>

<div style="display:none; position:absolute; left: 374px; top: 179px;" id="conmenu">
         <table style="width: 124px; height: 111px; background-color:#f5f5f5">
            <tr>
```

```
<td style="height: 11px">
    <asp:DropDownList ID="drpChartType" runat="server"
 AutoPostBack="True" OnSelectedIndexChanged="drpChartType_SelectedIndexChanged" Width="102px">
        <asp:ListItem Value="-1">차트종류</asp:ListItem>
        <asp:ListItem Value="0">라인차트</asp:ListItem>
        <asp:ListItem Value="1">컬럼차트</asp:ListItem>
         <asp:ListItem Value="2">파이차트</asp:ListItem>
     </asp:DropDownList></td>
</tr>
<tr>
<td>
   <asp:LinkButton ID="lnkFigures" runat="server" OnClick="lnkFigures_Click" Font-Size="9pt"
Font-Underline="False" ForeColor="Gray">수치표시</asp:LinkButton></td>      </tr>
<tr>
   <td>
 <asp:LinkButton ID="lnkLegend" runat="server" OnClick="lnkLegend_Click" Font-Size="9pt"
Font-Underline="False" ForeColor="Gray">범례표시</asp:LinkButton></td>
</tr>
<tr>
   <td>
   <asp:LinkButton ID="lnk3D" runat="server" OnClick="lnk3D_Click" Font-Size="9pt"
Font-Underline="False" ForeColor="Gray">입체로보기</asp:LinkButton></td>
</tr>
</table>
      </div>
     </form>
</body>
</html>
```

C#

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

using Hippo;

public partial class _Default : System.Web.UI.Page
{
    public ChartType charts
```

```
{
    get { return (ChartType)ViewState["charttype"]; }
    set { ViewState["charttype"] = value; }
}

public bool Is3D
{
    get { return (bool)ViewState["3d"]; }
    set { ViewState["3d"] = value; }
}

public bool IsFigure
{
    get { return (bool)ViewState["Figure"]; }
    set { ViewState["Figure"] = value; }
}

public bool IsShowLegend
{
    get { return (bool)ViewState["Legend"]; }
    set { ViewState["Legend"] = value; }
}

protected void Page_Load(object sender, EventArgs e)
{
    this.WHippoChart1.Attributes.Add("onclick", "javascript:viewmenu();");

    if (!IsPostBack)
    {
        this.charts = ChartType.Column;

        this.Is3D = false;
        this.IsFigure = false;
        this.IsShowLegend = false;

        DrawChart();
    }
}

private void DrawChart()
{
    SeriesList sList = new SeriesList();
    sList.ChartType = this.charts;

    // 입체
    sList.IsShow3D = this.Is3D;

    Series sr = new Series();
    sr.Name = "시리즈";
```

```
        SeriesItem item1 = new SeriesItem(55);
        SeriesItem item2 = new SeriesItem(22);
        SeriesItem item3 = new SeriesItem(77);

        item1.Name = "item1";
        item2.Name = "item2";
        item3.Name = "item3";

        // 수치
        item1.IsShowFigureText = this.IsFigure;
        item2.IsShowFigureText = this.IsFigure;
        item3.IsShowFigureText = this.IsFigure;

        sr.items.Add(item1);
        sr.items.Add(item2);
        sr.items.Add(item3);

        sList.SeriesCollection.Add(sr);

        // 범례
        this.WHippoChart1.LegendBox.Visible = this.IsShowLegend;

        this.WHippoChart1.SeriesListDictionary.Add(sList);

        this.WHippoChart1.ImageURL = "images/chart/chart" + DateTime.Now.ToString("yyyyMMdd")
+ ".png";
        this.WHippoChart1.DrawChart();
    }

    protected void drpChartType_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (drpChartType.SelectedIndex == 1)
        {
            this.charts = ChartType.Line;
        }
        else if (drpChartType.SelectedIndex == 2)
        {
            this.charts = ChartType.Column;
        }
        else if (drpChartType.SelectedIndex == 3)
        {
            this.charts = ChartType.Pie;
        }

        DrawChart();
    }

    protected void lnkFigures_Click(object sender, EventArgs e)
```

```
    {
        if (this.IsFigure)
        {
            this.IsFigure = false;
        }
        else
        {
            this.IsFigure = true;
        }

        DrawChart();
    }

    protected void lnkLegend_Click(object sender, EventArgs e)
    {
        if (this.IsShowLegend)
        {
            this.IsShowLegend = false;
        }
        else
        {
            this.IsShowLegend = true;
        }

        DrawChart();
    }

    protected void lnk3D_Click(object sender, EventArgs e)
    {
        if (this.Is3D)
        {
            this.Is3D = false;
        }
        else
        {
            this.Is3D = true;
        }

        DrawChart();
    }
}
```

VB

```
Imports System
Imports System.Data
Imports System.Configuration
```

```
Imports System.Web
Imports System.Web.Security
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.Web.UI.WebControls.WebParts
Imports System.Web.UI.HtmlControls

Imports Hippo

Partial Public Class _Default
    Inherits System.Web.UI.Page
    Public Property charts() As ChartType
        Get
            Return DirectCast(ViewState("charttype"), ChartType)
        End Get
        Set(ByVal value As ChartType)
            ViewState("charttype") = value
        End Set
    End Property

    Public Property Is3D() As Boolean
        Get
            Return CBool(ViewState("3d"))
        End Get
        Set(ByVal value As Boolean)
            ViewState("3d") = value
        End Set
    End Property

    Public Property IsFigure() As Boolean
        Get
            Return CBool(ViewState("Figure"))
        End Get
        Set(ByVal value As Boolean)
            ViewState("Figure") = value
        End Set
    End Property

    Public Property IsShowLegend() As Boolean
        Get
            Return CBool(ViewState("Legend"))
        End Get
        Set(ByVal value As Boolean)
            ViewState("Legend") = value
        End Set
    End Property

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
        Me.WHippoChart1.Attributes.Add("onclick", "javascript:viewmenu();")
```

```
        If Not IsPostBack Then
            Me.charts = ChartType.Column

            Me.Is3D = False
            Me.IsFigure = False
            Me.IsShowLegend = False

            DrawChart()
        End If
    End Sub

    Private Sub DrawChart()
        Dim sList As New SeriesList()
        sList.ChartType = Me.charts

        ' 입체
        sList.IsShow3D = Me.Is3D

        Dim sr As New Series()
        sr.Name = "시리즈"

        Dim item1 As New SeriesItem(55)
        Dim item2 As New SeriesItem(22)
        Dim item3 As New SeriesItem(77)

        item1.Name = "item1"
        item2.Name = "item2"
        item3.Name = "item3"

        ' 수치
        item1.IsShowFigureText = Me.IsFigure
        item2.IsShowFigureText = Me.IsFigure
        item3.IsShowFigureText = Me.IsFigure

        sr.items.Add(item1)
        sr.items.Add(item2)
        sr.items.Add(item3)

        sList.SeriesCollection.Add(sr)

        ' 범례
        Me.WHippoChart1.LegendBox.Visible = Me.IsShowLegend

        Me.WHippoChart1.SeriesListDictionary.Add(sList)

        Me.WHippoChart1.ImageURL = "images/chart/chart" & DateTime.Now.ToString("yyyyMMdd") &
".png"
        Me.WHippoChart1.DrawChart()
```

```
    End Sub

    Protected Sub drpChartType_SelectedIndexChanged(ByVal sender As Object, ByVal e As
EventArgs)
        If drpChartType.SelectedIndex = 1 Then
            Me.charts = ChartType.Line
        ElseIf drpChartType.SelectedIndex = 2 Then
            Me.charts = ChartType.Column
        ElseIf drpChartType.SelectedIndex = 3 Then
            Me.charts = ChartType.Pie
        End If

        DrawChart()
    End Sub

    Protected Sub lnkFigures_Click(ByVal sender As Object, ByVal e As EventArgs)
        If Me.IsFigure Then
            Me.IsFigure = False
        Else
            Me.IsFigure = True
        End If

        DrawChart()
    End Sub

    Protected Sub lnkLegend_Click(ByVal sender As Object, ByVal e As EventArgs)
        If Me.IsShowLegend Then
            Me.IsShowLegend = False
        Else
            Me.IsShowLegend = True
        End If

        DrawChart()
    End Sub
    Protected Sub lnk3D_Click(ByVal sender As Object, ByVal e As EventArgs)
        If Me.Is3D Then
            Me.Is3D = False
        Else
            Me.Is3D = True
        End If

        DrawChart()
    End Sub
End Class
```

실전 히포차트 개발하기(2)

- 기능편 -

히포차트 기초편을 통해 디자인하기와 윈도우즈 폼 개발 방법, 실시간 차트 그리기, WPF에서의 적용, 웹에서의 차트 그리기 등을 알아보았습니다. 히포차트를 한 번 이상 다루어 보신 분들은 다소 평이한 내용이었을 수 있습니다.
이번 장부터는 보다 상세하게 세부 객체들에 대해 알아볼텐데요. 히포차트를 구성하는 각각의 내부 객체들을 기능별로 분리하여 쉽게 사용할 수 있도록 진행하겠습니다.

1. 타이틀(Titles)

타이틀은 차트의 제목을 의미합니다. 타이틀의 위치는 차트 레이아웃의 상단으로 고정되어 있고 라벨 객체를 통해 폰트 등의 설정이 가능합니다.

타이틀 속성

속성	설명
Label	앞에서 알아본 문자를 나타내는 기초 클래스로 타이틀의 폰트, 글자색, 글자크기 등을 설정합니다.
TitleAlign	타이틀의 좌우 위치를 설정합니다. StringAlignment.Center (가운데) StringAlignment.Far(오른쪽) StringAlignment.Near(왼쪽: 기본값)
TitleIconPath	(현재 지원하지 않습니다.)

(표24 - 타이틀 속성)

샘플 코드

```
this.hHippoChart1.Titles.Label.Text= "히포차트타이틀입니₩r₩n(알맞은제목을넣어주세요";
this.hHippoChart1.Titles.Label.Font = new Font("굴림, 12, FontStyle.Bold);
this.hHippoChart1.Titles.TitleAlign = StringAlignment.Near;
```

2. 로고(Logo)

로고는 차트 하단에 들어가는 짧은 문구를 말합니다. 회사 홈페이지 주소, 차트 비고 등 다양하게 활용이 가능합니다.

로고 속성

속성	설명
Label	앞에서 알아본 문자를 나타내는 기초 클래스로 로고의 폰트, 글자색, 글자크기 등을 설정합니다.
LogoAlign	타이틀의 좌우 위치를 설정합니다. StringAlignment.Center (가운데:기본값) StringAlignment.Far(오른쪽) StringAlignment.Near(왼쪽)

(표25 - 로고 속성)

샘플 코드

```
this.hHippoChart1.Logo.Label.Text = "www.hippochart.com";
this.hHippoChart1.Logo.LogoAlign = StringAlignment.Near;
```

3. 차트 디스크립션(Description)

차트 디스크립션은 차트에 간단한 “설명 문구”를 나타내는 객체입니다. 그래프 영역의 4등분하여 좌상, 좌하, 우상, 우하 4군데에 디스크립션을 표시할 수 있습니다. 기본값은 기울어진 형태의 회색 글자를 가지고 있습니다.

디스크립션 속성

속성	설명
Label	앞에서 알아본 문자를 나타내는 기초 클래스로 디스크립션의 폰트, 글자색, 글자크기 등을 설정합니다.
XDirection	디스크립션의 좌우 위치를 설정합니다. 이 속성은 반드시 left 혹은 right 로만 설정되어야합니다.
YDirection	디스크립션의 상하 위치를 설정합니다. 이 속성은 반드시 Top 혹은 Bottom으로만 설정되어야합니다.

(표26 - 디스크립션 속성)

샘플 코드

```
SeriesList slist = new SeriesList();
slist.Description.Label.Text = "이차트는내부용입니다";
slist.Description.XDirection = AxisDirection.Left;
slist.Description.YDirection = AxisDirection.Bottom;
```

4. 축(Axis)과 그리드(Grid)

히포차트의 좌표 영역은 기본적인 형태에서 크게 2개의 축과 내부 그리드로 구성되어 있습니다.

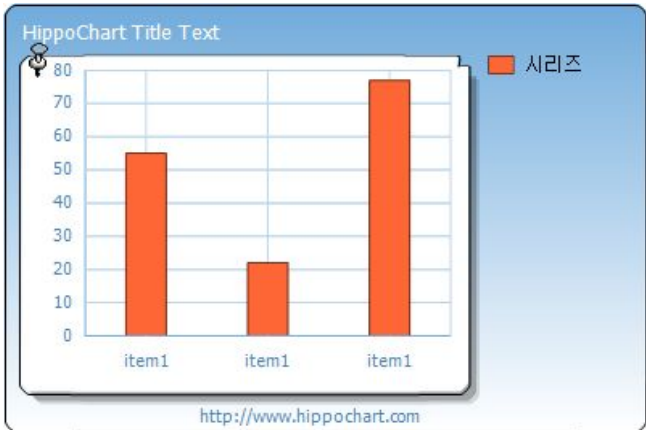
축(Axis)

히포차트 좌표(AxisFactor)에는 기본적으로 2개의 축이 존재합니다. X축은 데이터의 이름을 의미하기도 하고 분산형 차트일 경우는 수를 의미하기도 합니다. Y축은 실제 데이터 수치를 의미하며 숫자일 수도 있고 간트차트일 경우는 DateTime 일 수도 있습니다.

(1) 축 눈금 안보이기

개인적인 취향 혹은 디자인 상의 문제로 눈금을 제거해야할 경우가 있을 경우 사용하는 기능입니다.

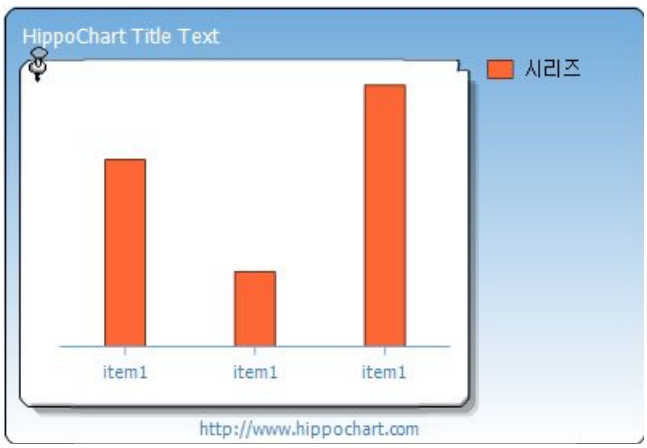
```
slist.AxisFactor.YAxis.IsShowTick = false;
slist.AxisFactor.XAxis.IsShowTick = false;
```



(2) 축 안보이기

축 자체를 안보이게 설정함으로써 이색적인 표현이 가능하고 그리드와 함께 설정할 경우 다양한 디자인을 연출할 수 있습니다.

```
slist.AxisFactor.YAxis.Visible = false;
slist.AxisFactor.XAxis.Visible = true;
slist.GraphArea.Grid.GridDirection = GridDirection.None;
```



(3) 축 수동 설정하기

히포차트는 기본적으로 사용자의 데이터를 분석하여 자동으로 Y축의 눈금 단계를 결정합니다. 하지만, 데이터의 특성상 최대값 또는 최소값이 고정되어야 하거나 interval이 고정일 경우 등 축 단계의 설정을 사용자 임의로 변경해야할 경우를 위해 수동으로 설정할 수 있는 기능을 이용할 수 있습니다.

수동으로 설정하는 방법은 속성을 이용하는 방법과 메소드를 이용하는 방법이 있습니다.

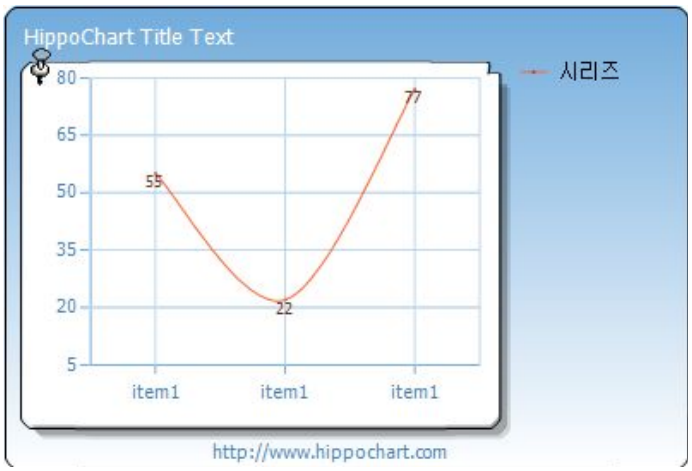
속성 방식

```
slist.AxisFactor.YAxis.IsAutoSetting = false;
slist.AxisFactor.YAxis.MinUnitValue = 5;
slist.AxisFactor.YAxis.MaxUnitValue = 80;
slist.AxisFactor.YAxis.Interval = 15;
```

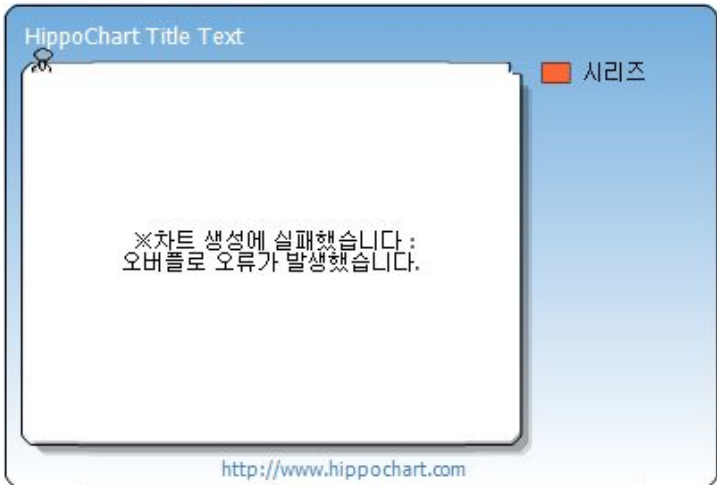
메소드 방식

```
slist.AxisFactor.YAxis.SetAxisStep(5, 80, 15);
```

위 두 가지 방법은 동일한 결과를 도출합니다.



히포차트 코어 엔진이 복잡한 연산을 통해 결정하는 축 단계를 사용자 임의로 설정하는 것인 만큼 그 값에 대해 주의를 요합니다. 예를 들어 최소 단위 값이 최대 단위 값보다 크거나 인터벌이 최대 단위 값보다 크거나 인터벌을 설정하지 않은 경우 등은 아래와 같은 심각한 에러를 유발할 수 있습니다.

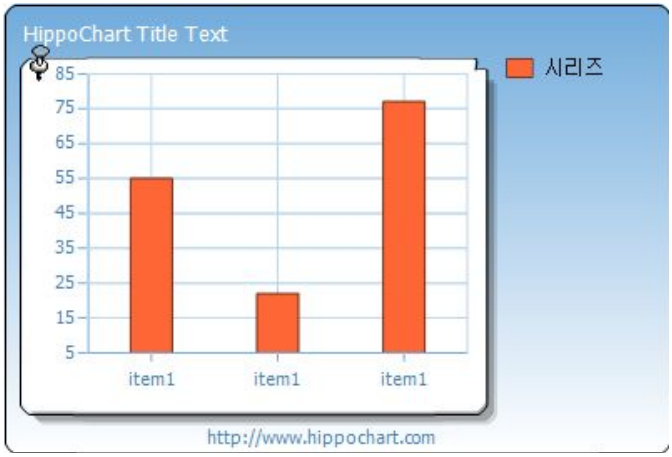


또한, 사용하는 3가지 값은 일련의 연관성이 있어야합니다. 위 예제를 보면 5, 80, 15로 구성이 되어 있는데 이는 5부터 시작해서 15씩 더해서 80까지 단계를 표시하겠다는 의미입니다.

실제 5에서 15씩 더해보면 아래와 같이 완벽하게 나누어짐을 알 수 있습니다.

5 , 20, 35, 50, 65, 80

예를 들어, 위 설정에서 인터벌을 10으로 설정한다면 히포차트 코어 엔진에서는 아래와 같이 최대 단위 값을 85로 설정함으로써 보다 올바른 방향으로 재설정을 합니다.



축의 수동 기능은 Axis 클래스의 기능으로 추가된 축(멀티축)에서도 동일하게 적용되며 분산형 차트일 경우 X축에도 마찬가지로 적용이 됩니다.

X축은 **분산형 차트**일 경우에 적용을 할 수 있는데 아래 코드를 참고하세요.

C#

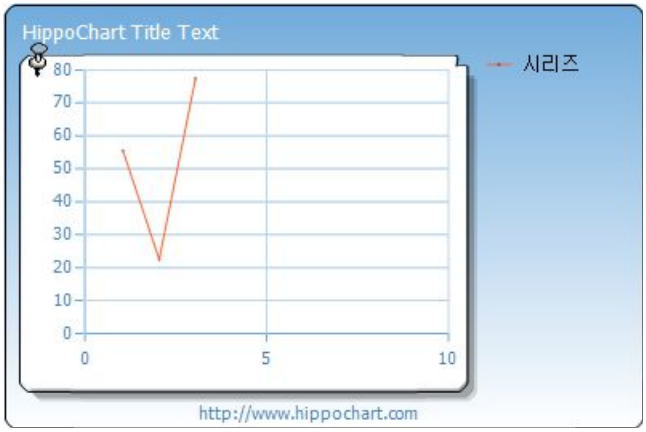
```
SeriesList slist = new SeriesList();

slist.AxisFactor.XAxis.DataType = AxisDataType.Number;
slist.AxisFactor.XAxis.SetAxisStep(0, 4, 1);

Series sr = new Series();
sr.Name = "시리즈";

SeriesItem item1 = new SeriesItem(55);
SeriesItem item2 = new SeriesItem(22);
SeriesItem item3 = new SeriesItem(77);

item1.XValue = 1;
item2.XValue = 2;
item3.XValue = 3;

sr.items.Add(item1);
sr.items.Add(item2);
sr.items.Add(item3);

slist.SeriesCollection.Add(sr);
```

```
this.hHippoChart1.SeriesListDictionary.Add(slist);
this.hHippoChart1.DrawChart();
```

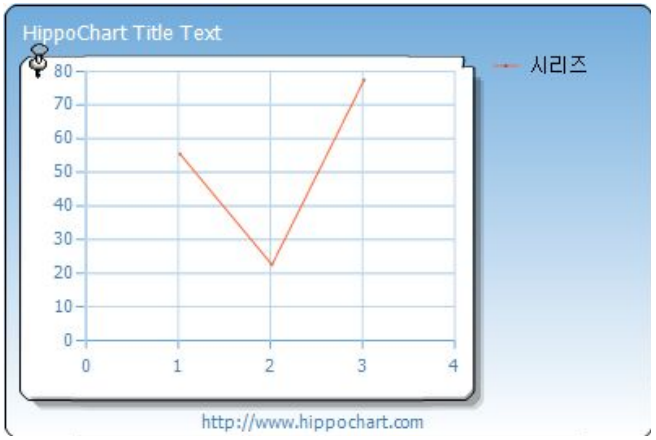
VB

```
 Dim slist As New SeriesList()

 slist.AxisFactor.XAxis.DataType = AxisDataType.Number
 slist.AxisFactor.XAxis.SetAxisStep(0, 4, 1)

 Dim sr As New Series()
 sr.Name = "시리즈"

 Dim item1 As New SeriesItem(55)
 Dim item2 As New SeriesItem(22)
 Dim item3 As New SeriesItem(77)

 item1.XValue = 1
 item2.XValue = 2
 item3.XValue = 3

 sr.items.Add(item1)
 sr.items.Add(item2)
 sr.items.Add(item3)

 slist.SeriesCollection.Add(sr)

 Me.HHippoChart1.SeriesListDictionary.Add(slist)
 Me.HHippoChart1.DrawChart()
```

분산형 차트로 그리는 방법은 X축의 데이터타입(DataType)을 Number로 설정하고 시리즈아이템의 XValue 에 해당 값을 넣어주면 됩니다. 그러면 X축을 숫자로 인식하여 일반 Y축을 설정하듯이 각 단계 값을 계산하여 축을 생성하는데 이 역시 위와 같이 수동으로 처리할 수 있습니다.



(자동으로 처리할 경우 위 차트와 같이 빈 공간이 생길 수 있습니다.)

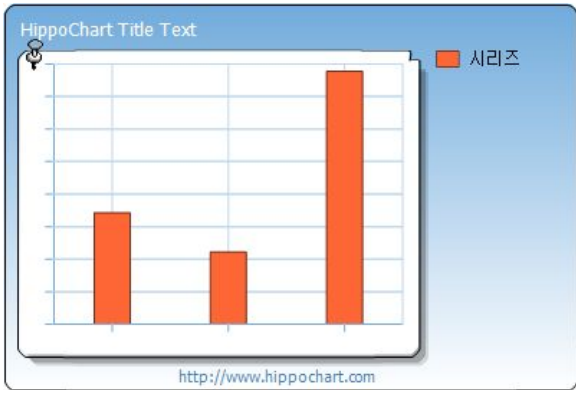


(이 차트는 위 샘플 코드로 수동 처리하여 보기 좋은 모양으로 만든 예입니다.)

(4) 축 단계 수치 Visible

히포차트는 축의 단계 수치 보이기/안보이기를 조절할 수 있어 다양한 효과를 줄 수 있습니다.

```
slist.AxisFactor.YAxis.IsVisibleFigures = false;
slist.AxisFactor.XAxis.IsVisibleFigures = false;
```



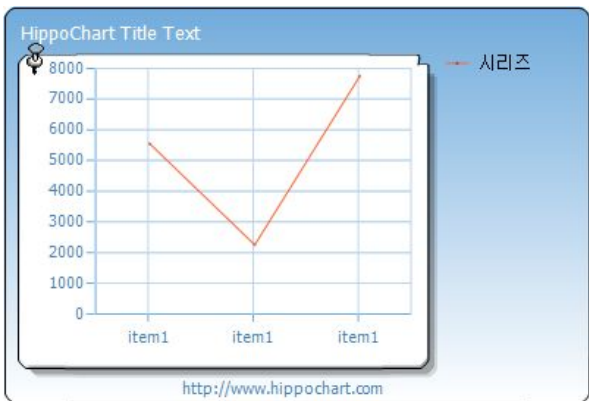
(5) FigureFormat(숫자포맷)와 Decimalpoint로 다양한 데이터 표현하기

차트를 그리고자 하는 데이터는 참으로 다양합니다. 그냥 일반적인 정수형의 수치일 수도 있고 퍼센트 값을 나타내는 소수일수도 있으며 통화를 나타낼 수도 있습니다. 또한, 같은 통화일지라도 한화, 달러화 등 많은 경우가 있습니다. 히포차트에서는 이러한 다양한 환경을 적용할 수 있도록 FigureFormat과 Decimalpoint라는 속성을 축(Axis)의 멤버로 제공하고 있습니다.

1) None

숫자포맷의 기본형입니다. 어떠한 설정도 하지 않고 숫자 그대로를 보여주며(마치 문자처럼) Decimalpoint를 설정하면 안 됩니다.

```
slist.AxisFactor.YAxis.FigureFormat = FigureFormat.None;
```

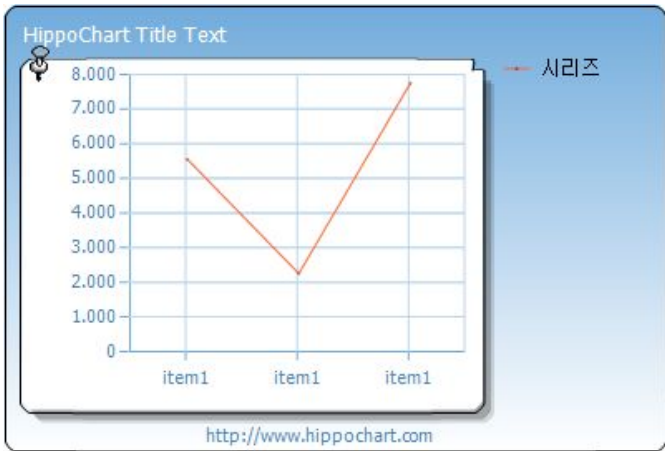


2) Number

숫자를 각 문화권별 고유의 양식으로 표현합니다. 예를 들어 한국의 경우 10000이라는 숫자는 10,000와 같이 3자리마다 콤마를 찍는 형식입니다. 스페인 문화권(es-ES)인 경우는 1255.34라는 소수가 있을 때 1.255,34와 같이 콤마와 콜론이 반대로 표시됩니다.

아래 숫자포맷과 더불어 Decimalpoint가 0으로 설정이 되어 있는데 이는 기본적으로 Number형으로 설정할 경우 Decimalpoint가 2로 설정되기 때문에 정수인 경우에는 0으로 처리해 주어야 합니다.

```
slist.AxisFactor.YAxis.FigureFormat = FigureFormat.Number;
slist.AxisFactor.YAxis.Decimalpoint = 0;
slist.AxisFactor.YAxis.CultureInfo.Name = "ko-KR";
```



TIP.

Decimalpoint는 축(Axis)의 속성이다. 즉, 기본 축(YAxis)을 토대로 그려지는 모든 시리즈들은 이 축의 Decimalpoint를 따르게 된다. 그러나 뒤에서 나올 멀티 축을 이용하면 다른 숫자 포맷과 Decimalpoint를 설정할 수 있다.

3) Percent

퍼센트는 백분율을 의미하므로 예를 들어 0.1이면 10%, 0.34 이면 34% 로 표기가 됩니다. 즉, 데이터가 56%인데 56 그대로 관리가 되고 있다면 100을 나누

어 주는 작업을 해주어야합니다.

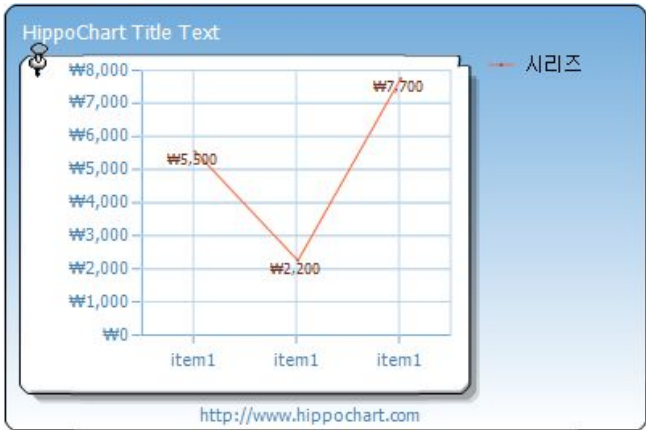
Percent 역시 Decimalpoint를 통해 소수 몇 째 자리까지 표시할 것인지를 설정할 수 있는데 설정을 하지 않을 경우 아래 이미지처럼 소수 둘째자리까지 표시됩니다.

```
slist.AxisFactor.YAxis.FigureFormat = FigureFormat.Percent;
```

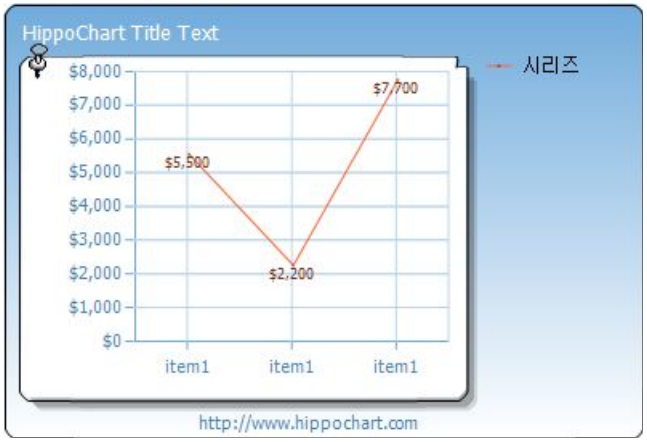
4) Currency

Currency는 통화 형태의 숫자를 의미합니다. 통화는 Number형과 마찬가지로 문화권 정보의 영향을 받기 때문에 아래 코드와 같이 문화권 정보를 같이 설정해 주어야합니다. 단, 기본 값이 ko-KR로 한국이므로 다른 문화권일 경우만 해당합니다.

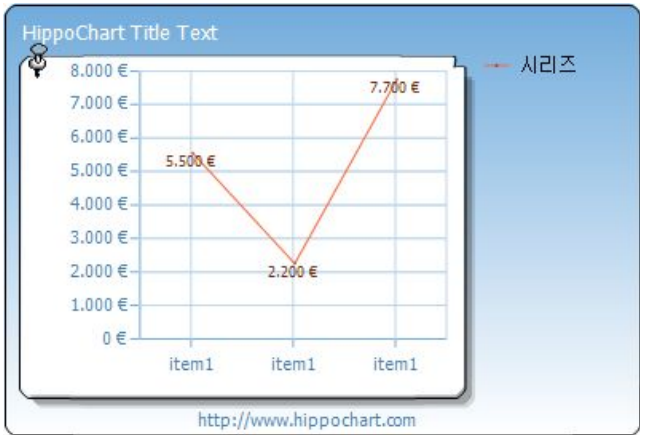
```
slist.AxisFactor.YAxis.FigureFormat = FigureFormat.Currency;
slist.AxisFactor.YAxis.Decimalpoint = 0;
slist.AxisFactor.YAxis.CultureInfo.Name = "en-US";
```



(ko-KR)



(en-US)

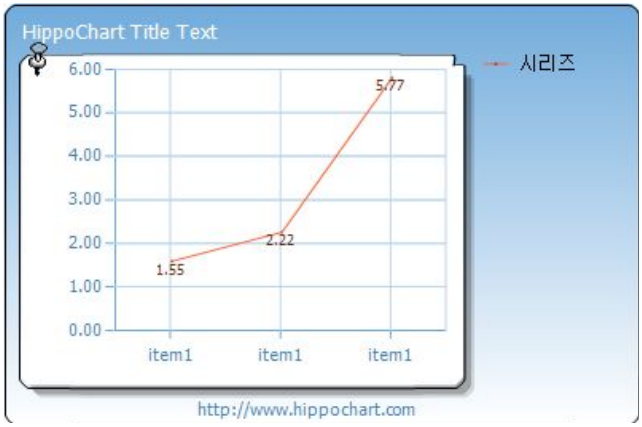


(es-ES)

5) FixedPoint

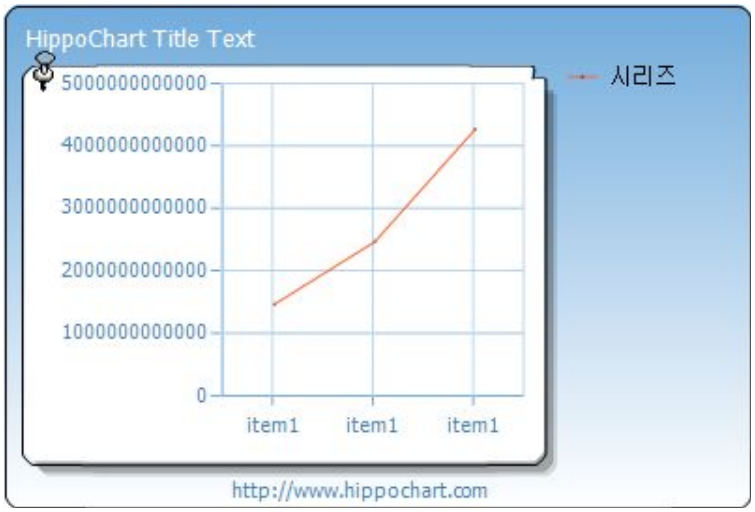
데이터가 부동 소수일 경우 사용하면 유용합니다.

```
slist.AxisFactor.YAxis.FigureFormat = FigureFormat.FixedPoint;
```



6) Exponential

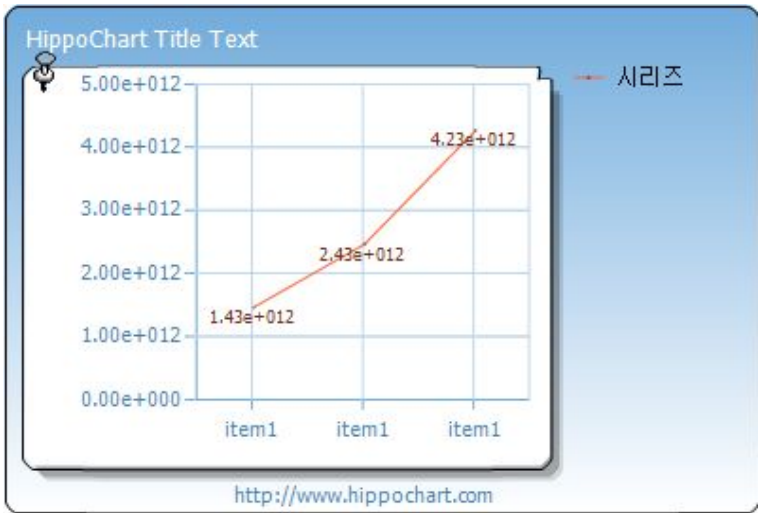
이 타입은 상당히 주목을 해야 할 타입입니다. 차트를 그리다 보면 엄청나게 큰 수치를 그려야 할 경우가 있습니다. 예를 들어 1,478,332,553,000 라는 값을 차트로 표현해야 한다면 히포차트의 범위로 그릴 수 있는 숫자이지만 그 문자 자체가 길기 때문에 차트로 그릴 경우 상당히 보기 안 좋을 수 있습니다.



하지만, 이 Exponential 타입을 이용하면 아주 큰 수에 대해 원활하게 처리할 수 있습니다.

```
slist.AxisFactor.YAxis.FigureFormat = FigureFormat.Exponential;
slist.AxisFactor.YAxis.Decimalpoint = 2;
```

와 같이 설정하면 아래와 같이 지수 형태로 표시해줌으로써 보다 보기 좋게 표현이 가능합니다.



7) Custom

(커스텀 타입은 현재 지원하지 않습니다.)

TIP. 다양한 문화권 정보 이름

""(빈 문자열)	고정 문화권
af	남아공 공용어
af-ZA	아프리칸스어(남아프리카 공화국)
sq	알바니아어
sq-AL	알바니아어(알바니아)
ar	아랍어
ar-DZ	아랍어(알제리)
ar-BH	아랍어(바레인)
ar-EG	아랍어(이집트)
ar-IQ	아랍어(이라크)
ar-JO	아랍어(요르단)
ar-KW	아랍어(쿠웨이트)
ar-LB	아랍어(레바논)
ar-LY	아랍어(리비아)
ar-MA	아랍어(모로코)
ar-OM	아랍어(오만)
ar-QA	아랍어(카타르)
ar-SA	아랍어(사우디아라비아)
ar-SY	아랍어(시리아)
ar-TN	아랍어(튀니지)
ar-AE	아랍어(아랍에미리트)
ar-YE	아랍어(예멘)
hy	아르메니아어
hy-AM	아르메니아어(아르메니아)
az	아제리어
az-Cyrl-AZ	아제리어(아제르바이잔, 키릴 자모)
az-Latn-AZ	아제리어(아제르바이잔, 라틴 문자)
eu	바스크어
eu-ES	바스크어(바스크)
be	벨로루시어
be-BY	벨로루시어(벨로루시)
bg	불가리아어
bg-BG	불가리아어(불가리아)
ca	카탈로니아어
ca-ES	카탈로니아어(카탈로니아)
zh-HK	중국어(홍콩 특별 행정구, 중국)
zh-MO	중국어(마카오 특별 행정구)
zh-CN	중국어(중국)

zh-Hans	중국어(간체)
zh-SG	중국어(싱가포르)
zh-TW	중국어(대만)
zh-Hant	중국어(번체)
hr	크로아티아어
hr-BA	크로아티아어(보스니아 헤르체고비나)
hr-HR	크로아티아어(크로아티아)
cs	체코어
cs-CZ	체코어(체코)
da	덴마크어
da-DK	덴마크어(덴마크)
dv	디베히어
dv-MV	디베히어(몰디브)
nl	네덜란드어
nl-BE	네덜란드어(벨기에)
nl-NL	네덜란드어(네덜란드)
en	영어
en-AU	영어(오스트레일리아)
en-BZ	영어(벨리즈)
en-CA	영어(캐나다)
en-029	영어(카리브 해)
en-IE	영어(아일랜드)
en-JM	영어(자메이카)
en-NZ	영어(뉴질랜드)
en-PH	영어(필리핀)
en-ZA	영어(남아프리카 공화국)
en-TT	영어(트리니다드 토바고)
en-GB	영어(영국)
en-US	영어(미국)
en-ZW	영어(짐바브웨)
et	에스토니아어
et-EE	에스토니아어(에스토니아)
fo	페로스어
fo-FO	페로어(페로 제도)
fa	페르시아어
fa-IR	페르시아어(이란)
fi	핀란드어
fi-FI	핀란드어(핀란드)
fr	프랑스어
fr-BE	프랑스어(벨기에)
fr-CA	프랑스어(캐나다)

fr-FR	프랑스어(프랑스)
fr-LU	프랑스어(룩셈부르크)
fr-MC	프랑스어(모나코)
fr-CH	프랑스어(스위스)
gl	갈리시아어
gl-ES	갈리시아어(스페인)
ka	그루지야어
ka-GE	그루지야어(그루지야)
de	독일어
de-AT	독일어(오스트리아)
de-DE	독일어(독일)
de-DE_phoneb	독일어(독일, 전화 번호부 정렬)
de-LI	독일어(리히텐슈타인)
de-LU	독일어(룩셈부르크)
de-CH	독일어(스위스)
el	그리스어
el-GR	그리스어(그리스)
gu	구자라트어
gu-IN	구자라트어(인도)
he	히브리어
he-IL	히브리어(이스라엘)
hi	힌디어
hi-IN	힌디어(인도)
hu	헝가리어
hu-HU	헝가리어(헝가리)
is	아이슬란드어
is-IS	아이슬란드어(아이슬란드)
id	인도네시아어
id-ID	인도네시아어(인도네시아)
it	이탈리아어
it-IT	이탈리아어(이탈리아)
it-CH	이탈리아어(스위스)
ja	일본어
ja-JP	일본어(일본)
kn	카나다어
kn-IN	카나다어(인도)
kk	카자흐어
kk-KZ	카자흐어(카자흐스탄)
kok	콘칸어
kok-IN	콘칸어(인도)
ko	한국어

ko-KR	한국어(대한민국)
ky	키르기스어
ky-KG	키르기스어(키르기스스탄)
lv	라트비아어
lv-LV	라트비아어(라트비아)
lt	리투아니아어
lt-LT	리투아니아어(리투아니아)
mk	마케도니아어
mk-MK	마케도니아어(마케도니아, FYROM)
ms	말레이어
ms-BN	말레이어(브루나이)
ms-MY	말레이어(말레이시아)
mr	마라티어
mr-IN	마라티어(인도)
mn	몽골어
mn-MN	몽골어(몽골)
no	노르웨이어
nb-NO	노르웨이어(복말, 노르웨이)
nn-NO	노르웨이어(니노르스크, 노르웨이)
pl	폴란드어
pl-PL	폴란드어(폴란드)
pt	포르투갈어
pt-BR	포르투갈어(브라질)
pt-PT	포르투갈어(포르투갈)
pa	펀잡어
pa-IN	펀잡어(인도)
ro	루마니아어
ro-RO	루마니아어(루마니아)
ru	러시아어
ru-RU	러시아어(러시아)
sa	산스크리트어
sa-IN	산스크리트어(인도)
sr-Cyrl-CS	세르비아어(세르비아, 키릴 자모)
sr-Latn-CS	세르비아어(세르비아, 라틴 문자)
sk	슬로바키아어
sk-SK	슬로바키아어(슬로바키아)
sl	슬로베니아어
sl-SI	슬로베니아어(슬로베니아)
es	스페인어
es-AR	스페인어(아르헨티나)
es-BO	스페인어(볼리비아)

es-CL	스페인어(칠레)
es-CO	스페인어(콜롬비아)
es-CR	스페인어(코스타리카)
es-DO	스페인어(도미니카 공화국)
es-EC	스페인어(에콰도르)
es-SV	스페인어(엘살바도르)
es-GT	스페인어(과테말라)
es-HN	스페인어(온두라스)
es-MX	스페인어(멕시코)
es-NI	스페인어(니카라과)
es-PA	스페인어(파나마)
es-PY	스페인어(파라과이)
es-PE	스페인어(페루)
es-PR	스페인어(푸에르토리코)
es-ES	스페인어(스페인)
es-ES_tradnl	스페인어(스페인, 전통 정렬)
es-UY	스페인어(우루과이)
es-VE	스페인어(베네수엘라)
sw	스와힐리어
sw-KE	스와힐리어(케냐)
sv	스웨덴어
sv-FI	스웨덴어(핀란드)
sv-SE	스웨덴어(스웨덴)
syr	시리아어
syr-SY	시리아어(시리아)
ta	타밀어
ta-IN	타밀어(인도)
tt	타타르어
tt-RU	타타르어(러시아)
te	텔루구어
te-IN	텔루구어(인도)
th	태국어
th-TH	태국어(태국)
	터키어
tr-TR	터키어(터키)
uk	우크라이나어
uk-UA	우크라이나어(우크라이나)
ur	우르두어
ur-PK	우르두어(파키스탄)
uz	우즈베크어
uz-Cyrl-UZ	우즈베크어(우즈베키스탄, 키릴 자모)

uz-Latn-UZ	우즈베크어(우즈베키스탄, 라틴 문자)
vi	베트남어
vi-VN	베트남어(베트남)

http://msdn.microsoft.com/ko-kr/library/system.globalization.cultureinfo(VS.95).aspx 를 참고하세요.

그리드(Grid)

그리드 객체는 좌표 내부에 X축과 Y축의 눈금들을 연결한 라인들을 의미합니다. 그리드는 아래와 같은 속성들로 구성되어 있습니다.

그리드 속성

속성	설명
GridDirection	그리드가 그려지는 방향을 설정합니다. Both 양쪽(기본값) Horizontal 가로방향 Vertical 세로방향 None 그리드 없음
GridLine	그리드의 Line 성질을 설정합니다.
Interval	그리드의 그려지는 간격을 설정합니다. 설정하지 않을 경우 X축의 interval을 따릅니다.
ScaleBreakLine	스케일 브레이크 설정 시 그 라인의 성질을 설정합니다.
ZeroLineColor	음수 영역이 있는 그래프일 경우 제로 라인이 자동으로 그려지는데 그 색상을 설정합니다.(기본 값: red)

(표27 - 그리드 속성)

샘플 코드

C#

```
SeriesList slist = new SeriesList();
slist.GraphArea.Grid.GridDirection = GridDirection.None;
slist.GraphArea.Grid.GridLine.LineColor = Color.Green;
slist.GraphArea.Grid.Interval = 5;
slist.GraphArea.Grid.ZeroLineColor = Color.Black;
```

VB

```
Dim slist As New SeriesList()
slist.GraphArea.Grid.GridDirection = GridDirection.None
slist.GraphArea.Grid.GridLine.LineColor = Color.Green
slist.GraphArea.Grid.Interval = 5
slist.GraphArea.Grid.ZeroLineColor = Color.Black
```

5. 마커(Maker)

영어로 mark가 표시라는 의미이듯이 히포차트에서 마커란 차트에 뭔가를 표시하는 객체입니다. 라인과 면적을 이용해 표시할 수 있으며 이는 라인마커(AxisMaker)와 영역마커(AreaMaker) 두 개의 객체로 표현합니다.

라인마커(AxisMaker)

라인마커는 차트에 특정 수치에 라인을 표시해주는 객체로서 그래프 상의 중요한 값, 상한선, 하한선 등을 표현하는데 유용합니다.

(1)라인마커 속성

속성	설명
BringToFront	마커를 맨 앞으로 보낼 것인가 여부를 설정합니다. 기본값은 true입니다.
DateTimeValue	표시되는 마커의 값을 설정합니다. 이 속성은 간트차트를 그릴 경우 사용하는 날짜 형식의 값입니다.
Label	마커 라인 상단에 표시되는 문구를 설정합니다.
LegendVisible	마커를 범례에 표시할 것인가 여부를 설정합니다. 기본값은 true입니다.
Line	마커의 라인의 나타내는 기초 클래스 객체입니다. 라인마커의 굵기, 색상 등을 설정할 수 있습니다.
Name	마커의 이름을 나타내며 이 내용은 범례에 표시됩니다. ※일반적으로 Name이라는 속성은 범례에 표시되는 이름입니다.
TextFormat	마커 Label의 다양한 텍스트 포맷을 설정합니다. 수직좌표 형 차트일 경우와 수평 차트일 경우 알맞게 설정

	을 해야 하며 라벨의 좌우 위치를 수정할 수 있습니다. (다음 페이지 샘플을 통해 알아봅니다)
Value	표시되는 마커의 값을 설정합니다.

(표28 - 라인마커 속성)

(2)샘플 코드

1) 라인차트류(Vertical type)

라인, 스플라인, 컬럼 차트 등 수직 형태의 좌표를 가진 차트들은 아래와 같은 코드로 마커의 추가가 가능합니다. Name 이라는 속성 값이 범례에 표시되고 있으며 마커의 라인 색은 설정한 blue로 잘 그려졌습니다.

C#

```
AxisMarker mk = new AxisMarker("여기를넘으면안되요“, 500);
mk.Name = "상한선";
mk.BringToFront = false;
mk.Line.LineColor = Color.Blue;

slist.AxisFactor.YAxis.Markers.Add(mk);
```

VB

```
Dim mk As New AxisMarker("여기를 넘으면 안되요", 500)
mk.Name = "상한선"
mk.BringToFront = False
mk.Line.LineColor = Color.Blue

slist.AxisFactor.YAxis.Markers.Add(mk)
```

2) 바차트류(Horizontal type)

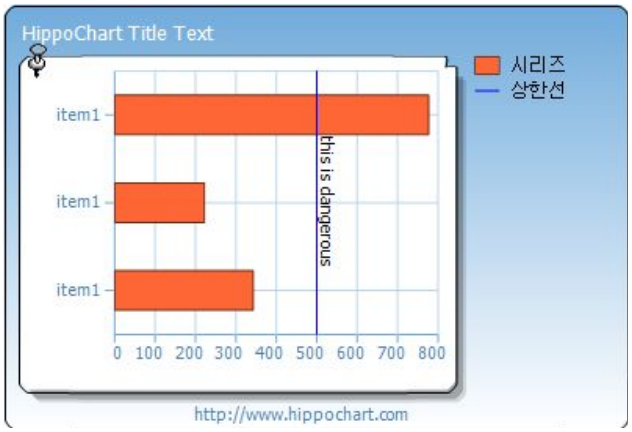
바, 간트차트처럼 수평 형태의 좌표를 가진 차트들은 마커 표현 시 몇 가지 추가 설정이 필요합니다. 아래 설정에서 보면 TextFormat을 통해 수직방향으로 설정한 것을 알 수 있습니다. 이는 한글일 경우 크게 문제가 되지 않지만 영어일 경우는 반드시 수직방향으로 설정을 해주어야합니다.

C#

```
AxisMarker mk = new AxisMarker("this is dangerous", 500);
mk.Name = "상한선";
mk.BringToFront = true;
mk.Label.ForeColor = Color.Black;
mk.Line.LineColor = Color.Blue;
mk.TextFormat.FormatFlags = StringFormatFlags.DirectionVertical; // 수직

slist.AxisFactor.YAxis.Markers.Add(mk);
```

VB

```
Dim mk As New AxisMarker("this is dangerous", 500)
mk.Name = "상한선"
mk.BringToFront = True
mk.Label.ForeColor = Color.Black
mk.Line.LineColor = Color.Blue
mk.TextFormat.FormatFlags = StringFormatFlags.DirectionVertical

slist.AxisFactor.YAxis.Markers.Add(mk)
```



전체 코드 샘플을 통해 확인하시기 바랍니다.

C#

```
SeriesList slist = new SeriesList();
slist.ChartType = ChartType.Bar;

AxisMarker mk = new AxisMarker("this is dangerous", 500);
mk.Name = "상한선";
mk.BringToFront = true;
mk.Label.ForeColor = Color.Black;
mk.Line.LineColor = Color.Blue;
mk.TextFormat.FormatFlags = StringFormatFlags.DirectionVertical;

slist.AxisFactor.YAxis.Markers.Add(mk);

Series sr = new Series();
sr.Name = "시리즈";

SeriesItem item1 = new SeriesItem(343);
SeriesItem item2 = new SeriesItem(222);
SeriesItem item3 = new SeriesItem(777);

item1.Name = "item1";
item2.Name = "item1";
item3.Name = "item1";
```

```
sr.items.Add(item1);
sr.items.Add(item2);
sr.items.Add(item3);

slist.SeriesCollection.Add(sr);

this.hHippoChart1.SeriesListDictionary.Add(slist);
this.hHippoChart1.DrawChart();
```

VB

```
Dim slist As New SeriesList()
slist.ChartType = ChartType.Bar

Dim mk As New AxisMarker("this is dangerous", 500)
mk.Name = "상한선"
mk.BringToFront = True
mk.Label.ForeColor = Color.Black
mk.Line.LineColor = Color.Blue
mk.TextFormat.FormatFlags = StringFormatFlags.DirectionVertical

slist.AxisFactor.YAxis.Markers.Add(mk)

Dim sr As New Series()
sr.Name = "시리즈"

Dim item1 As New SeriesItem(343)
Dim item2 As New SeriesItem(222)
Dim item3 As New SeriesItem(777)

item1.Name = "item1"
item2.Name = "item1"
item3.Name = "item1"

sr.items.Add(item1)
sr.items.Add(item2)
sr.items.Add(item3)

slist.SeriesCollection.Add(sr)

Me.HHippoChart1.SeriesListDictionary.Add(slist)
Me.HHippoChart1.DrawChart()
```

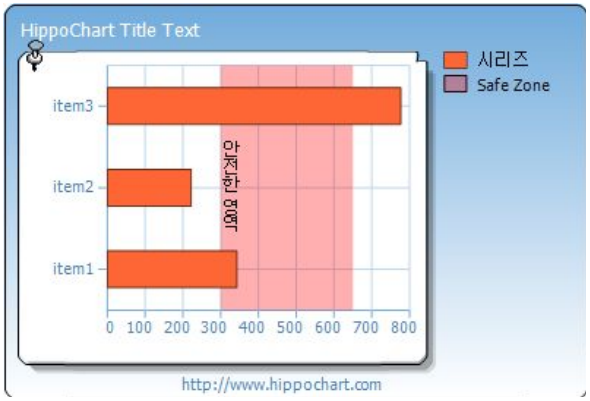
영역마커(AreaMaker)

영역마커는 중요한 기간, 수치 범위 등을 면적의 배경색으로 나타내주는 마커입니다.

(1)영역마커 속성

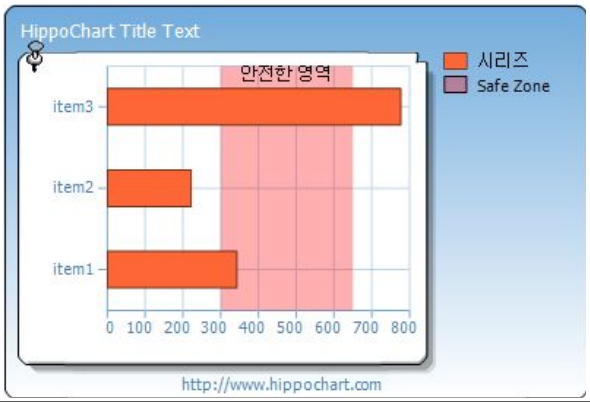
속성	설명
BackColor	시작 시점과 종료 시점을 통해 만들어진 면적의 배경색을 설정합니다.
BringToFront	영역 마커를 맨 앞으로 가져올 것인지 여부를 설정합니다. 기본값은 true입니다.
EndDateTimeValue	간트차트일 경우 종료 날짜를 설정합니다. 이 속성은 생성자를 통해 설정할 수 있습니다.
EndValue	영역마커의 종료 수치를 설정합니다. 이 속성은 생성자를 통해 설정할 수 있습니다.
IsGradation	(현재 적용되지 않습니다)
Label	영역 내부에 쓰이는 문구를 설정합니다.
LegendVisible	범례에 영역마커를 표시할 것인지 여부를 설정합니다.
Line	(현재 적용되지 않습니다.)
Name	영역 마커를 범례에 표시할 경우 그 이름을 설정합니다.
StartDateTimeValue	간트차트일 경우 시작 날짜를 설정합니다. 이 속성은 생성자를 통해 설정할 수 있습니다.
StartValue	영역마커의 시작 수치를 설정합니다. 이 속성은 생성자를 통해 설정할 수 있습니다.

TextFormat	영역 내부에 그려지는 문구의 상하, 좌우 위치와 글자의 수직, 수평을 설정합니다. 예를 들어
	
	area.TextFormat.FormatFlags = StringFormatFlags.DirectionVertical;
	area.TextFormat.LineAlignment = StringAlignment.Near;
	
	로 설정할 경우 아래와 같이 왼쪽으로 정렬되고



area.TextFormat.FormatFlags = StringFormatFlags.DisplayFormatControl;
area.TextFormat.LineAlignment = StringAlignment.Near;

로 설정할 경우 아래와 같이 위로 정렬합니다.



(표29 - 영역마커 속성)

(2)샘플 코드

C#

```
AxisArea area = new AxisArea("안전한영역", 300, 650);
area.BackColor = Color.FromArgb(80, Color.Red);
area.Name = "Safe Zone";
area.TextFormat.FormatFlags = StringFormatFlags.DisplayFormatControl;
area.TextFormat.LineAlignment = StringAlignment.Near;

slist.AxisFactor.YAxis.Areas.Add(area);
```

VB

```
Dim area As New AxisArea("안전한 영역", 300, 650)
area.BackColor = Color.FromArgb(80, Color.Red)
area.Name = "Safe Zone"
area.TextFormat.FormatFlags = StringFormatFlags.DisplayFormatControl
area.TextFormat.LineAlignment = StringAlignment.Near

slist.AxisFactor.YAxis.Areas.Add(area)
```

6. 틱(Tick)

차트에서 Tick이란 “눈금”을 의미합니다. 눈금은 x축 혹은 y축과 추가된 축들의 아이템에 붙는 작은 라인을 통해 그려지는데 이는 Tick과 Extra Tick 두 종류로 분류할 수 있습니다. Tick은 자동 혹은 수동으로 설정한 축의 각 단계별로 생기는 눈금을 의미하고, Extra Tick은 사용자 임의로 생성한 눈금을 의미합니다.

1) Tick 구성하기

Tick의 구성을 통해 각 단계별 눈금과 라벨을 디자인할 수 있습니다. 예를 들어 x축에 한국, 미국, 일본, 중국 이라는 아이템 중에 중요한 내용은 한국의 데이터이므로 “한국” 이라는 글자를 크게 표시한다거나 각 나라별 아이콘 아미지를 사용하여 눈금을 대치한다거나 하는 기능을 구현할 수 있습니다.

X축 Tick 구성하기

아래 코드는 “item1” 이라는 x축 눈금을 “이름”을 통해서 찾아 폰트를 조절하고 있습니다. 이처럼 tick은 존재하는 아이템의 눈금을 설정하는 기능입니다.

C#

```
AxisTick tk = new AxisTick("item1");
tk.Label.Font = new Font("굴림, 13, FontStyle.Bold);

slist.AxisFactor.XAxis.Ticks.Add(tk);
```

VB

```
Dim tk As New AxisTick("item1")
tk.Label.Font = New Font("굴림", 13, FontStyle.Bold)

slist.AxisFactor.XAxis.Ticks.Add(tk)
```

Y축 Tick 구성하기

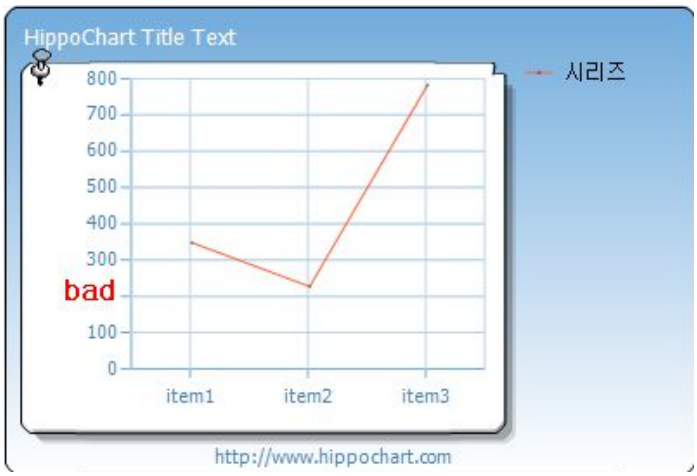
아래 코드는 Y축의 단계를 “200”이라는 문자열로 알아내어 “bad"라는 문자로 재설정하고 있습니다. Tick에서는 Y축이라 하더라도 글자 자체를 비교하기 때문에 ”200“과 같이 문자열로 파라미터를 넘겨야합니다.

C#

```
AxisTick tk2 = new AxisTick("200");
tk2.Label.Text = "bad";
tk2.Label.Font = new Font("굴림, 13, FontStyle.Bold);

slist.AxisFactor.YAxis.Ticks.Add(tk2);
```

VB

```
Dim tk2 As New AxisTick("200")
tk2.Label.Text = "bad"
tk2.Label.Font = New Font("굴림", 13, FontStyle.Bold)

slist.AxisFactor.YAxis.Ticks.Add(tk2)
```



2) Extra Tick 구성하기

Extra Tick은 사용자가 임의로 특정 위치에 눈금을 삽입할 수 있는 객체로써 대표 눈금 사이에 작은 눈금들을 삽입하거나 중요한 x축 경계선을 표시할 수 있는 등 다양한 효과를 내는데 유용합니다.

X축 Extra Tick 구성하기

x축의 Extra Tick은 x축의 datatype이 number 형일 경우만 적용할 수 있습니다. 축이 숫자 형일 경우만 정확한 위치를 지정할 수가 있기 때문입니다. 또한 AxisTick의 생성자의 파라미터로 2.5와 같은 정확한 double형의 숫자를 넘겨야

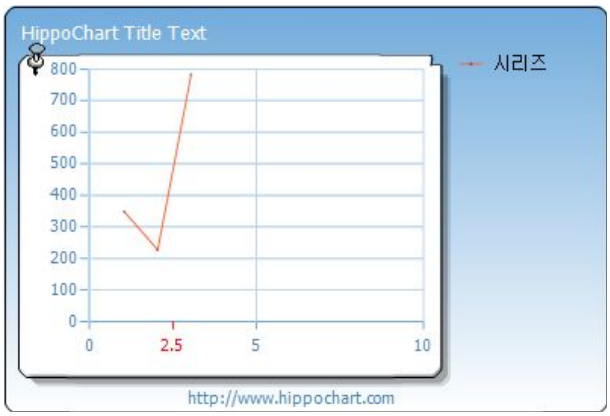
합니다.

C#

```
AxisTick tk3 = new AxisTick(2.5);
tk3.Label.Text = "2.5";
slist.AxisFactor.XAxis.ExtraTicks.Add(tk3);
```

VB

```
Dim tk3 As New AxisTick(2.5)
tk3.Label.Text = "2.5"
slist.AxisFactor.XAxis.ExtraTicks.Add(tk3)
```



Y축 Extra Tick 구성하기

Y축도 X축과 마찬가지로 사용자 지정 눈금을 지정할 수 있습니다. Tick의 경우는 아이템의 이름을 통해 접근하지만 Extra Tick은 아이템이 없으므로 정확한 수치를 넘겨야 눈금의 위치를 계산하여 표시할 수 있습니다.

아래 코드는 1000과 2000 의 단계 사이에 200 차이의 작은 눈금 4개를 삽입한 예제입니다. 눈금과 글자의 색상의 기본 값은 모두 red이므로 눈금의 색상을 하늘색 계열로 수정하였고, 눈금의 그리드로도 표현하기 위해 IsShowGridLine 라는 속성을 사용하고 있습니다.

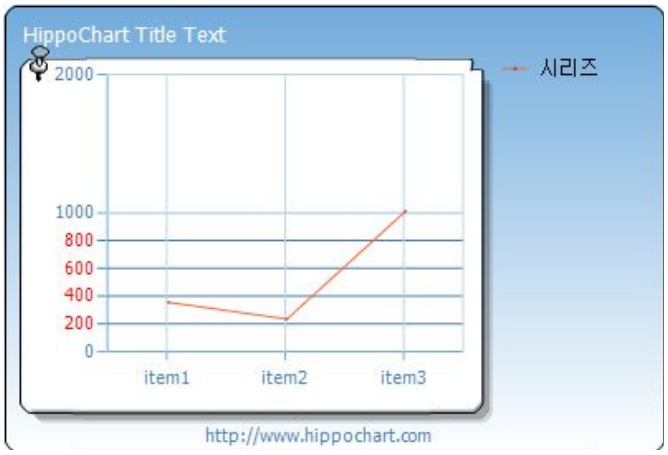
C#

```
AxisTick ex1 = new AxisTick(200);
AxisTick ex2 = new AxisTick(400);
AxisTick ex3 = new AxisTick(600);
AxisTick ex4 = new AxisTick(800);

ex1.GridLine.LineColor = Color.SteelBlue;
ex2.GridLine.LineColor = Color.SteelBlue;
ex3.GridLine.LineColor = Color.SteelBlue;
ex4.GridLine.LineColor = Color.SteelBlue;

ex1.IsShowGridLine = true;
ex2.IsShowGridLine = true;
ex3.IsShowGridLine = true;
ex4.IsShowGridLine = true;

ex1.Label.Text = "200";
ex2.Label.Text = "400";
ex3.Label.Text = "600";
ex4.Label.Text = "800";

slist.AxisFactor.YAxis.ExtraTicks.Add(ex1);
slist.AxisFactor.YAxis.ExtraTicks.Add(ex2);
slist.AxisFactor.YAxis.ExtraTicks.Add(ex3);
slist.AxisFactor.YAxis.ExtraTicks.Add(ex4);
```

VB

```
Dim ex1 As New AxisTick(200)
Dim ex2 As New AxisTick(400)
Dim ex3 As New AxisTick(600)
Dim ex4 As New AxisTick(800)

ex1.GridLine.LineColor = Color.SteelBlue
ex2.GridLine.LineColor = Color.SteelBlue
ex3.GridLine.LineColor = Color.SteelBlue
ex4.GridLine.LineColor = Color.SteelBlue

ex1.IsShowGridLine = True
```

```
ex2.IsShowGridLine = True
ex3.IsShowGridLine = True
ex4.IsShowGridLine = True

ex1.Label.Text = "200"
ex2.Label.Text = "400"
ex3.Label.Text = "600"
ex4.Label.Text = "800"

slist.AxisFactor.YAxis.ExtraTicks.Add(ex1)
slist.AxisFactor.YAxis.ExtraTicks.Add(ex2)
slist.AxisFactor.YAxis.ExtraTicks.Add(ex3)
slist.AxisFactor.YAxis.ExtraTicks.Add(ex4)
```



7. 통계아이템

히포차트는 AxisCalculator 라는 클래스를 통해 데이터를 분석하고 축을 구성하는 요소들을 생성해 냅니다. 이 클래스의 또 다른 역할은 데이터 분석을 통해 각종 통계 수치들을 계산하는데 그 항목은 아래와 같습니다. 통계 수치는 한 시리즈리스트 전체 데이터를 대상으로 계산됩니다.

통계 아이템	설명
Average	시리즈리스트 전체 데이터의 평균을 의미합니다.

Max	시리즈리스트 전체 데이터의 최대값을 의미합니다.
Median	시리즈리스트 전체 데이터의 중앙값을 의미합니다.
Min	시리즈리스트 전체 데이터의 최소값을 의미합니다.
StandardDeviation	시리즈리스트 전체 데이터의 표준편차을 의미합니다.
Sum	시리즈리스트 전체 데이터의 총합을 의미합니다.
Variance	시리즈리스트 전체 데이터의 분산을 의미합니다.

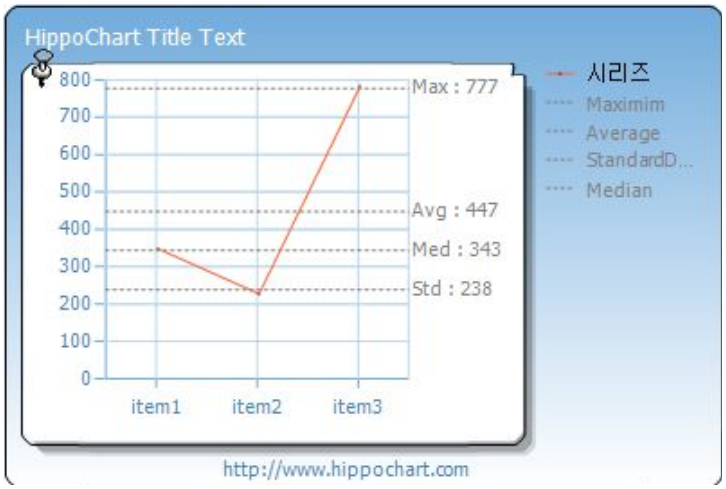
TIP.

중앙값 – 통계집단(統計集團)의 변량을 크기의 순서로 늘어 놓았을 때, 중앙에 위치하는 값으로 메디안이라고도 하는데, 총수 n이 홀수일 때는 (n+1)/2번째의 변량, n이 짝수일 때는 n/2번째와 (n+2)/2번째의 변량의 산술평균을 취한다. (네이버백과사전)

분산 – 각 변량에서 통계값을 뺀 값의 제곱의 합을 총 개수로 나눈 수

표준편차 – 분산의 제곱근. 표준 편차가 클수록 평균에서 떨어진 값이 많이 존재한다는 의미

위 통계 아이템 중에 Sum과 Variance는 보통 축의 최대 단위값 보다 큰 값이 산출되기 때문에 표시를 하더라도 축에 표시되지 않는 경우가 있습니다. 이럴 경우는 좌표축을 수동으로 처리하여 보이게 할 수 있습니다.



통계 아이템은 AxisCalculator를 통해 이미 계산이 되어 있기 때문에 아래와 같이 Visible을 true로 처리함으로써 차트에 표시할 수 있습니다.

통계 아이템은 각 축의 AnalysisItems라는 컬렉션에서 관리를 하고 있는데 이

는 AnalysisCategory라는 통계 아이템을 나타내는 열거형을 첫 번째 파라미터로 하고 있는 제네릭 컬렉션으로 이루어져 있어 인덱스 방식이 아닌 열거형을 사용해서 해당 아이템을 직관적으로 접근할 수 있습니다.

C#

```
slist.AxisFactor.YAxis.AnalysisItems[AnalysisCategory.Average].Visible = true;
slist.AxisFactor.YAxis.AnalysisItems[AnalysisCategory.Max].Visible = true;
slist.AxisFactor.YAxis.AnalysisItems[AnalysisCategory.Median].Visible = true;
slist.AxisFactor.YAxis.AnalysisItems[AnalysisCategory.Min].Visible = true;
slist.AxisFactor.YAxis.AnalysisItems[AnalysisCategory.StandardDeviation].Visible =
true;
slist.AxisFactor.YAxis.AnalysisItems[AnalysisCategory.Sum].Visible = true;
slist.AxisFactor.YAxis.AnalysisItems[AnalysisCategory.Variance].Visible = true;
```

VB

```
slist.AxisFactor.YAxis.AnalysisItems(AnalysisCategory.Average).Visible = True
slist.AxisFactor.YAxis.AnalysisItems(AnalysisCategory.Max).Visible = True
slist.AxisFactor.YAxis.AnalysisItems(AnalysisCategory.Median).Visible = True
slist.AxisFactor.YAxis.AnalysisItems(AnalysisCategory.Min).Visible = True
slist.AxisFactor.YAxis.AnalysisItems(AnalysisCategory.StandardDeviation).Visible = True
slist.AxisFactor.YAxis.AnalysisItems(AnalysisCategory.Sum).Visible = True
slist.AxisFactor.YAxis.AnalysisItems(AnalysisCategory.Variance).Visible = True
```

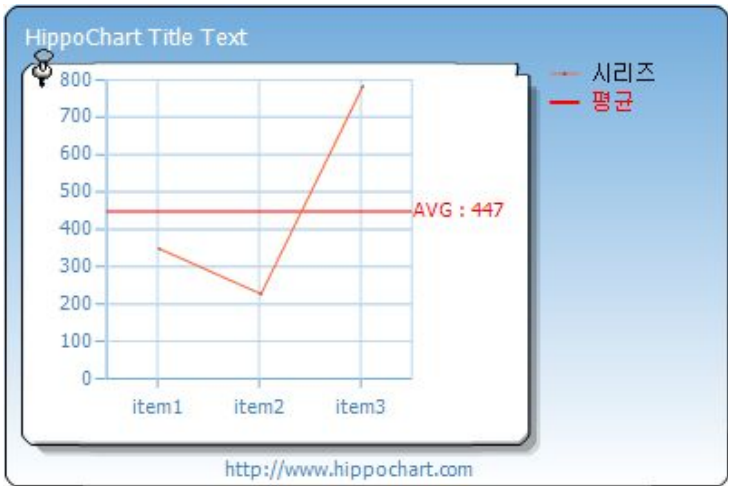
통계 아이템은 아래와 같은 코드로 다양하게 디자인을 수정할 수도 있습니다.

C#

```
slist.AxisFactor.YAxis.AnalysisItems[AnalysisCategory.Average].Visible = true;
slist.AxisFactor.YAxis.AnalysisItems[AnalysisCategory.Average].Label.Text = "AVG";
slist.AxisFactor.YAxis.AnalysisItems[AnalysisCategory.Average].Label.ForeColor =
Color.Red;
slist.AxisFactor.YAxis.AnalysisItems[AnalysisCategory.Average].Name = "평균";
slist.AxisFactor.YAxis.AnalysisItems[AnalysisCategory.Average].Line.LineWidth = 2;
slist.AxisFactor.YAxis.AnalysisItems[AnalysisCategory.Average].Line.LineColor =
Color.Red;
slist.AxisFactor.YAxis.AnalysisItems[AnalysisCategory.Average].Line.DashStyle =
System.Drawing.Drawing2D.DashStyle.Solid;
```

VB

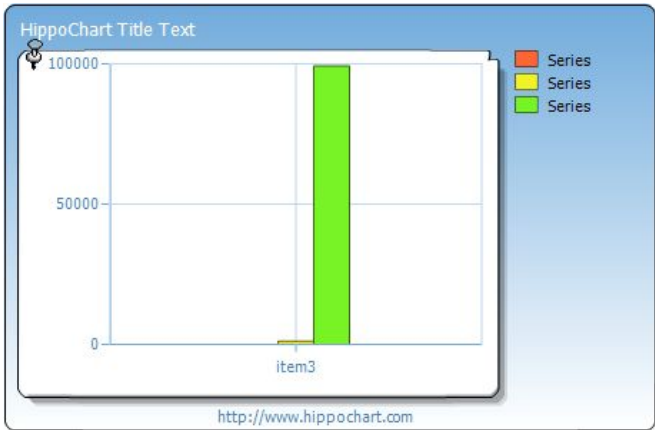
```
slist.AxisFactor.YAxis.AnalysisItems(AnalysisCategory.Average).Visible = True
slist.AxisFactor.YAxis.AnalysisItems(AnalysisCategory.Average).Label.Text = "AVG"
slist.AxisFactor.YAxis.AnalysisItems(AnalysisCategory.Average).Label.ForeColor = Color.Red
slist.AxisFactor.YAxis.AnalysisItems(AnalysisCategory.Average).Name = "평균"
slist.AxisFactor.YAxis.AnalysisItems(AnalysisCategory.Average).Line.LineWidth = 2
slist.AxisFactor.YAxis.AnalysisItems(AnalysisCategory.Average).Line.LineColor = Color.Red
slist.AxisFactor.YAxis.AnalysisItems(AnalysisCategory.Average).Line.DashStyle =
System.Drawing.Drawing2D.DashStyle.Solid
```



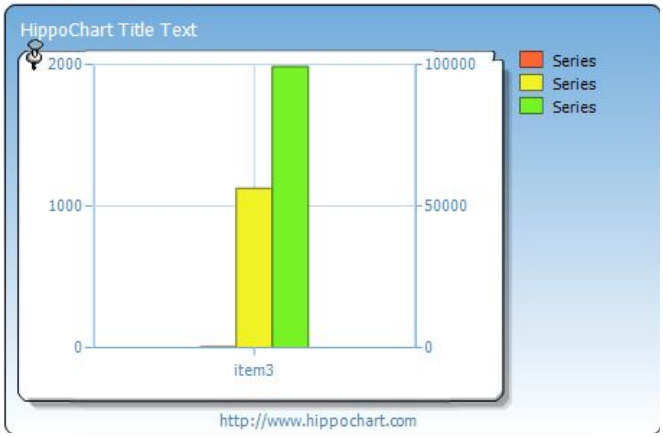
8. 멀티 축

멀티 축은 이중 축이라고도 하며 시리즈 별로 다른 축으로 그래프를 그리는 기능입니다. 다음 항목에서 설명하겠지만 스케일 브레이크가 같은 시리즈 내에서의 시리즈아이템들의 큰 데이터 격차를 해결하는 방안이라면 멀티 축은 한 시리즈 리스트에 포함되어 있는 시리즈들 간의 큰 데이터 격차 문제를 해소할 수 있는 방안이라고 할 수 있습니다.

3개의 시리즈에 각각 한 개의 데이터가 있고 그 값들이 3, 1123, 99203이라고 정의합니다. 이를 차트로 그려보면 아래와 같습니다.

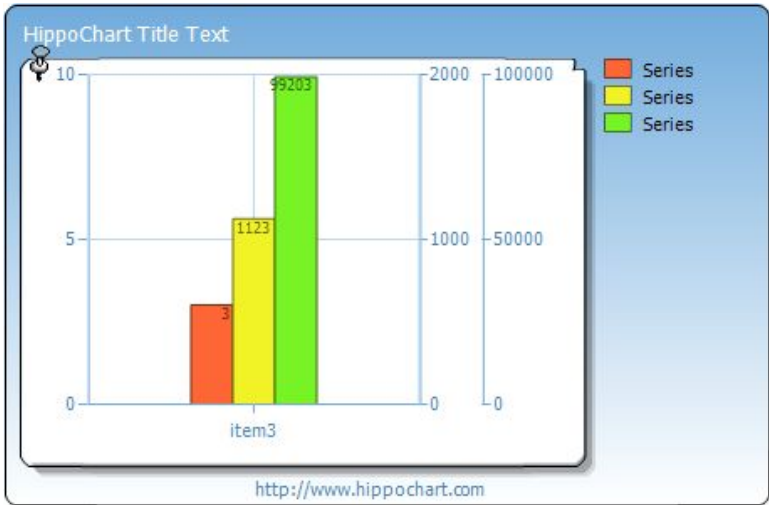


마지막 시리즈의 데이터가 10만에 가까운 수라서 3은 보이지도 않고 1123도 아주 작은 수처럼 느껴집니다. 이럴 경우 마지막 시리즈를 별도의 축으로 그린다면 아래와 같이 조금 원활한 모양을 보여줄 것입니다.



하지만 역시 3이라는 수는 1123에 비해서도 여전히 작은 수이기 때문에 잘 보이지 않습니다. 이럴 경우는 3개의 축으로 표시하면 이 문제를 해결할 수 있겠습니다.

이처럼, 멀티 축을 이용하면 각 시리즈별로 축을 할당하여 별도로 AxisCalculator를 호출하여 단계 수치 및 통계 아이템을 계산할 수 있어 데이터 차이가 크거나 데이터의 성질이 다른(퍼센트 혹은 통화 등) 경우도 원활하게 처리할 수 있습니다.



멀티 축 구현하기

추가된 축 역시 Axis를 구현하기 때문에 기본 축(YAxis)와 동일한 모든 기능을 수행합니다.

C#

```
Axis ax1 = new Axis(AxisType.Ytype); ....(1)

ax1.AnalysisItems[AnalysisCategory.Average].Visible = true ;
ax1.Markers.Add(new AxisMarker("새마커", 34));
ax1.SetAxisStep(0, 3000, 500);

slist.AxisFactor.AxisItems.Add(ax1); ....(2)
```

VB

```
Dim ax1 As New Axis(AxisType.Ytype)

ax1.AnalysisItems(AnalysisCategory.Average).Visible = True
ax1.Markers.Add(New AxisMarker("새 마커", 34))
ax1.SetAxisStep(0, 3000, 500)

slist.AxisFactor.AxisItems.Add(ax1)
```

멀티 축을 구현하기 위해서 가장 먼저 해야 할 일은 새로운 축을 구성하는 일입니다. 위의 코드는 추가될 축을 구성하는 코드인데 통계 아이템과 마커를 하나 추가했고, 수동으로 설정을 하고 있습니다. 물론 일반적인 예는 특별한 설정 없이 (1)과 (2) 부분만 설정해주면 나머지는 모두 자동으로 처리됩니다.

Axis 클래스로 사용자 객체를 생성하고 AxisItems라는 추가 축을 관리하는 컬렉션에 추가를 해줍니다. 다음은 해당 시리즈에서 “어떤 축을 사용 하겠다”라는 명시가 있어야겠지요??

```
Series sr1 = new Series();
Series sr2 = new Series();
Series sr3 = new Series();

sr2.AxisIndex = 1;
sr3.AxisIndex = 2;
```

보통 첫 번째 시리즈는 기본 축을 사용할 것이므로 설정을 하지 않습니다. (설정을 하지 않으면 자동으로 0으로 설정됩니다.) 코드 상에 sr2 라는 시리즈는 축 인덱스(AxisIndex) 가 1로 설정이 되어 있습니다. 이는 추가된 축들의 인덱스가 아니고 기본축을 포함한 총 축 중에 1번 인덱스 축을 사용하겠다는 의미입니다. 아래 표를 참고하세요.

축	축 인덱스
YAxis	0
추가된 축 1	1
추가된 축 2	2

추가된 축은 기본 축이 그려지는 반대쪽 방향에 계속 추가가 되는 형태이고 인덱스가 작은(먼저 추가된) 축일수록 좌표 영역에 가깝게 그려집니다.

샘플 코드

마지막에 그렸던 3개의 시리즈 모두 다른 축으로 그리는 예제입니다.

C#

```
SeriesList slist = new SeriesList();
slist.ChartType = ChartType.Column;

Axis ax1 = new Axis(AxisType.Ytype);
slist.AxisFactor.AxisItems.Add(ax1);

Axis ax2 = new Axis(AxisType.Ytype);
slist.AxisFactor.AxisItems.Add(ax2);

Series sr1 = new Series();
Series sr2 = new Series();
Series sr3 = new Series();

sr2.AxisIndex = 1;
sr3.AxisIndex = 2;

SeriesItem item1 = new SeriesItem(3);
SeriesItem item2 = new SeriesItem(1123);
SeriesItem item3 = new SeriesItem(99203);

item1.Name = "item1";
item2.Name = "item1";
item3.Name = "item1";

sr1.items.Add(item1);
sr2.items.Add(item2);
sr3.items.Add(item3);

slist.SeriesCollection.Add(sr1);
slist.SeriesCollection.Add(sr2);
slist.SeriesCollection.Add(sr3);

this.hHippoChart1.SeriesListDictionary.Add(slist);
this.hHippoChart1.DrawChart();
```

VB

```
Dim slist As New SeriesList()
slist.ChartType = ChartType.Column
```

```
Dim ax1 As New Axis(AxisType.Ytype)
slist.AxisFactor.AxisItems.Add(ax1)

Dim ax2 As New Axis(AxisType.Ytype)
slist.AxisFactor.AxisItems.Add(ax2)

Dim sr1 As New Series()
Dim sr2 As New Series()
Dim sr3 As New Series()

sr2.AxisIndex = 1
sr3.AxisIndex = 2

Dim item1 As New SeriesItem(3)
Dim item2 As New SeriesItem(1123)
Dim item3 As New SeriesItem(99203)

item1.Name = "item1"
item2.Name = "item2"
item3.Name = "item3"

item1.XValue = 1
item2.XValue = 2
item3.XValue = 3

sr1.items.Add(item1)
sr2.items.Add(item2)
sr3.items.Add(item3)

slist.SeriesCollection.Add(sr1)
slist.SeriesCollection.Add(sr2)
slist.SeriesCollection.Add(sr3)

Me.hHippoChart1.SeriesListDictionary.Add(slist)
Me.hHippoChart1.DrawChart()
```

9. 스케일 브레이크(Scale Break)

멀티 축이 여러 시리즈들의 데이터 격차의 해소 방안이었다면 스케일 브레이크

는 한 시리즈 내의 데이터 간의 데이터 격차 문제를 해소하는 방안입니다.

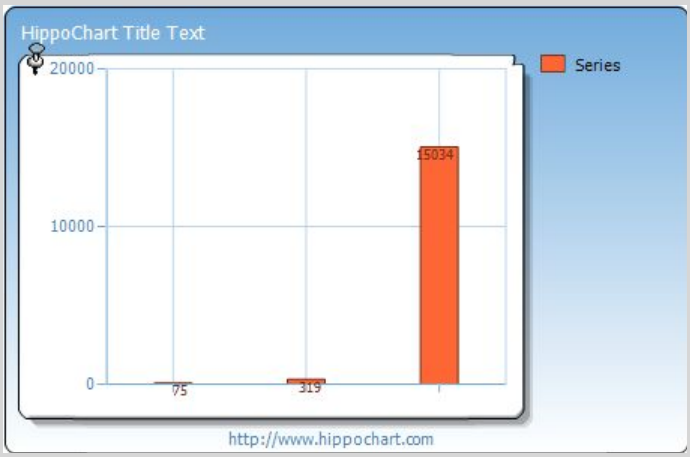
스케일 브레이크는 말 그대로 축의 크기를 자른다는 의미입니다. 즉, 사용자 임의로 필요 없는 영역의 스케일을 잘라내야 하므로 수동으로 처리해야하고, 잘못 설정할 경우 좌표 축 단계에 문제가 생길 수 있으니 주의해야할 기능입니다.

복잡한 내용이므로 우선 간단한 예시를 통해 알아보겠습니다.

김 군이 그려야할 데이터는 아래와 같습니다.

75 319 14,034

이 데이터는 보통 500 이하의 값만 있지만 간혹 14,000이 넘는 데이터가 생긴다고 합니다. 결국 500 이상부터 14,000 사이 스케일은 필요가 없는 샘입니다. 우선, 위 데이터를 차트로 한 번 그려봅니다.



역시 우려했던 대로 중요한 데이터인 500 이하의 값들을 구분할 수가 없었습니다.

그래서 김 군은 히포차트의 스케일 브레이크 기능을 이용하기 위해 데이터를 분석해보았습니다.

1) 수동 처리:

- 최소 단위 값: 음수가 없으므로 0
- **인터벌**: 위 데이터는 크게 두 부류로 나눌 수 있는데 그 중 작은 데이터 군의 최대 값이 319이므로 319보다 작고 단계를 나눌 수 있는 명확한 값을 설정하면 됩니다. 여기서는 150, 100이 가능하겠습니다. 100으로 설정.
- 최대 단위 값: 인터벌이 100이고 데이터의 최대값이 14,034 이므로 14,100

2) 스케일 브레이크 영역 설정:

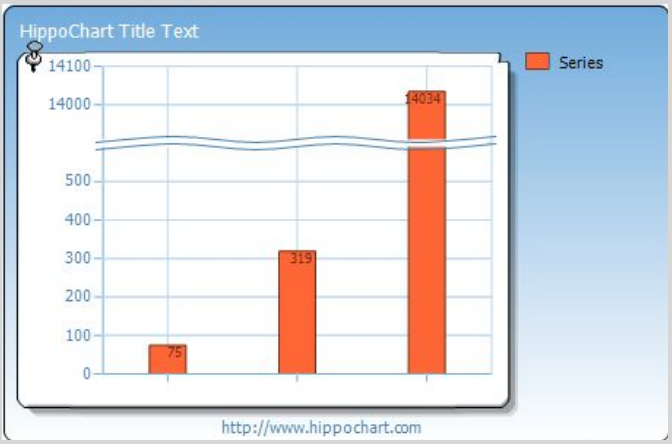
아래 공식에 따라

R1: 500

R2: 14,000

a = 500 + 100 = 600

b = 14000 - 100 = 13900



TIP.

※ 스케일 브레이크 영역 구하기 공식

R1: 없어져야할 공간 시작 값

R2: 없어져야할 공간 종료 값 (단, R1 < R2)

a(스케일 브레이크 설정 시작 값) = R1 + Interval
b(스케일 브레이크 설정 종료 값) = R2 - Interval
(단, Interval > 0)

∴sList.AxisFactor.YAxis.AxisScaleRanges.Add(new AxisScaleRange(a, b));

이상 공식에 의해서 간단히 설정해보았습니다. 중요한 것은 어디를 잘라야할 것인지를 정하고 작은 데이터 군에서도 단계가 잘 나누어지도록 인터벌을 구하는 것이 핵심입니다. 위에서는 319라는 값이 있으므로 인터벌은 그보다 작아야할 것이고(작아야 한다는 것은 절대적인 내용은 아닙니다) 그 값은 큰 데이터 군에서도 단계가 잘 적용이 되어야할 것입니다.

쉽게 이해하기 위해 차트가 세로로 엄청 길다고 상상하고 그린다고 한다면 100씩 끊어서 600부터 13900까지 잘라내야 한다는 결론은 쉽게 나올 수 있을 것으로 생각됩니다.

그리고 위에 제시한 공식은 제가 일반화 시켜 본 것으로 반드시 사용해야하는 공식은 아닙니다. 몇 번 설정해보면 응용이 되어 데이터를 보고 짐작으로 설정이 가능 합니다.

아래 전체 코드를 통해 정리하세요.

C#

```
SeriesList sList = new SeriesList();

sList.SeriesCollection.Add(new Series());
sList.ChartType = ChartType.Column;

Random rr1 = new Random();

for (int i = 0; i < 3; i++)
{
    SeriesItem item = new SeriesItem();
    item.IsShowFigureText = true;
    sList.SeriesCollection[0].items.Add(item);
}
```

```
sList.SeriesCollection[0].items[0].YValue = 75;
sList.SeriesCollection[0].items[1].YValue = 319;
sList.SeriesCollection[0].items[2].YValue = 14034;

sList.AxisFactor.YAxis.AxisScaleRanges.Add(new AxisScaleRange(600, 13900));

sList.AxisFactor.YAxis.IsAutoSetting = false;
sList.AxisFactor.YAxis.MaxUnitValue = 14100;
sList.AxisFactor.YAxis.MinUnitValue = 0;
sList.AxisFactor.YAxis.Interval = 100; // 키포인트

this.hHippoChart1.SeriesListDictionary.Add(sList);
this.hHippoChart1.DrawChart();
```

VB

```
Dim sList As New SeriesList()

sList.SeriesCollection.Add(New Series())
sList.ChartType = ChartType.Column

Dim rr1 As New Random()

For i As Integer = 0 To 2
    Dim item As New SeriesItem()
    item.IsShowFigureText = True
    sList.SeriesCollection(0).items.Add(item)
Next

sList.SeriesCollection(0).items(0).YValue = 75
sList.SeriesCollection(0).items(1).YValue = 319
sList.SeriesCollection(0).items(2).YValue = 14034

sList.AxisFactor.YAxis.AxisScaleRanges.Add(New AxisScaleRange(600, 13900))

sList.AxisFactor.YAxis.IsAutoSetting = False
sList.AxisFactor.YAxis.MaxUnitValue = 14100
sList.AxisFactor.YAxis.MinUnitValue = 0
sList.AxisFactor.YAxis.Interval = 100
```

```
' 키포인트
Me.hHippoChart1.SeriesListDictionary.Add(sList)
Me.hHippoChart1.DrawChart()
```

TIP.

멀티 축과 스케일 브레이크는 동시에 적용되지 않습니다.

10. 범례(Legend)

히포차트에서 범례란 차트에 표시되는 모든 객체들에 대해 아이콘과 함께 간단한 이름을 명시해 주는 객체를 말합니다.

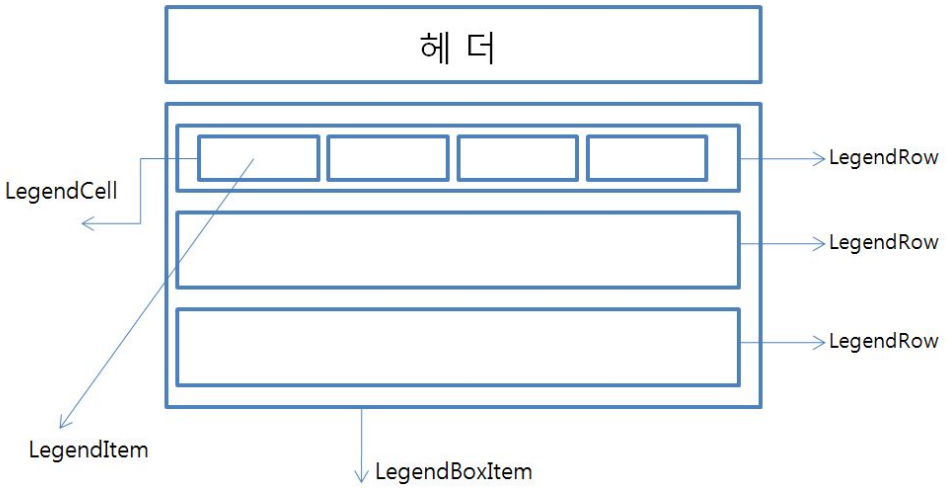
범례에 표시되는 객체

① 좌표 차트일 경우 시리즈별 그래프 종류
② 비 좌표 차트일 경우 각 시리즈아이템
③ 라인마커
④ 영역마커
⑤ 통계아이템

(※ Tick은 범례에 표시되지 않습니다.)

범례의 구성

범례는 크게 헤더와 LegendItem로 이우러져 있으며 LegendItem은 LegendRow와 LegendCell로 세부 구성되어 있습니다. 그리고 헤더는 Value영역, Percent영역, Icon영역, Name영역으로 구분되어 있습니다. 아래 그림을 통해 이해하시기 바랍니다.



범례 속성

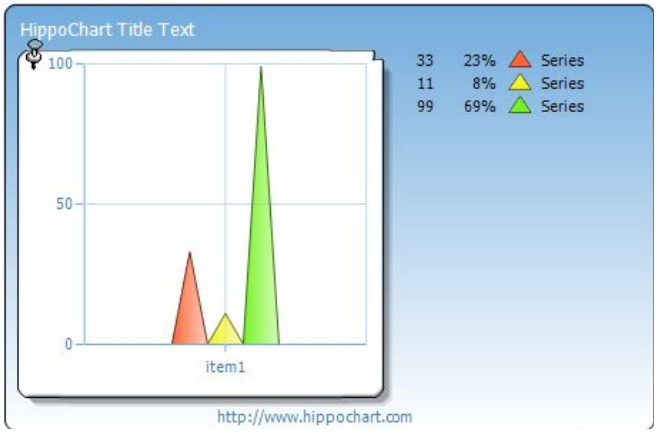
속성	설명
DivideLineColor	가로 구분선의 라인 색상을 설정합니다. 기본은 회색 (현재 사용되지 않습니다.)
FigureFormat	Value 부분의 수치 포맷을 설정합니다.
Header	범례 헤더를 나타내는 속성입니다. 사용자 설정 사항이 아닙니다.
Height	범례 영역의 높이를 나타냅니다. 사용자 설정 사항이 아닙니다.
IsAutoSetting	범례를 수동으로 등록할 것인지 여부를 설정합니다. 수동으로 구성할 경우 false로 설정합니다.
IsDivideLine	가로 구분선을 표시할 것인지 여부를 설정합니다. (현재 사용되지 않습니다)
IsShowHeader	범례의 헤더를 보이게 설정할 것인지 여부를 설정합니다.
IsVerticalLine	범례의 수직 구분라인을 표시할 것인지 여부를 설정합니다.
LegendBoxItems	범례를 구성하는 LegendBoxItem을 관리하는 컬렉션 객체입니다. 범례를 수동으로 등록할 경우만 설정합니다.

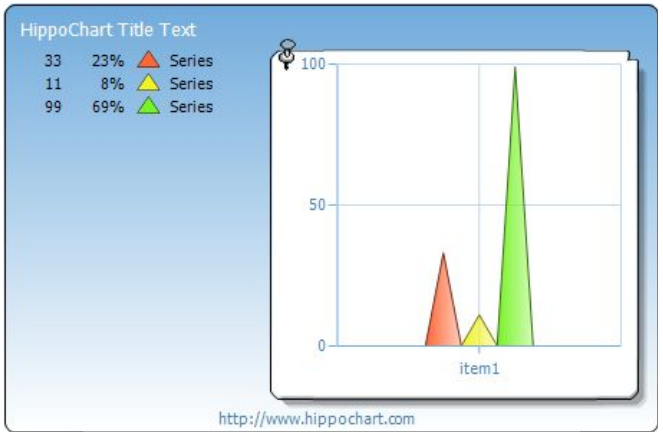
LegendFormat	범례의 4가지 포맷을 설정합니다. Icon_Name 아이콘 + 이름 Value_Icon_Name 합(값) + 아이콘 + 이름 Value_Percent_Icon_Name 합(값) + 퍼센트값 + 아이콘 + 이름 Percent_Icon_Name 퍼센트값 + 아이콘 + 이름
Location	범례의 4가지 위치를 설정합니다. Default 기본방향 범례(오른쪽) Left 왼쪽 Top 위 방향 범례 Bottom 아래방향 범례
PercentDecimalPoint	범례의 퍼센트 부분에 사용되는 수치의 소수 자리수를 설정합니다.
ValueDecimalPoint	범례의 Value 부분에 사용되는 수치의 소수 자리수를 설정합니다.
ValueType	범례의 Value 부분에는 좌표 차트인 경우는 각 시리즈 별로 합 또는 평균을 표시해주는데 이 경우 표시되는 항목을 선택할 수 있습니다. (단, 비 좌표 차트일 경우는 범례에 각 항목별로 표시가 되므로 시리즈 아이템의 값이 Value 값에 표시됩니다.) Sum 합계 Average 평균
VerticalLineColor	수직 구분선의 색상을 설정합니다. (현재 사용하지 않습니다.)
Visible	범례를 보일 것인지 여부를 설정합니다.
Width	범례 영역의 가로 길이입니다. 사용자 설정 사항이 아닙니다.

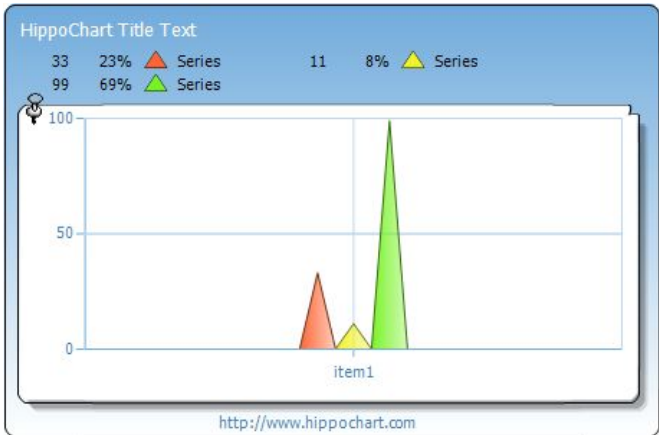
(표30 - 범례 속성)

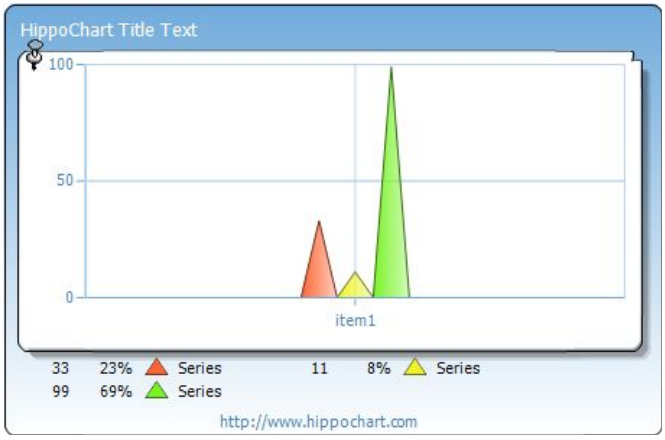
범례의 위치

범례는 Location이라는 속성을 이용해 총 4가지 방향으로 변경이 가능합니다.


HippoChart Title Text
100
50
0
item1
33 23% Series
11 8% Series
99 69% Series
http://www.hippochart.com

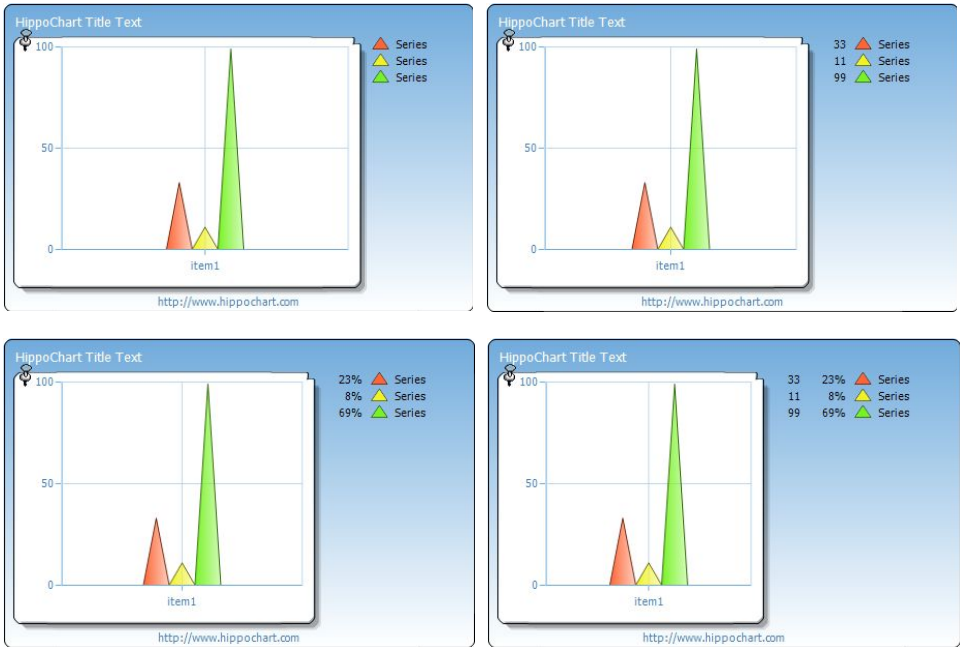

HippoChart Title Text
33 23% Series
11 8% Series
99 69% Series
100
50
0
item1
http://www.hippochart.com


HippoChart Title Text
33 23% Series
11 8% Series
99 69% Series
100
50
0
item1
http://www.hippochart.com



범례 타입

범례의 타입은 LegendFormat의 속성에 의해 결정되며 아래 4가지 형태를 제공합니다.



범례 기타 설정

1) 범례 헤더와 넓이 조정

히포차트 카페를 통해 가장 많은 질문을 받은 부분이 아닐까 합니다. 바로 범례의 넓이를 어떻게 조절하는가 하는 질문인데, 이는 전적으로 헤더에 달려있습니다. 범례의 넓이는 기본적으로 설정되어 있는 헤더의 텍스트 넓이에 따라 결정되며 변경하고자 할 경우 범례의 4가지 항목의 텍스트를 각각 변경해주어야 합니다.

범례의 항목들 중에 가장 긴 것을 찾아서 그 넓이를 범례의 넓이로 하지 않은 이유는 너무 길어질 가능성과 긴 내용은 다 보여줄 필요가 없을 경우를 위해서이고, 픽셀로 설정을 하지 않는 이유는 사용자가 픽셀보다 MeasureString을 보다 직관적으로 이해할 수 있을 것 같아서입니다.

넓이에서 전체 넓이는 수정을 할 수가 없고 각 셀 단위로 설정하여 전체 넓이를 만듭니다. 셀은 앞서 범례 구조 그림에서 보았듯이 값, 퍼센트, 아이콘, 이름의 총 4개로 구성되어 있고 각각의 헤더 텍스트는 아래와 같이 Header 클래스를 통해 접근합니다.

C#

```
this.hHippoChart1.LegendBox.Header.ValueLabel.Text = "값 헤더";
this.hHippoChart1.LegendBox.Header.PercentLabel.Text = "퍼센트 헤더";
this.hHippoChart1.LegendBox.Header.IconLabel.Text = "아이콘 헤더";
this.hHippoChart1.LegendBox.Header.NameLabel.Text = "이름 헤더";
```

VB

```
Me.hHippoChart1.LegendBox.Header.ValueLabel.Text = "값 헤더"
Me.hHippoChart1.LegendBox.Header.PercentLabel.Text = "퍼센트 헤더"
Me.hHippoChart1.LegendBox.Header.IconLabel.Text = "아이콘 헤더"
Me.hHippoChart1.LegendBox.Header.NameLabel.Text = "이름 헤더"
```

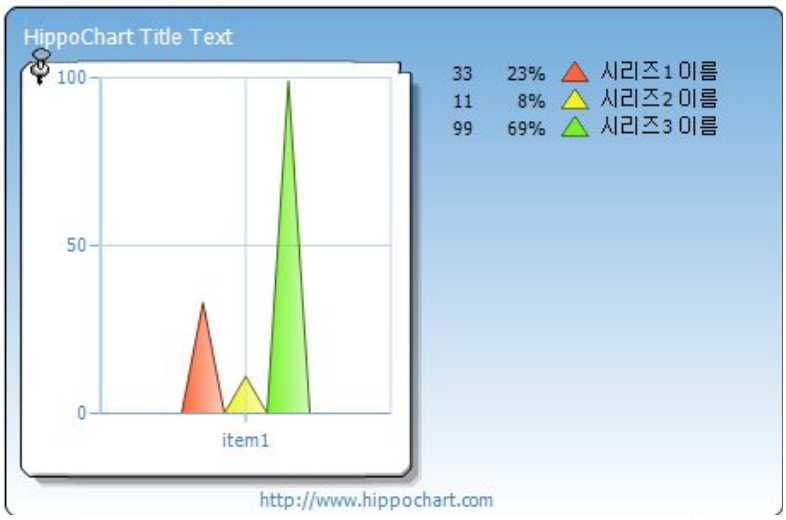
헤더의 길이에 따라 각 항목의 넓이가 좌우되지만 이는 헤더를 보이지 않게 설정했을 경우도 동일하게 적용되므로 이를 이용하면 굳이 헤더를 보이지 않게해도 원활하게 범례 텍스트를 조정할 수 있겠습니다.

또한, 범례에는 2가지 제한 사항이 있습니다.

- 범례의 총 길이는 차트 전체 길이의 1/2를 넘지 못한다.
- 범례는 Row 당 한 줄로만 표시된다.

범례를 너무 길게 설정해버리면 차트 공간이 사라지게 되므로 1/2정도로 제한이 있고 마찬가지로 세로 공간의 확보를 위해 한 줄의 제한이 있습니다.

전체 코드를 보면서 정리하시기 바랍니다.



C#

```
SeriesList slist = new SeriesList();
slist.ChartType = ChartType.Cone;

Series sr1 = new Series();
sr1.Name = "시리즈1 이름";

Series sr2 = new Series();
sr2.Name = "시리즈2 이름";

Series sr3 = new Series();
sr3.Name = "시리즈3 이름";
```

```
SeriesItem item1 = new SeriesItem(33);
SeriesItem item2 = new SeriesItem(11);
SeriesItem item3 = new SeriesItem(99);

item3.Name = "item1";

item1.XValue = 1;
item2.XValue = 2;
item3.XValue = 3;

sr1.items.Add(item1);
sr2.items.Add(item2);
sr3.items.Add(item3);

slist.SeriesCollection.Add(sr1);
slist.SeriesCollection.Add(sr2);
slist.SeriesCollection.Add(sr3);

this.hHippoChart1.LegendBox.LegendFormat = LegendFormat.Value_Percent_Icon_Name;

this.hHippoChart1.LegendBox.Header.NameLabel.Text = "우헤헤헤헤헤헤"; // 범례를 안보이게 했으므로 이런 식으로 넣어도 글자 길이만 맞으면 상관없다는 의미입니다


this.hHippoChart1.SeriesListDictionary.Add(slist);
this.hHippoChart1.DrawChart();
```

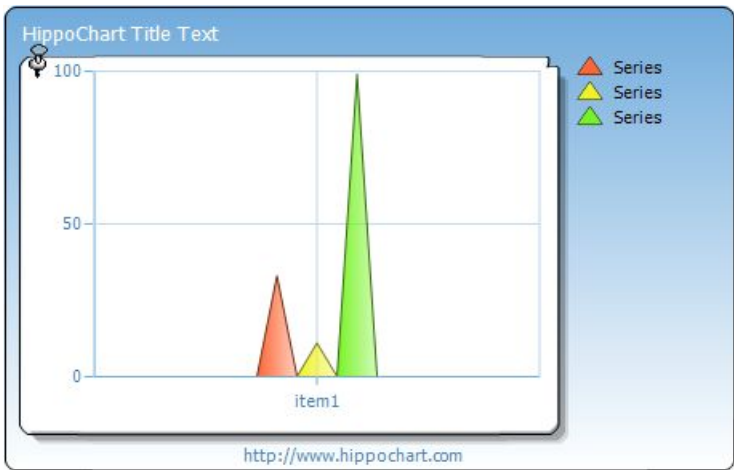
VB

```
Dim slist As New SeriesList()
slist.ChartType = ChartType.Cone

Dim sr1 As New Series()
sr1.Name = "시리즈1 이름"

Dim sr2 As New Series()
sr2.Name = "시리즈2 이름"

Dim sr3 As New Series()
sr3.Name = "시리즈3 이름"
```

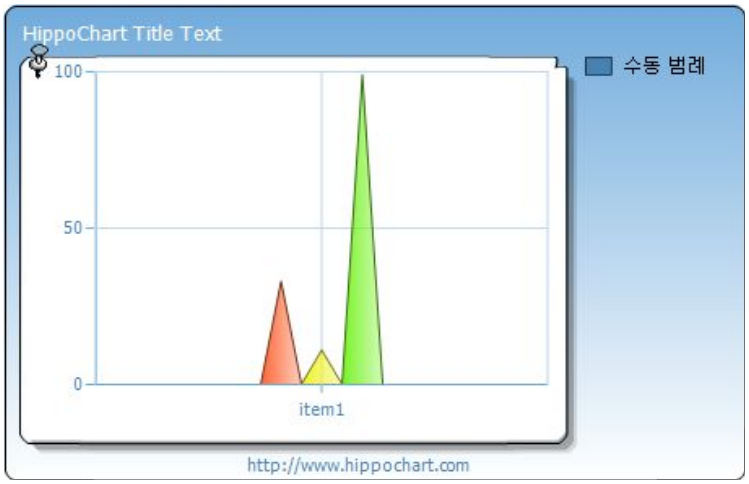
```
Dim item1 As New SeriesItem(33)
Dim item2 As New SeriesItem(11)
Dim item3 As New SeriesItem(99)

item3.Name = "item1"

item1.XValue = 1
item2.XValue = 2
item3.XValue = 3

sr1.items.Add(item1)
sr2.items.Add(item2)
sr3.items.Add(item3)

slist.SeriesCollection.Add(sr1)
slist.SeriesCollection.Add(sr2)
slist.SeriesCollection.Add(sr3)

Me.hHippoChart1.LegendBox.LegendFormat = LegendFormat.Value_Percent_Icon_Name

Me.hHippoChart1.LegendBox.Header.NameLabel.Text = "우헤헤헤헤헤헤"


Me.hHippoChart1.SeriesListDictionary.Add(slist)
Me.hHippoChart1.DrawChart()
```

2) 범례 수동으로 설정하기

히포차트에는 기본적으로 차트 종류와 각종 마커 등을 이용해서 그래프에 표시되는 내용들을 범례에 자동으로 추가를 하지만 사용자가 임의로 내용을 수정할 수 있게 열어놓고 있습니다.



아래는 수동으로 설정한 차트입니다. 범례를 수동으로 설정한 후 사용자가 하나 하나 추가해주는 개념입니다. 코드를 통해 자세히 알아보겠습니다.



크게 초기화 단계와 추가 단계로 나눌 수 있습니다.

초기화 하기

가장 먼저 해야할 일은 자동으로 설정되는 범례를 수동 처리하고 범례에 추가되어 있는 LegendBoxItem들을 모두 제거해야합니다. 또한, 시리즈리스트에 추가되어 있는 이너 레전드의 내용에도 동일한 방식으로 적용해 주어야합니다.

C#

```
this.hHippoChart1.LegendBox.IsAutoSetting = false;
this.hHippoChart1.LegendBox.LegendBoxItems.Clear();

slist.Legend.IsAutoSetting = false;
slist.Legend.LegendBoxItems.Clear();
```

VB

```
Me.hHippoChart1.LegendBox.IsAutoSetting = False
Me.hHippoChart1.LegendBox.LegendBoxItems.Clear()

slist.Legend.IsAutoSetting = False
slist.Legend.LegendBoxItems.Clear()
```

추가 하기

우선 전체 아이템을 관리할 LegendBoxItem을 생성하고 필요한 항목만큼 LegendRow를 구성하여 추가합니다.

LegendRow는 범례형식을 생성자 파라미터로 가지고 있고 LegendBoxType을 통해 모양을 IconColor를 통해 색상을 조정할 수 있습니다. 모두 추가를 한 LegendBoxItem를 이너 레전드와 외부 범례 객체의 LegendBoxItems 컬렉션에 각각 추가를 해주면 되겠습니다.

C#

```
LegendBoxItem box = new LegendBoxItem();

LegendRow row = new LegendRow(LegendFormat.Icon_Name);
row.Item.LegendBoxType = ChartType.Column;
row.Item.IconColor = Color.SteelBlue;
row.Item.Label.Text = "수동 범례";

box.Rows.Add(row);

slist.Legend.LegendBoxItems.Add(box);
this.hHippoChart1.LegendBox.LegendBoxItems.Add(box);
```

VB

```
Dim box As New LegendBoxItem()

Dim row As New LegendRow(LegendFormat.Icon_Name)
row.Item.LegendBoxType = ChartType.Column
row.Item.IconColor = Color.SteelBlue
row.Item.Label.Text = "수동 범례"

box.Rows.Add(row)

slist.Legend.LegendBoxItems.Add(box)
Me.hHippoChart1.LegendBox.LegendBoxItems.Add(box)
```

범례의 수동 처리는 상당히 복잡한 로직이 들어가므로 상당히 주의를 요합니다. 또한, 멀티 시리즈리스트인 경우는 올바르게 동작을 하지 않습니다.

전체 코드를 통해 복습하기 바랍니다.

C#

```
SeriesList slist = new SeriesList();
slist.ChartType = ChartType.Cone;

Series sr1 = new Series();
Series sr2 = new Series();
Series sr3 = new Series();

SeriesItem item1 = new SeriesItem(33);
SeriesItem item2 = new SeriesItem(11);
SeriesItem item3 = new SeriesItem(99);

item3.Name = "item1";

item1.XValue = 1;
item2.XValue = 2;
item3.XValue = 3;

sr1.items.Add(item1);
sr2.items.Add(item2);
sr3.items.Add(item3);
```

```
slist.SeriesCollection.Add(sr1);
slist.SeriesCollection.Add(sr2);
slist.SeriesCollection.Add(sr3);

// 범례 수동
this.hHippoChart1.LegendBox.IsAutoSetting = false;
this.hHippoChart1.LegendBox.LegendBoxItems.Clear();

slist.Legend.IsAutoSetting = false;
slist.Legend.LegendBoxItems.Clear();

LegendBoxItem box = new LegendBoxItem();

LegendRow row = new LegendRow(LegendFormat.Icon_Name);
row.Item.LegendBoxType = ChartType.Column;
row.Item.IconColor = Color.SteelBlue;
row.Item.Label.Text = "수동 범례";

box.Rows.Add(row);

slist.Legend.LegendBoxItems.Add(box);
this.hHippoChart1.LegendBox.LegendBoxItems.Add(box);


this.hHippoChart1.SeriesListDictionary.Add(slist);
this.hHippoChart1.DrawChart();
```

VB

```
Dim slist As New SeriesList()
slist.ChartType = ChartType.Cone

Dim sr1 As New Series()
Dim sr2 As New Series()
Dim sr3 As New Series()

Dim item1 As New SeriesItem(33)
Dim item2 As New SeriesItem(11)
Dim item3 As New SeriesItem(99)
```

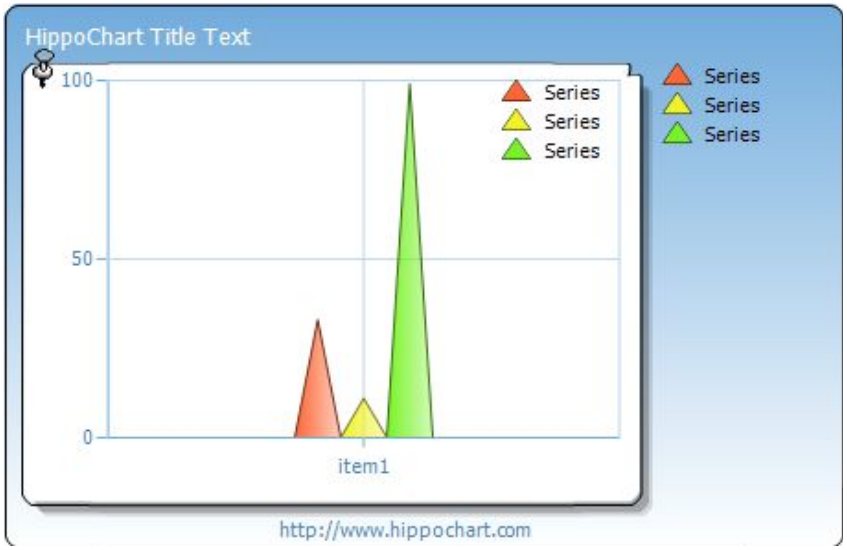
```
item3.Name = "item1"

item1.XValue = 1
item2.XValue = 2
item3.XValue = 3

sr1.items.Add(item1)
sr2.items.Add(item2)
sr3.items.Add(item3)

slist.SeriesCollection.Add(sr1)
slist.SeriesCollection.Add(sr2)
slist.SeriesCollection.Add(sr3)

' 범례 수동
Me.hHippoChart1.LegendBox.IsAutoSetting = False
Me.hHippoChart1.LegendBox.LegendBoxItems.Clear()

slist.Legend.IsAutoSetting = False
slist.Legend.LegendBoxItems.Clear()

Dim box As New LegendBoxItem()

Dim row As New LegendRow(LegendFormat.Icon_Name)
row.Item.LegendBoxType = ChartType.Column
row.Item.IconColor = Color.SteelBlue
row.Item.Label.Text = "수동 범례"

box.Rows.Add(row)

slist.Legend.LegendBoxItems.Add(box)
Me.hHippoChart1.LegendBox.LegendBoxItems.Add(box)


Me.hHippoChart1.SeriesListDictionary.Add(slist)
Me.hHippoChart1.DrawChart()
```

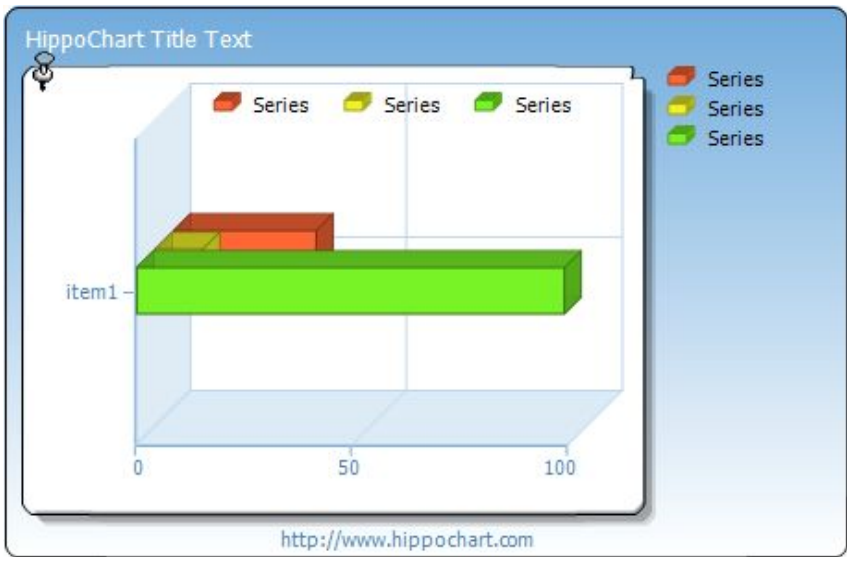
3) 이너 레전드(내부 범례)

이너 레전드란 그래프 내부에 범례가 들어가는 기능을 말합니다. 외부 범례는 그래프 영역의 바깥쪽에서 그래프 영역의 넓이를 조절하며 별도로 동작하지만

이너 레전드는 각 시리즈리스트 별로 범례를 보여줄 수 있어 다양한 효과를 줄 수 있습니다. 또한 공간의 활용면에서 더욱 유용합니다.

이너 레전드와 외부 범례는 별도로 동작하므로 이너 레전드를 사용하려면 외부 범례의 Visible 속성을 false로 처리하는 것이 좋습니다. 물론 두 군데 모두 표시해도 문제는 없습니다.





적용 방법은 간단합니다. 아래와 같이 Visible 속성만 변경해주면 반영이 됩니

다.

slist.Legend.Visible = true;

> **TIP. 외부 범례와 이너 레전드 와의 관계**
>
> 내부 범례는 외부 범례와 별도로 동작을 하지만 일부 속성을 통해 외부 범례에 종속되어 있기도 하다.
>
> IsAutiSetting
> LegendFormat
>
> 이 두 가지인데, 수동으로 설정할 경우 외부 범례에서 반드시 수동 처리를 해주어야 하며 범례의 포맷을 설정할 경우는 외부, 내부 모두 동일하게 설정해야한다. 이는 외부 범례를 보이지 않게 해도 적용된다.

전체 코드를 통해 정리하시기 바랍니다.

C#

```
SeriesList slist = new SeriesList();
slist.ChartType = ChartType.Cone;

//이너 레전드
slist.Legend.Visible = true;
slist.Legend.Header.NameLabel.Text = "Series";
slist.Legend.LegendFormat = LegendFormat.Value_Percent_Icon_Name;
slist.Legend.Location = LegendLocation.Top;

Series sr1 = new Series();
Series sr2 = new Series();
Series sr3 = new Series();

SeriesItem item1 = new SeriesItem(33);
SeriesItem item2 = new SeriesItem(11);
SeriesItem item3 = new SeriesItem(99);

item3.Name = "item1";

item1.XValue = 1;
item2.XValue = 2;
```

```
item3.XValue = 3;

sr1.items.Add(item1);
sr2.items.Add(item2);
sr3.items.Add(item3);

slist.SeriesCollection.Add(sr1);
slist.SeriesCollection.Add(sr2);
slist.SeriesCollection.Add(sr3);

this.hHippoChart1.LegendBox.Visible = false;
this.hHippoChart1.LegendBox.LegendFormat = LegendFormat.Value_Percent_Icon_Name;

this.hHippoChart1.SeriesListDictionary.Add(slist);
this.hHippoChart1.DrawChart();
```

VB

```
Dim slist As New SeriesList()
slist.ChartType = ChartType.Cone

'이너 레전드
slist.Legend.Visible = True
slist.Legend.Header.NameLabel.Text = "Series"
slist.Legend.LegendFormat = LegendFormat.Value_Percent_Icon_Name
slist.Legend.Location = LegendLocation.Top

Dim sr1 As New Series()
Dim sr2 As New Series()
Dim sr3 As New Series()

Dim item1 As New SeriesItem(33)
Dim item2 As New SeriesItem(11)
Dim item3 As New SeriesItem(99)

item3.Name = "item1"

item1.XValue = 1
item2.XValue = 2
item3.XValue = 3
```

```
sr1.items.Add(item1)
sr2.items.Add(item2)
sr3.items.Add(item3)

slist.SeriesCollection.Add(sr1)
slist.SeriesCollection.Add(sr2)
slist.SeriesCollection.Add(sr3)

Me.hHippoChart1.LegendBox.Visible = False
Me.hHippoChart1.LegendBox.LegendFormat = LegendFormat.Value_Percent_Icon_Name

Me.hHippoChart1.SeriesListDictionary.Add(slist)
Me.hHippoChart1.DrawChart()
```

11. 풍선도움말(balloon)

풍선 도움말은 시리즈아이템에 간단한 코멘트를 달 수 있는 객체입니다. 특정 시리즈아이템에 부가설명을 달거나 중요한 수치일 경우 또는 중요한 항목일 경우 등 다양한 활용이 가능하며 디자인 요소 또한 다양하여 특별한 효과를 줄 수 있습니다.

풍선도움말 속성

속성	설명
BackColor	풍선도움말에 글자가 쓰이는 공간의 배경색을 설정합니다.
BalloonType	풍선의 타입을 설정합니다. Default 기본형(아이보리색의 둥근형) Rectangle 사각형
Height	풍선 도움말 사각형의 세로 길이입니다. (사용자 설정 기능이 아닙니다.)
HeightType	풍선 도움말이 표시되는 상대적 세로 위치를 설정합니다. 표시해야하는 시리즈아이템의 값이 최대 단위 값과 유사해서 가려지는 문제 등을 해결합니다.

	Top 위 Center 가운데 Bottom 아래
IsShadow	풍선 도움말의 주변 그림자 효과를 설정합니다. 기본 값은 true입니다.
Label	풍선 도움말의 글자를 나타내는 객체입니다. 실제 쓰이는 문구를 설정합니다.
Line	풍선 도움말의 배경 라인을 설정합니다. 배경 라인색상 변경, 배경 라인을 없애거나 점선으로 표시하는 등의 효과를 줄 수 있습니다.
TextLocation	StringAlignment 타입의 속성으로 풍선 도움말 영역 내부에서 글자의 정렬을 설정합니다. Near 왼쪽으로 정렬합니다. Center 가운데로 정렬합니다. Far 오른쪽으로 설정합니다. (단, 방향은 상대적입니다.)
Width	풍선 도움말 영역의 가로 넓이입니다. (사용자 설정 기능이 아닙니다.)
X	풍선 도움말의 Left 위치를 의미합니다. (사용자 설정 기능이 아닙니다.)
Y	풍선 도움말의 Top 위치를 의미합니다. (사용자 설정 기능이 아닙니다.)

(표31 - 풍선도움말 속성)

샘플 코드

아래는 Balloon을 구현하는 예제입니다. 앞에서 잠깐 살펴보았듯이 시리즈아이템의 내부 속성들은 거의 null인 상태이기 때문에 반드시 인스턴스를 생성해야한다고 했습니다. 풍선도움말 객체 역시 그러하므로 아래 코드에서 인스턴스를 생성한 후 각 속성들을 설정하고 있습니다.

C#

```
SeriesItem item1 = new SeriesItem(33);

item1.Balloon = new Balloon();
item1.Balloon.Label.Text = "빨강색";
item1.Balloon.IsShadow = false;
item1.Balloon.Line.LineColor = Color.Red;
```

```
item1.Balloon.TextLocation = StringAlignment.Far;
```

VB

```
Dim item1 As New SeriesItem(33)

item1.Balloon = New Balloon()
item1.Balloon.Label.Text = "빨강색"
item1.Balloon.IsShadow = False
item1.Balloon.Line.LineColor = Color.Red
item1.Balloon.TextLocation = StringAlignment.Far
```

아래가 결과 이미지 차트입니다. 배경 라인을 빨간색으로 처리하였고, “빨강색” 이라는 문구를 삽입 하였습니다. 그리고 주변 그림자 효과는 제거하였고 텍스트를 오른쪽 정렬 하였습니다.

쉬운 내용이므로 전체 코드를 통해 간단히 정리하시기 바랍니다.

C#

```
SeriesList slist = new SeriesList();
slist.ChartType = ChartType.Circular;

Series sr1 = new Series();
Series sr2 = new Series();
```

```
Series sr3 = new Series();

SeriesItem item1 = new SeriesItem(33);

item1.Balloon = new Balloon();
item1.Balloon.Label.Text = "빨강색";
item1.Balloon.IsShadow = false;
item1.Balloon.Line.LineColor = Color.Red;
item1.Balloon.TextLocation = StringAlignment.Far;

SeriesItem item2 = new SeriesItem(11);
SeriesItem item3 = new SeriesItem(55);

item3.Name = "item1";

item1.XValue = 1;
item2.XValue = 2;
item3.XValue = 3;

sr1.items.Add(item1);
sr2.items.Add(item2);
sr3.items.Add(item3);

slist.SeriesCollection.Add(sr1);
slist.SeriesCollection.Add(sr2);
slist.SeriesCollection.Add(sr3);

this.hHippoChart1.SeriesListDictionary.Add(slist);
this.hHippoChart1.DrawChart();
```

VB

```
Dim slist As New SeriesList()
slist.ChartType = ChartType.Circular

Dim sr1 As New Series()
Dim sr2 As New Series()
Dim sr3 As New Series()

Dim item1 As New SeriesItem(33)
```

```
item1.Balloon = New Balloon()
item1.Balloon.Label.Text = "빨강색"
item1.Balloon.IsShadow = False
item1.Balloon.Line.LineColor = Color.Red
item1.Balloon.TextLocation = StringAlignment.Far

Dim item2 As New SeriesItem(11)
Dim item3 As New SeriesItem(55)

item3.Name = "item1"

item1.XValue = 1
item2.XValue = 2
item3.XValue = 3

sr1.items.Add(item1)
sr2.items.Add(item2)
sr3.items.Add(item3)

slist.SeriesCollection.Add(sr1)
slist.SeriesCollection.Add(sr2)
slist.SeriesCollection.Add(sr3)

Me.hHippoChart1.SeriesListDictionary.Add(slist)
Me.hHippoChart1.DrawChart()
```

12. 차트 초기화

차트 개발을 하다 보면 현재 그리고 있던 상태 및 데이터를 초기화하고 다시 처음부터 그리거나 다른 데이터 군을 그려야할 경우가 있습니다. 히포차트에는 다양한 차트 초기화 도구들이 있는데 그 적절한 사용법과 예제를 알아보겠습니다.

전체 초기화

차트의 전체 초기화는 전체 시리즈리스트를 제거하고 각종 설정된 모든 속성들을 초기화 하는 작업입니다. 히포차트에는 이를 위해 ResetAll() 이라는 메소드를 제공하고 있는데 이 메소드를 통해 차트 전체를 초기화할 수 있습니다.

```
this.hHippoChart1.ResetAll();
```

단, 여기서 주의해야할 점은 이 메소드를 계속해서 반복 호출 할 경우 차트의 깜빡임 현상이 있을 수 있다는 점입니다. ResetAll 메소드는 타이머, For 문, 반복 호출되는 사용자 지정 메소드 등에서의 사용은 권장하지 않습니다.

그럼 그럴 경우는 어떻게 할까?

그럴 경우는 직접적으로 히포차트 객체에서 시리즈리스트를 제거합니다.

```
this.hHippoChart1.SeriesListDictionary.Clear();
```

히포차트는 시리즈 기반으로 차트에 대한 데이터와 모든 설정들을 하고 있으므로 시리즈리스트 딕셔너리를 비워버리면 데이터와 모든 설정이 초기화 됩니다. 다만 타이틀, 로고, 범례(외부) 등 히포차트 객체에 있는 객체들의 내용은 남아있습니다.

이를 응용하면 다양한 초기화가 가능한데 시리즈리스트의 설정을 유지하고 데이터 및 각 차트별 속성을 초기화해야 할 경우는 아래와 같은 코드로 처리할 수 있습니다.

```
this.hHippoChart1.SeriesListDictionary[0].SeriesCollection.Clear();
```

위 코드는 첫 번째 시리즈리스트의 모든 “시리즈”를 제거하는 코드입니다. 시리즈란 한 개의 그래프가 가져야할 속성들과 데이터(시리즈아이템)을 가지고 있으므로 이를 제거함으로써 새로운 설정을 가능하게 합니다.

데이터 초기화

히포차트에서 데이터란 “시리즈아이템”과 동일 시 할 수 있습니다. 데이터의 초기화란 결국 시리즈아이템을 모두 제거한다는 의미입니다. 이는 위 시리즈 계층구조를 이용해 아래와 같이 제거할 수 있습니다.

```
this.hHippoChart1.SeriesListDictionary[0].SeriesCollection[0].items.Clear();
```

하지만 이렇게 코드를 작성할 경우 모든 데이터를 초기화해야할 경우 상당히 긴 반복문을 사용해야하므로 히포차트에서는 간단한 ResetData() 라는 메소드를 지원하고 있습니다.

```
this.hHippoChart1.ResetData();
```

TIP.

차트 초기화는 정확하게 계획을 하지 않는다면 의도하지 않은 부분을 삭제할 수 도 있는 등 문제가 발생할 수 있다. 그래서 필자가 실제 코드를 작성하면서 가장 많이 사용하는 초기화는 시리즈 리스트를 초기화하는 아래의 코드다.

```
this.hHippoChart1.SeriesListDictionary.Clear();
```

매번 호출해야하는 메소드의 구성 시 사용하는 코드인데 가장 성능에 영향을 미치지 않고 간단하게 모든 설정을 제거하고 재구성할 수 있기 때문에 사용자 지정 메소드 구성 시 코드 상단에 배치하여 사용하고 있다.