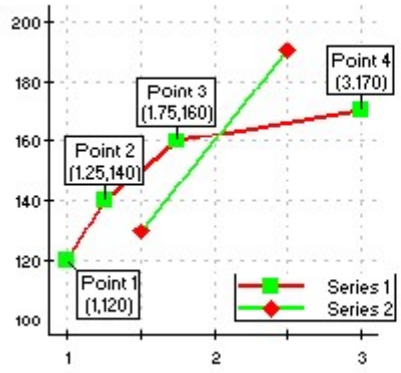
绘制数据

2D 图表显示一般布局提供的数据。数据被装载在 ChartDataArray
对象中，这些对象都有一个类型（type）对象。数据可以用提前装载的数组，也可以在设计时填充数据。
每个图表类型都有一种通用的布局。布局包括一个 X 数组，一个 Y 数组，和 Y1，Y2，Y3， Y4
数组。所有这些数组都可以装载数据，也可以为空。 例如画出一个 XY 图表只需要一个数组系列，只要 X 和 Y 数组有值就可以了，其他的 Y 数组可以为空。另外，不像一组数据的图表，每个数据系列可以有不同数量的点，并且不需要一定和 X 值匹配，这样就能做出下面的图。



General Data
2 Series, Maximum of 4 Points

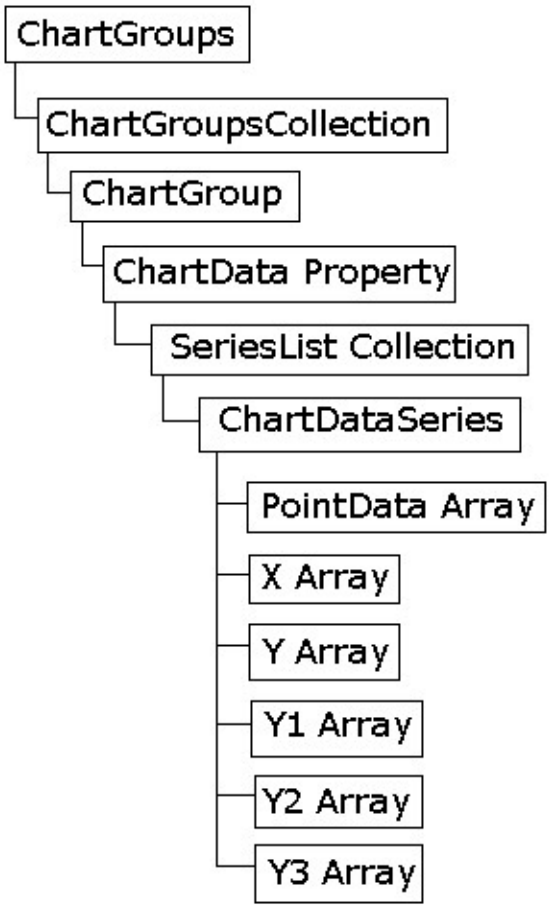
4	Series 1 Num. Points			
1	1.25	1.75	3	Series 1 X-value
120	140	160	170	Series 1 Y-value
2	Series 2 Num. Points			
1.5	2.5			Series 2 X-value
130	190			Series 2 Y-value
Point 1	Point 2	Point 3	Point 4	

为方便起见,一个 PointData 属性(PointF 的一个数组)也可以被用来提供 X 和 Y 的数据。
PointF 的值并不是独立的 X 和 Y 的数组,仅仅是另一种形式的数据输入和输出。PointData 属性值的变化会更改 X 和 Y 的数组,反之亦然。
在作图之前，一个重要的总体设计必须有下面的特征

- 每个系列的每个点都有自己的 X 和 Y 值。
- 每个系列可以包含不同数量的点。

定义图表数据（ChartData）对象

C1Chart 有一个特殊的层，该层包含数据关联的类，集合和属性。 本章节对每个数据对象的细节和如何访问的信息详细地逐一介绍。



定义 ChartGroup 对象

图表中的数据需要被组织到 ChartGroup 对象。每个图表包含两个 ChartGroup（尽管大部
分图表只是使用第一个 ChartGroup），其中每个都被视为一个单独的实体。ChartGroup 允许超过一个图表显示在 ChartArea
并且提供多种对图表特有属性的访问。
在 C1Chart 中，一个 ChartGroup 对象代表一个图表组。ChartGroup 对象被组织到
ChartGroupsCollection 中，ChartGroupsCollection 可以通过 ChartGroups 对象来访问。这个集合提供两种方法访问 ChartGroups.
首先单独的图表组可以通过集合来访问。例如，下面的代码列出了如何通过
ChartGroupsCollection 来访问单独的图表组。

- Visual Basic

```
C1Chart1.ChartGroups.ChartGroupsCollection(0).ChartType = Chart2DTypeEnum.XYPlot
```

- C#

```
c1Chart1.ChartGroups.ChartGroupsCollection[0].ChartType = Chart2DTypeEnum.XYPlot;
```

其次，ChartGroup 对象的 Group0 和 Group1 属性返回相关的 ChartGroup,并且允许用下列的代码规避长集合名。

- Visual Basic

```
C1Chart1.ChartGroups.Group1.ChartType = Chart2DTypeEnum.XYPlot
```

- C#

```
c1Chart1.ChartGroups.Group1.ChartType = Chart2DTypeEnum.XYPlot;
```

如下面的代码所示，ChartGroupsCollection 允许通常的迭代方法：

- Visual Basic

```
Dim cg As ChartGroup For Each cg In C1Chart1.ChartGroups.ChartGroupsCollection cg.ChartType = Chart2DTypeEnum.XYPlot
Next
```

- C#

```
ChartGroup cg; foreach ( cg In c1Chart1.ChartGroups.ChartGroupsCollection ) cg.ChartType = Chart2DTypeEnum.XYPlot;
```

定义 ChartData 对象

ChartGroup 对象还包含 ChartData 对象。ChartData 对象包含 Hole 属性, FunctionsList 属性, 能返回 ChartDataSeries 对象的 SeriesList 属性，PointStylesList 属性和 TrendsList 集合
属性。这个 ChartGroup 对象基本上能够访问所有数据相关的对象和图表的所有属性：

- Visual Basic

```
C1Chart1.ChartGroups.ChartGroupsCollection(0).ChartData
```

- C#

```
c1Chart1.ChartGroups.ChartGroupsCollection[0].ChartData;
```

FunctionsList 集合包含了绘制图表数据和图表区域的函数。FunctionsList 属性能够得到现在的 ChartData 对象关联的 FunctionsCollection 对象。
PointStyles 属性提供了一个机制可以用不同的视觉效果标注特殊的数据点，和同一个数据系列的其他点区分开来。PointStyles 包含在 PointStylesCollection 中。
SeriesList 属性返回一个 ChartDataSeriesCollection，ChartDataSeriesCollection 是一个 ChartDataSeries 对象的集合。ChartDataSeries 对象包含图表的所有系列和数据。Series 对象包含 ChartDataArray，后者持有图表数据。
TrendLine 对象将趋势线分为两组，包括回归和非回归。回归趋势线表示的是包括多项式，指数，对数和傅立叶函数这类关于数据的函数趋势.

ChartData 的对象,还可以访问 Hole 属性，数据的 Hole 可以打破数据集的连续性。

定义 ChartDataSeries 对象

C1Chart 最重要的对象之一就是 ChartDataSeries 对象。这个数据系列包含图表中的所有数据和许多重要的数据相关的属性。ChartDataSeries 包含在 ChartData 类型的 SeriesList 对象集合中，ChartData 对象是 ChartGroup 的一个对象。

- Visual Basic

```
C1Chart1.ChartGroups.ChartGroupsCollection(0).ChartData.SeriesList(0)
```

- C#

```
c1Chart1.ChartGroups.ChartGroupsCollection[0].ChartData.SeriesList[0];
```

这个系列对象对数据和数据访问来说非常重要，原因有两个：

1. a. 首先，ChartData 对象的 SeriesList 集合中系列的数量，决定了在能够显示在图表中数据系列的数量。如果一个 x-y 的图表希望有五套图，那么在 SeriesList 就应该有五个系列。
   b. 其次，系列中 ChartDataArray 对象的 x、y、y1、y2、y3 定义了多少个值就决定了那个系列有多少个点，和如何绘制到图表中。如果图表 0 系列的 x 和 y 数组对象都有 10 个值，1 系列 x 和 y 数组都有 5 个值，那么第一个系列就会有 10 个点，第二个系列就会有 5 个点。

需要注意的是一个系列点的数量是数据组对象中长的那个元素的数量。例如，一个 XY 图表使用了 X 和 Y 数组。如果 X 有 5 个元素，Y 有 3 个元素，那么点的数量是 5，Y 值剩下的 2 个元素被假定为数据漏洞。即使 Y1 数组有十个元素，点的数量还是 5，因为 XY 图表不使用 Y1 数组。但是，如果图表类型变为 HiLo，那么 XY 和 Y1 都要被使用，所以点的数量是 10.X 和 Y 数组不存在的元素都被假定为数据漏洞。

定义 ChartDataArray 对象

每个系列包含六个 ChartDataArray 对象。X、Y、Y1、Y2、Y3 数组对象持有数据组和 X 轴， Y 轴，Y1 轴，Y2 轴，Y3 轴对应。第六个数据组对象是 PointData 对象。这个对象的数据是 PointF 数组或者 Point 值。这个对象用于作为 X，Y,Y1，Y2，Y3 数据组对象的替代选择。
每个 ChartDataArray 都是一个只读对象，只能通过两种方法修改。CopyDataIn 和 CopyDataOut 的方法拷贝数据到这些对象，并且从这些对象中返回数据组。
X,Y,Y1，Y2，Y3 数据组对象接收数据类型，这样不同的.NET 数据组类型可以被输入到 ChartDataArray 对象中（除了 PointData）。

输入和修改图表数据

在ChartDataSeries对象中，六个ChartDataArray对象包含有绘制图表用的数据组。X,Y,Y1，Y2，Y3 数据组对象持有并且返回 X 和 Y 轴用的数据组，同时 PointData 数据组对象持有并返回 X 轴和 Y 轴用的 PointF 和 Point 对象。

9.2.1 加载和提取图表数据

CopyDataIn 方法
在 C1Chart 中，X，Y,Y1，Y2,Y3 数据组对象接收对象数据类型。因此，不同的.NET 类型数据组可能被输入到 ChartDataArray 对象中（除了 PointData）。ChartDataArray 的 CopyDataIn 方法接收一个对象数据类型（可以是一组不同的类型）并把它装载到数据组中。
一个好的实现应该在图表绘制之前维护数据系列的一套数组，用数组的值更新 C1Chart 数据组对象。这样就能够完全控制数据，并可以批处理数据值。以下代码给出了一个简单的例子，说明数据组如何被构建并设定到图表的数据组对象中去的：

- Visual Basic

```
Dim xp(9) As Single
Dim yp(9) As Single
Dim i As Integer For i = 0 To 9 xp = i yp = i * i
Next i
With C1Chart1.ChartGroups.ChartGroupsCollection(0).ChartData.SeriesList(0)
.X.CopyDataIn(xp)
.Y.CopyDataIn(yp)
End With
```

- C#

```
float xp(10); float yp(10); int i;
for( i=0; i < 10; i++)

{ xp[i] = i; yp[i] = i * i;


}

ChartDataSeries cds = c1Chart1.ChartGroups.ChartGroupsCollection[0].ChartData.SeriesList[0]; cds.X.CopyDataIn(xp);
cds.Y.CopyDataIn(yp);
```

CopyDataOut 方法
CopyDataIn 方法给 ChartDataArray 对象装载数据组，而 CopyDataOut 方法从数据组对象中提取数据。这个方法返回一个对象数据类型，包含数据组。

- Visual Basic

```
Dim xpo As Object Dim cds As ChartDataSeries cds = C1Chart1.ChartGroups.ChartGroupsCollection(0).ChartData.SeriesList(0) xpo = cds.X.CopyDataOut()
```

- C#

```
object xpo; ChartDataSeries cds; cds = c1Chart1.ChartGroups.ChartGroupsCollection[0].ChartData.SeriesList[0]; xpo = cds.X.CopyDataOut();
```

为了保证这个对象转换的数据类型正确，CopyDataOut 方法也能够接收 System.Type 参数，该参数产生一个特定类型的数组。因为参数需要实际的数据类型，代码必须包含 GetType() 函数(typeof()在c#)来传输正确的类型。在接下来的例子中，使用 CopyDataOut 方法从图表的 ChartDataArray 对象中返回数据组。但是数据组的类型是 Single：

- Visual Basic

```
Dim xpo As Single()
Dim ypo As Single()
With C1Chart1.ChartGroups.ChartGroupsCollection(0).ChartData.SeriesList(0) xpo = .X.CopyDataOut(GetType(Single())) ypo = .Y.CopyDataOut(GetType(Single())) End With
```

- C#

```
ChartDataSeries cds;

cds = c1Chart1.ChartGroups.ChartGroupsCollection[0].ChartData.SeriesList[0]; float[] xpo =
cds.X.CopyDataOut(typeof(float[])); float[] ypo = cds.Y.CopyDataOut(typeof(float[]));
```

注意： C#用户，CopyDataOut 的实现有一点不同。C#中，数组数据类型不能被设定到类型对象的变量中。如果对象被转换成了正确的数据类型，数组就能设定到对象中。

- Visual Basic

```
Dim xpo() As Single xpo = CType(C1Chart1.ChartGroups.ChartGroupsCollection(0), Single())_
.ChartData.SeriesList(0).X.CopyDataOut(GetType(Single()))
```

- C#

```
float[] xpo; xpo =

(float[])c1Chart1.ChartGroups.ChartGroupsCollection[0].ChartData .SeriesList[0].X.CopyDataOut(type of(float[]));
```

改变数据组的数据元素

将单独的数据元素集成到数据组的方法涉及到的 ChartDataSeries 和 ChartDataArray 对象。使用 ChartDataArray 集合编辑器使得在.NET 属性窗口中设定单独的值非常的容易。X,Y,Y1，Y2，Y3 和 PointData 数据组都有数据组编辑器。点击 SeriesList 集合编辑器（在 ChartGroupsCollection 编辑器的 ChartData 节点下可用）旁边的省略符号就可以打开这些编辑器。和.NET 集合编辑器相同，ChartDataArray 编辑器在文本框的左边有数组的索引，右边显示数组的值。

注意：ChartDataArray 集合编辑器有让显示格式在日期，日期时间和时间之间转换的函数。如果 ChartDataArray 值是日期时间类型，点击列头这个值就能在日期和时间格式之间转换。
运行时 ChartDataArray 元素和一般数据组元素一样是可用的，改变数组对象的值，只需简单改变一个数组的值。

- Visual Basic

```

C1Chart1.ChartGroups.Group0.ChartData.SeriesList(0).X(4) = 4

```

- C#

```
c1Chart1.ChartGroups.Group0.ChartData.SeriesList[0].X[4] = 4;
```

从 ChartDataArray 返回值需要一个相似的过程：

- Visual Basic

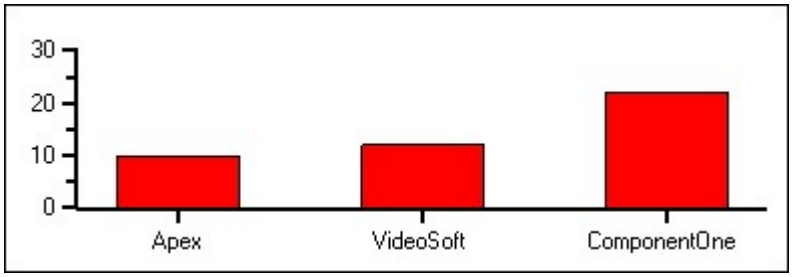
```
Dim xval As Single
xval = C1Chart1.ChartGroups.Group0.ChartData.SeriesList(0).X(4)
```

- C#

```
float xval;

xval = c1Chart1.ChartGroups.Group0.ChartData.SeriesList[0].X[4];
```

显示字符串值作为注释

有时，用字符串的值给 X 轴和 Y 轴作为注释，其他轴用数字作为注释，这是非常方便的。
例如，假设一个数据源包含两列，一个是公司名称（字符串），另一个是这个公司购买 ComponentOne
产品的数量。创建一个条形图来展示这些数据，你可以在 Y 轴创建一个产品数量的系列，然后创建一个数组用于计算数量，并把这个数组作为 X 值。然后给 X 轴创建 ValueLabels，每个公司名称和计算的数量相匹配，从而产生图表如下：



由于这种情况经常发生，C1Chart 接收 ChartDataArray 的字符串，ChartDataArray 会自动
列举字符串的值，并且创建相关轴的 ValueLabels。使用这个功能，就有可能（并且非常容易）把 ChartDataSeries 中的 DataField
属性直接设定为一个字符串的值，并且也能得到字符串的值。

- Visual Basic

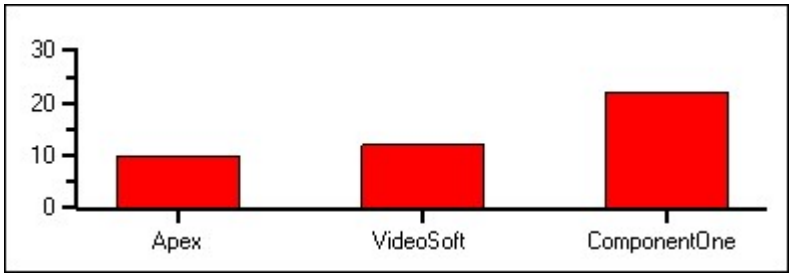
```
C1Chart1.ChartGroups(0).ChartData.SeriesList(0).X.DataField = "CompanyNames"
C1Chart1.ChartGroups(0).ChartData.SeriesList(0).Y.DataField = "UnitsPurchased"
```

- C#

```
c1Chart1.ChartGroups[0].ChartData.SeriesList[0].X.DataField = "CompanyNames";
c1Chart1.ChartGroups[0].ChartData.SeriesList[0].Y.DataField = "UnitsPurchased";
```

混合 ChartDataArray 输入

C1Chart 允许一些 ChartDataArrays 被绑定，其他的使用 ChartDataSeries 的
AutoEnumerate 属性，或者使用 ChartDataArray 的
CopyDataIn()方法来设定。当数据绑定时，图表使用所有的绑定数据作为定义系列数据的数据源，直接使用这些系列和其他图表属性，在图表的绘图区绘制出来。当 AutoEnumerate 属性设定为 True，ChartDataArrays/系列的 X
值自动设定为索引值。尽管有不同的方法设置数据系列，提供了最大灵活性，还是需要删除额外的默认系列。



例如，上面的条形图，只包含一个系列（包含一个公司购买 ComponentOne 产品的数量）。所以如果使用了数据绑定给 C1Chart
提供输入，那么 4 个默认图表系列中的 3 个就要被删除，
删除这些系列可以在设计时做，也可以在运行时做。

使用 Point 值设定 X 和 Y 数据组

对使用 X 和 Y 轴（XY-Polt，Bar，等等）的图表，PointData 属性非常方便，ChartDataSeries 对象的 Point 属性允许输入一组 PonitF 点值。这个属性对使用点数据的应用非常方便，但是这个数组只能设定 X 和 Y 数据组，这个属性不能改变 Y1，Y2,Y3 数据组的值。
下面的例子创建了一个 PointF 数组，并且把它装载到 ChartDataSersies 中：
Visual Basic

```
Dim ps() As PointF = _
{ _
New PointF(30.0F, 20.0F), _
New PointF(60.0F, 24.0F), _
New PointF(90.0F, 42.0F), _
New PointF(120.0F, 13.0F), _
New PointF(150.0F, 10.0F) _
}
Dim s As New ChartDataSeries()
C1Chart1.ChartGroups.Group1.ChartData.SeriesList.Add(s) s.PointData.CopyDataIn(ps)
```

C#

```
PointF [] ps =

{ new PointF(30.0F, 20.0F), new PointF(60.0F, 24.0F), new PointF(90.0F, 42.0F), new PointF(120.0F, 13.0F), new
PointF(150.0F, 10.0F)
};
ChartDataSeries s = new ChartDataSeries();
c1Chart1.ChartGroups.Group1.ChartData.SeriesList.Add(s); s.PointData.CopyDataIn(ps);
```

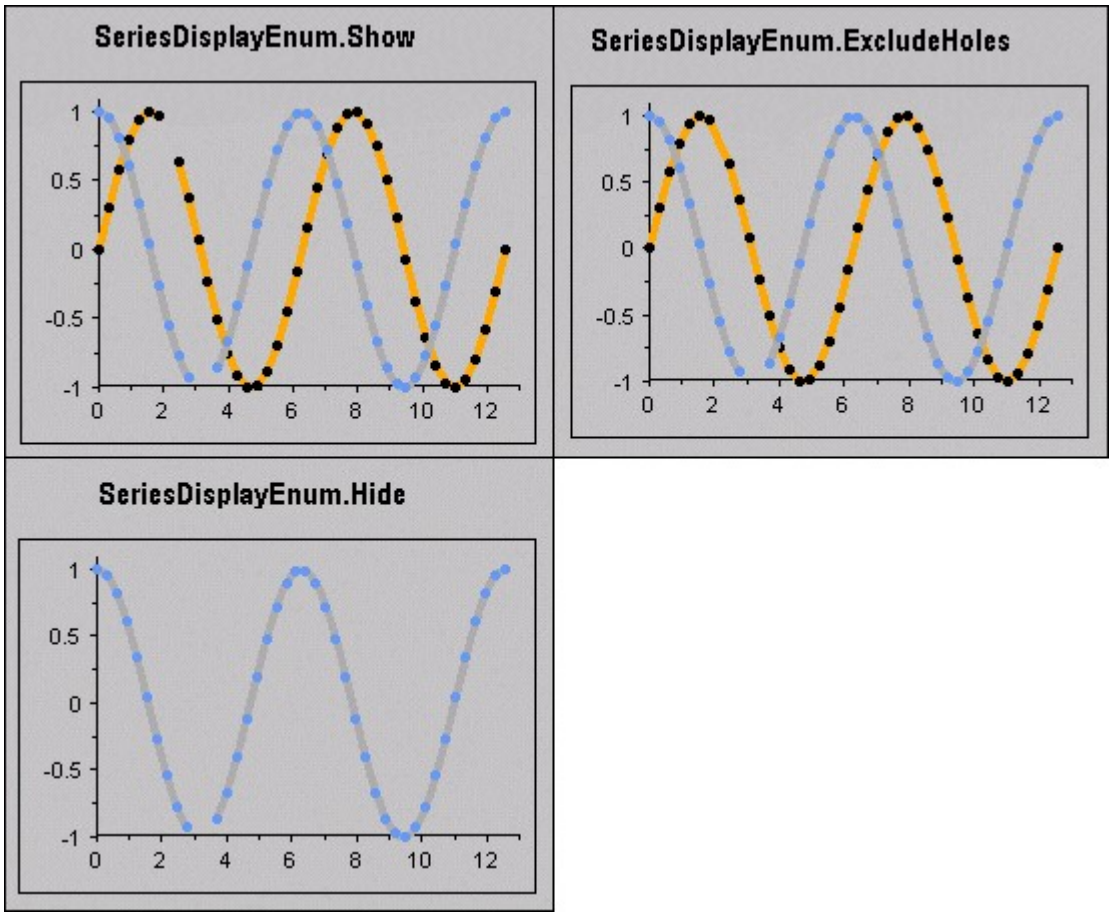
定制 ChartDataSeries

本章节描述在 ChartArea 中 ChartDataSeries 如何显示和隐藏排除漏洞，以及如何在
C1Chart 的特定图例中如何排除一个系列

显示、排除或隐藏一个系列

假设有成百个系列需要显示在图表中，由于图表只能这么大，必须控制显示的系列。
ChartDataSeries 的 Display 属性提供了这种功能。Display 属性接受一个 SeriesDisplayEnum 枚举类型。把这个属性设定成不同的值，就允许一个系列被显示，被隐藏或者被排除

值	描述
SeriesDisplayEnum.Show	系列显示在 ChartArea 中，这是默认值。
SeriesDisplayEnum.Hide	系列不显示在 ChartArea 中，但是 ChartArea(最大值和最小值)不会因为系列没有显示而改变。
SeriesDisplayEnum.Exclude	系列不会显示在 ChartArea 中，但是 ChartArea 会因为系列没有显示而改变大小。
SeriesDisplayEnum.ExcludeHoles	系列显示，但是 Data Hole 值被忽略。



从图例中除去一个系列

有时图形的形状，线和曲线不代表数据，更希望代表数据区域，例如代表误差范围，等等。
在这种情况下，多余的系列可能被加到图形图表中，但是应该在图例中除去，每个 ChartDataSeries 包含一个叫做 LegendEntry 的 Boolean 属性，如果设定成 True，也就是默认值，该系列将列在图例中，如果是 False,这个系列还是会被画出来，但是不会别列到图例中。

绘制不规则的数据

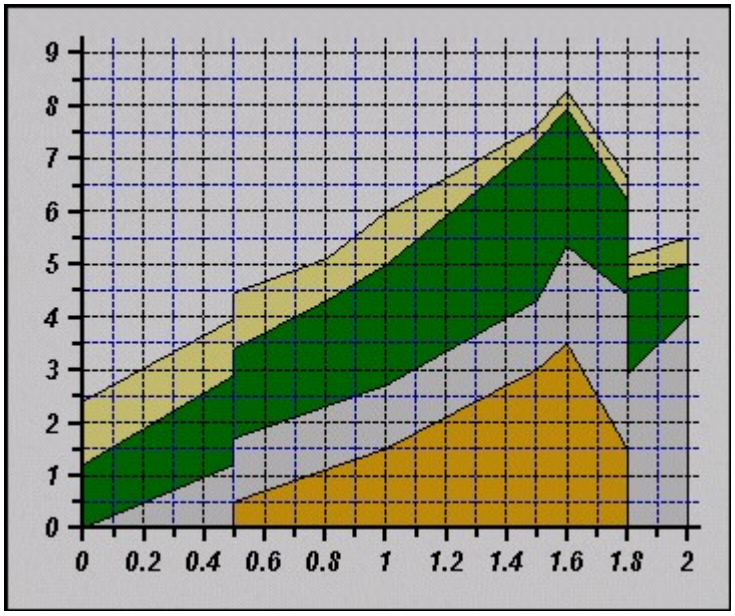
数据集,尤其当他们是动态创建的,有时会有不规则的属性,很难让一个绘图控件来处理。
因此，在把数据绘制到图表之前，花费了大量的时间，计算这些不一致的数据。虽然 ClChart
不能自动改变这些值，但是它能够接受不规则的数据，然后用一种逻辑和一致的方式来处理这些数据。

不规则堆积图表数据

堆积图表是指图表在一个基础的系列上，堆积一个或多个系列。图表中所有 Y 值都指的是和前一个系列 Y 值的间距。当数据装载在 X
的值与数组中的完全匹配时，堆积图表是一个非常简单是事情，所有的系列都堆积得非常整齐。
ClChart 在一般布局中为接收数据提供了更多的自由。一般布局意味着不是所有的系列都必须有完全相同的 X
值，这种自由使图表接收了一些非标准的数据，在使用堆积图表时就会遇到一些困难。

不规则 X 轴数据

如果数据集中的第一个系列不包含和其他系列的最小的点和最大的点相匹配的点，那么堆积图表就会遇到更大的困难。
如果图表的第一个系列没有足够数量的 X 值，不能成为一个连续的系列，那么这个系列将不能像下图一样有一个明确的开始点。ClChart
用一种逻辑的方式处理这些非规则的数据，第一个系列数据的 Y 值不存在，将作为 0
处理，然后每个连续的系列都被添加了这些数据从而产生了中断，下图就是个最好的示例。

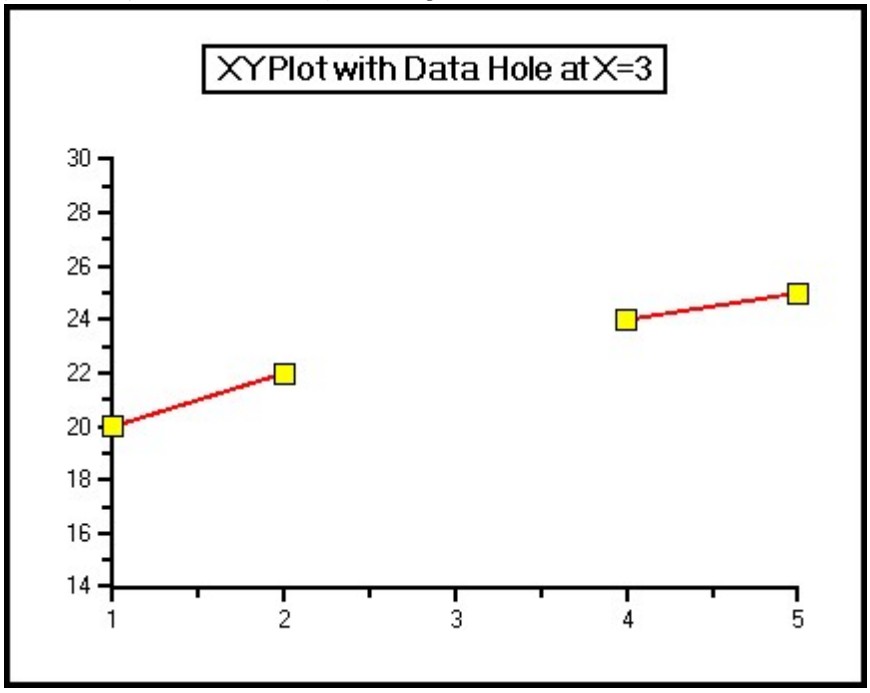


插入 Y 值数据

堆积图表时如果一个连续系列的 X值和第一个系列不一致也会给推挤图表增加困难。例如，图表的第一个系列 X 值有 2,4,6 和
8，而第二个系列有值 3,4,5 和 9.图表在判断在哪个位置放置第二个系列的第一个点的时候就会遇到难题。第二系列的点的值必须添加 Y
值数据来定位点在第一个系列的头两个点之间的某处，而这个数据时数据集中没有的。
不表示数据集中的数据，ClChart 以逻辑的方式处理这些不规则数据。根据线的走势，
ClChart 在第一个系列的前两个点之间插入一个虚拟的点。这个插入的点的值加到第二个系列的
Y 值上从而找到堆积图表用的新的点。

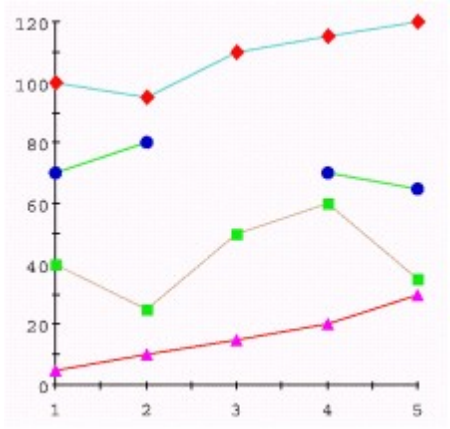
指定 Data Hole

使用组织好的数据作图，有时候必须在图表中标示出不可用的特殊的点。例如，你能标示出缺少的信息。一个 Data Hole 是对正在绘制的数据一个打断，用于在图表中标示出缺失的数据。
在 C1Chart 中，Hole 是指一个数据点不具有其他数据所有的显著特性，或者在一个数据组中是没有用的数据。例如，假设 Hole 的值被设定为-1000，如果一个系列的第三个点的数据不可用，一个中断或一个漏洞就会产生，绘制这个点的 Y 值到-1000.
默认的 Hole 值对 Visual Basic 来说是 Single MaxValue ，对 C#是 float MaxValue。下图是一个在 X—Y Plot 图表中数据漏洞的例子

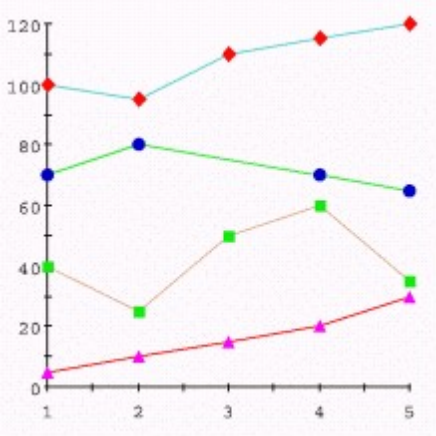


忽略 Data Hole

当数组中的数据有着相同的长度，所有的数组都很一致的情况下，编程是非常简单的。即使有一些数据点丢失了，也不希望标示出这些丢失的数据。在这种情况下，还是可以使用 Data Hole 作为丢失数据的值，C1Chart 可以很简单的完全忽略它们，就好像数据组元素完全没有 Data Hole 一样。
ChartDataSeries 的 Display 属性控制一个系列是否显示。这个属性接收一个 SeriesDisplayEnum 枚举。 如果 Display 属性设定成了 SeriesDisplayEnum.Show，那么 Data Hole 会像上图一样排除到图表之外。 如果属性设定为 SeriesDisplayEnum.ExcludeHoles，系列会显示，但是漏洞被忽略。这就意味着如果显示的是 XY-Plot，Data Hole 前面的点和 Data Hole 后面的点直接连起来如下图所示。注意即使绘制的线穿过了 Data Hole，Data Hole 点还是没有被画出来的。
SeriesDisplayEnum.Show



SeriesDisplayEnum.ExcludeHoles



绘图函数

C1Chart 有一个绘图函数的内置引擎。因为不同的应用使用不同类型的函数给，C1Char 为许多的应用提供了不同类型的函数。
提供的函数有两种类型

1. 一个变量的显式函数。

   - 一个变量的显式函数 例如 y=f （参照 YFunction 类）
   - 几个例子包括：有理函数（rational），线性函数 （linear），多项式函数

（polynomial），二次函数 （ quadratic），对数函数（logarithmic），和指数函数（exponential）。

- 科学家和工程师经常使用的函数，这些函数可以运用在财务，预报，性能测定等等

1. 参数函数
   - 由一对方程式定义的函数，例如 y=f1(t) 和 x=f2(t) 这里 t 是函数 f1 和 f2 的变量/坐标
   - 参数函数是特殊类型的函数，因为 X 和 Y 坐标有两个独立的函数定义，是分开的变量
   - 参数函数在数学和功能领域代表不同的情况，热传输，水力学，电磁理论，行星运动和相对论理论的一些方面使用了一些这样的函数

关于参数函数的更多信息 参照 ParametricFunction 类在 C1Chart 中，一个代码串可以用来计算 Y
函数和参数函数要使用一个代码串来计算 Y 函数和参数函数，你必须执行下面之一

- 一个包含 C#或者 VB 源代码的解释用字串
- 事件
- 一个支持 C1.Win.C1Chart.ISimpleFunction 层的类

C1Chart 有一个 FunctionBase 集合编辑器， 这个编辑器可以通过 C1Chart 的属性在设计时访问。FunctionBase 集合编辑器由Windows 窗口组成，可以允许用户很方便地编辑/创建函数。 使用这个编辑器，用户可以新增/删除一个过多个函数，选择函数种类（Y 函数或者参数函数）并定义代码类型和代码的语言，就能命名一些函数。关于 FunctionBase 集合编辑器的更多信息，参照 FunctionBase 集合编辑器（47 页）。
例子：说明如何创建 Y 函数和参数函数的完整实例，参照实例，FExplorer，位于 {+}http://helpcentral.componentone.com/Samples.aspx+ .

使用代码串定义函数

当一个解释用代码串用于定义一个函数类中的一个函数（YFunction 或者
ParametricFunction），代码串被编译，编译后的代码动态地被包含到应用中。执行速度和其他编译的代码一样。
这样就有一些优点：

- 当定义函数时，内置的 C#函数或者 VB 函数可以被使用
- 简单地操作代码串就能创建函数。
- 使用 SaveChartToFile() 或者 SaveChartToString()
  方法能将函数作为图表布局的一部分保存下来，并且可以作为数据的一部分直接装载到其他工程中。

但是也存在一些缺点：

- 由于要预编译第一次分配和改变代码内容时比较浪费时间
- 源代码的类型只能是 C#或者 VB

为了简单，使用一个变量的显式函数 YFunction 类对象. 这个对象有一个 c o d e 属性，
CodeText。对 YFunction 对象，单独的变量始终被假设为 X
对参数函数，必须使用 ParametricFunction 类对象定义一对方程式。这个对象有两个属性，每个坐标有一个。属性 CodeTextX 和 CodeTextY
接受编码，每个单独变量始终被假设为 t
对上面的两个类对象，单独的变量的值被假定为从最小值到最大值之间是均匀分布的，使用
PlotNumPoints 定义数据点的数量。
对于复杂的函数，使用 CodeText, CodeTextX 和 CodeTextY 提供 3 种编码类型。编码类型包括 Formula, Method 和
Unit，下面的章节逐一进行介绍。
  Formula 编码类型
对应 Formula 编码类型，一个函数的 CodeText 属性必须包含使用一个参数计算函数值的
代码。代码必须写到一行并且代表方程式的右半边。

- Visual Basic

```
Dim yf As C1.Win.C1Chart.YFunction = New C1.Win.C1Chart.YFunction() yf.CodeType =
C1.Win.C1Chart.FunctionCodeTypeEnum.Formula yf.CodeLanguage = C1.Win.C1Chart.FunctionCodeLanguageEnum.VB yf.CodeText = "x*x"
yf.MinX = -5 yf.MaxX = 5 yf.LineStyle.Color = Color.Red yf.LineStyle.Thickness = 3
C1Chart1.ChartGroups(0).ChartData.FunctionsList.Add(yf)
```

- C#

```
C1.Win.C1Chart.YFunction yf = new C1.Win.C1Chart.YFunction(); yf.CodeType = C1.Win.C1Chart.FunctionCodeTypeEnum.Formula;
yf.CodeText = "x*x"; yf.MinX = -5; yf.MaxX = 5; yf.LineStyle.Color = Color.Red; yf.LineStyle.Thickness = 3;

c1Chart1.ChartGroups[0].ChartData.FunctionsList.Add(yf);
```

  Method 编码类型
对于 Method 编码类型，一个函数类的 CodeText 属性必须包含方法的主体，这个方法用于
计算函数的值并且能明确地返回值。希望的返回值的类型是 Double。注意 VB 语法要求在编码
中每个声明的结尾都需要有 vbNewLines

- Visual Basic

```
Dim code As String = _
"Dim x2 As Double = x*x" & vbNewLine & _ "if x<0 then " & vbNewLine & _ " return x" & vbNewLine & _ "else" & vbNewLine & _
" return 0.5*x2" & vbNewLine & _
" End If"
Dim yf As C1.Win.C1Chart.YFunction = New C1.Win.C1Chart.YFunction() yf.CodeType =
C1.Win.C1Chart.FunctionCodeTypeEnum.Method yf.CodeLanguage = C1.Win.C1Chart.FunctionCodeLanguageEnum.VB yf.CodeText = code
yf.MinX = -5 yf.MaxX = 5 yf.LineStyle.Color = Color.Blue yf.LineStyle.Thickness = 3
C1Chart1.ChartGroups(0).ChartData.FunctionsList.Add(yf)
```

- C#

```
string code ="double x2 = x*x;" +
"if( x<0)" +
" return x;" +
"else" +
" return 0.5*x2;";
C1.Win.C1Chart.YFunction yf = new C1.Win.C1Chart.YFunction(); yf.CodeType = C1.Win.C1Chart.FunctionCodeTypeEnum.Method;
yf.CodeText = code; yf.MinX = -5; yf.MaxX = 5; yf.LineStyle.Color = Color.Blue; yf.LineStyle.Thickness = 2;

c1Chart1.ChartGroups[0].ChartData.FunctionsList.Add(yf);
```

Unit 编码类型
对于 Unit 编码类型，一个函数的 CodeText 属性必须包含完全编译的 Unit 文本。Unit 必须在 UserFunction namespace 中包含"Calculator"类，而且必须执行 ISimpleFunction 接口。

- Visual Basic

```
Dim code As String = _
"Namespace UserFunction" & vbNewLine & _
" Class Calculator" & vbNewLine & _
" Implements ISimpleFunction" & vbNewLine & _
" Public Function Calculate(x As Double) As Double _" & vbNewLine & _ " Implements
ISimpleFunction.Calculate" & vbNewLine & _ " Dim x2 As Double = x*x" & vbNewLine & _ " if( x<0)" & vbNewLine & _ "
return -x" & vbNewLine & _ " else" & vbNewLine & _ " return -0.5*x2" & vbNewLine & _ " End If" & vbNewLine & _ "
End Function" & vbNewLine & _ " End Class" & vbNewLine & _ "End Namespace"
Dim yf As C1.Win.C1Chart.YFunction = New C1.Win.C1Chart.YFunction() yf.CodeType =
C1.Win.C1Chart.FunctionCodeTypeEnum.Unit yf.CodeLanguage = C1.Win.C1Chart.FunctionCodeLanguageEnum.VB yf.CodeText =
code yf.MinX = -5 yf.MaxX = 5 yf.LineStyle.Color = Color.Green yf.LineStyle.Thickness = 3
C1Chart1.ChartGroups(0).ChartData.FunctionsList.Add(yf)
```

- C#

```
string code =
"namespace UserFunction" +
"{" +
" class Calculator : ISimpleFunction" + " {" +
" public double Calculate(double x)" + " {" + " double x2 = x*x;" + " if( x<0)" + " return -x;" + " else" + "
return -0.5*x2;" + " }" + " }" + "}";
C1.Win.C1Chart.YFunction yf = new C1.Win.C1Chart.YFunction(); yf.CodeType =
C1.Win.C1Chart.FunctionCodeTypeEnum.Unit; yf.CodeText = code; yf.MinX = -5; yf.MaxX = 5;

yf.LineStyle.Color = Color.Green; yf.LineStyle.Thickness = 2; c1Chart1.ChartGroups[0].ChartData.FunctionsList.Add(
yf);
```

计算函数的值

下面的内容介绍如何使用 CalculateX 和 CaleculateY 事件或者 ISimpleFunction 接口的 Calculate 方法计算恰当的函数的值。
使用事件计算函数的值
不想把源代码加到 CodeText, CodeTextX 和 CodeTextY 属性，那么就可以使用事件委托定义一个事件方法来计算恰当的函数的值。当使用事件委托，对于 YFunction 类对象，在 CalculateY 事件中计算函数的值。对于 ParametricFunction 类对象，两个事件，CalculateX 和 CalculateY 必须被设定，用于计算坐标。
非空的事件委托说明了事件应该被用于计算对应的函数的值，即使对应的 CodeText 属性在 Yfunction 或者 ParametricFunction 类对象中已经被定义了。 为了使用事件，程序员必须提供合适的事件处理程序。

- Visual Basic

```
Private Sub Button_Click(ByVal sender As System.Object, _ ByVal e As System.EventArgs)
Handles Button.Click
Dim yf As C1.Win.C1Chart.YFunction = New C1.Win.C1Chart.YFunction() AddHandler yf.CalculateY, AddressOf
Function_Calculate yf.MinX = -5 yf.MaxX = 5 yf.LineStyle.Color = Color.DarkBlue yf.LineStyle.Thickness = 3
C1Chart1.ChartGroups(0).ChartData.FunctionsList.Add(yf)
Private Sub Function_Calculate(ByVal sender As Object, _ ByVal e As
C1.Win.C1Chart.CalculateFunctionEventArgs)
e.Result = e.Parameter * e.Parameter * e.Parameter ' y = x*x*x
End Sub
```

- C#

```
private void button_Click(object sender, System.EventArgs e) {
C1.Win.C1Chart.YFunction yf = new C1.Win.C1Chart.YFunction(); yf.MinX = -5; yf.MaxX = 5;

yf.LineStyle.Color = Color.DarkBlue; yf.LineStyle.Thickness = 2; yf.CalculateY += new
C1.Win.C1Chart.CalculateFunctionEventHandler( Function_Calculate);
c1Chart1.ChartGroups[0].ChartData.FunctionsList.Add( yf);


} void Function_Calculate( object sender, C1.Win.C1Chart.CalculateFunctionEventArgs e)
{
e.Result = e.Parameter * e.Parameter * e.Parameter; // y = x*x*x
}
```

使用 Calculate 方法计算函数的值作为使用 code string 或者 event 的替代方法，程序员还可以执行 C1.Win.C1Chart.ISimpleFunction 层定义一个对象实例。这个对象必须从该层继承并且实现一个公共的函数名，Calculate。这个 Calculate 方法有单独的变量（Double）作为一个参数，并且返回单独的变量（Double）。对于 YFunction 类对象，CustomFunction 属性必须被设定为 ISimpleFunction 执行对象。对于 ParametricFunction 类对象，CustomFunctionX 和 CustomFunctionY 属性必须被设定为相应对象执行的 ISimpleFunction 接口。

- Visual Basic

```
Private Sub Button_Click(ByVal sender As System.Object, _ ByVal e As System.EventArgs)
Handles Button.Click
Dim yf As C1.Win.C1Chart.YFunction = New C1.Win.C1Chart.YFunction() yf.CustomFunction = New CustomFunction()
yf.MinX = -5 yf.MaxX = 5 yf.LineStyle.Color = Color.DarkGreen yf.LineStyle.Thickness = 3
C1Chart1.ChartGroups(0).ChartData.FunctionsList.Add(yf)
End Sub
Public Class CustomFunction
Implements C1.Win.C1Chart.ISimpleFunction
Public Function Calculate(ByVal x As Double) As Double _ Implements C1.Win.C1Chart.ISimpleFunction.Calculate
Return -x * x * x ' y = - x*x*x
End Function
End Class
```

- C#

```
private void button_Click(object sender, System.EventArgs e) {

C1.Win.C1Chart.YFunction yf = new C1.Win.C1Chart.YFunction(); yf.MinX = -5; yf.MaxX = 5; yf.LineStyle.Color =
Color.DarkGreen; yf.LineStyle.Thickness = 2; yf.CustomFunction = new CustomFunction();
c1Chart1.ChartGroups[0].ChartData.FunctionsList.Add( yf);


} class CustomFunction : C1.Win.C1Chart.ISimpleFunction
{ public double Calculate( double x)
{ return -x*x*x; // y = -x*x*x
}
}
```

使用趋势线

趋势线由图表的 TrendLine 对象提供，可以分为两组，包括回归和非回归。在 2D 的图表中，趋势线一般使用 X-Y 先，条状图，或者散点图。
非回归的趋势线有移动平均，平均，最小和最大。移动平均趋势线是在指定时间内平均分布。
回归趋势线表示的是包括多项式，指数，对数和傅立叶函数这类关于数据的函数趋势。

创建趋势线

在设计时创建趋势线
使用趋势线集合编辑器（TrendLine Collection
Editor）可以在设计时创建趋势线。使用这个集合编辑器，你可以添加，修改和删除趋势线，关于趋势线集合编辑器的更多信息，请参见
TrendLine Collection Editor （54 页).
以编程方式创建趋势线
以编程的方式创建趋势线，创建 TrendLine 对象实例并且设置它的属性。趋势线创建可以使用构造器或者使用 TrendLinesCollection 的 AddNewTrendLine.TrendLinesCollection()方法。
下面的编码给图表中的 ChartGroup(0)添加了红色的趋势线。

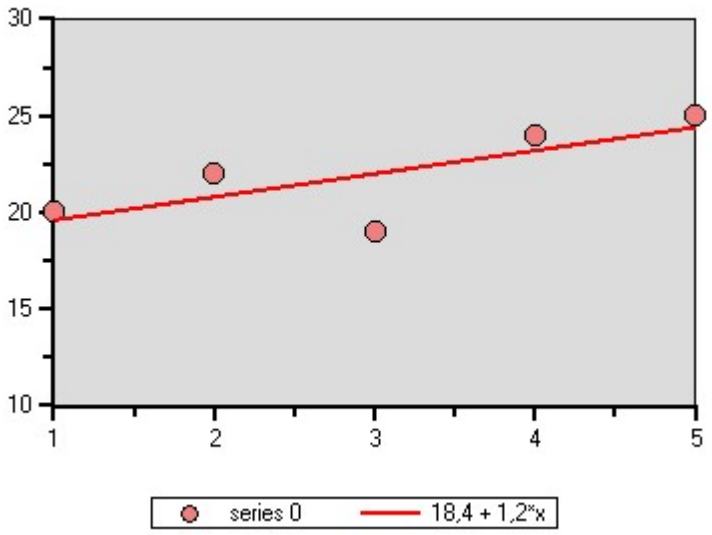
- Visual Basic

```
Dim tl As C1.Win.C1Chart.TrendLine = New C1.Win.C1Chart.TrendLine() tl.SeriesIndex = 0 tl.LineStyle.Color = Color.Red
tl.LineStyle.Thickness = 2 tl.Text = "{#FORMULA}"
c1Chart1.ChartGroups(0).ChartData.TrendsList.Add(tl)
```

- C#

```
C1.Win.C1Chart.TrendLine tl = new C1.Win.C1Chart.TrendLine(); tl.SeriesIndex = 0; tl.LineStyle.Color = Color.Red;
tl.LineStyle.Thickness = 2; tl.Text = "{#FORMULA}";

c1Chart1.ChartGroups[0].ChartData.TrendsList.Add( tl);
```

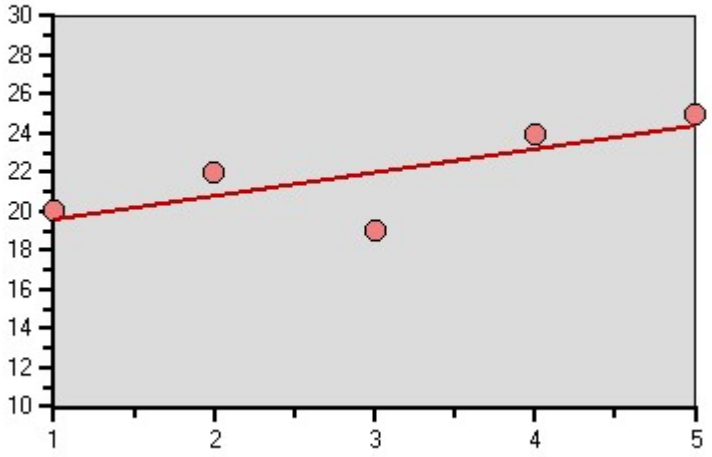
图例如下红色的先连接了第一个和最后一个数据点。



回归趋势线

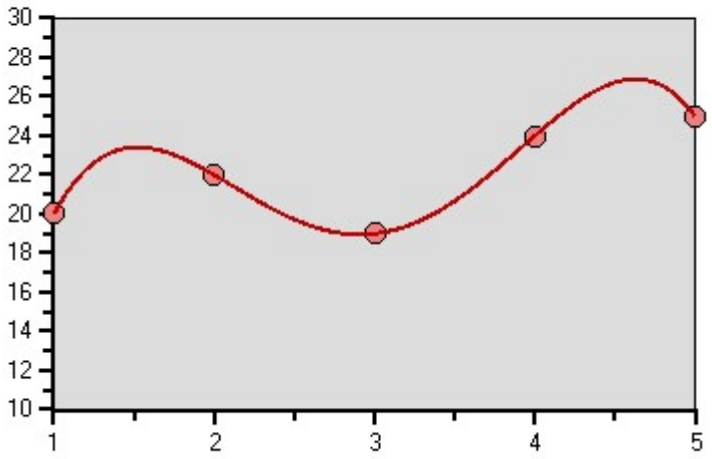
回归趋势线有两个相关的类对象，分别为 RegressionOptions 和
RegressionStatistics.这些对象描述了趋势线所希望的形式，通过统计合计结果趋势线。
回归选项
RegressionOption 对象允许回归模型的规范。NumTerms
属性定义一个回归所需要的系数的个数，和回归有关的仅是项的不等的个数（多项式和傅里叶）。
对于一个多项式回归，项的个数要比由此产生的多项式的阶多一。一个多项式的最多的项数
是数据点的数量，最小的项数是二（直线）
下面的图表示了一个直线的回归（由于有 2 个项）

RegressionOptions.NumTerms = 2



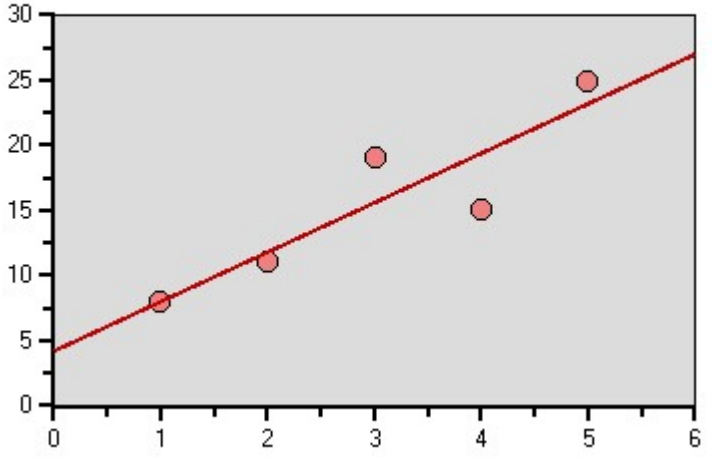
下面的图表示一个多项式回归（项数超过 2）

RegressionOptions.NumTerms = 5

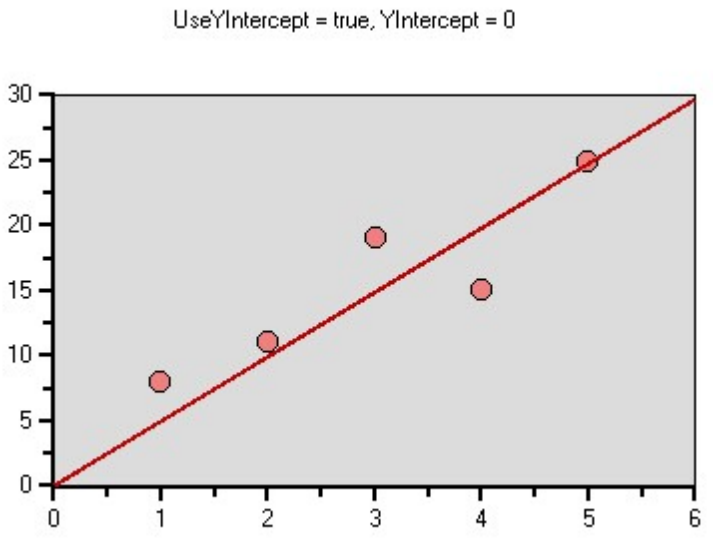


UseYIntercept 属性控制多项式回归的第一个项数是否是固定的，当 UseYIntercept 是True，趋势线截取 Yintercept 属性定义的点作为Y轴上X＝0的点下图描述了 UseYIntercept 属性是 False 时的一个非固定的直线回归线

UseYIntercept = false



下图表示了一个固定的直线回归，因为趋势线设定为截取 RegressionOptions.Yintercept 属性定义的在 Y 坐标上 X=0 的线。



回归统计
RegressionStatistics属性返回了一个包含回归模型的统计结果的RegressionStatistics对象。当趋势线不是回归的或者现在的数据不能解决回归，RegressionStatistics 属性返回 Nothing（VB）
或者空（C#）。
下面的表定义了统计方程式所需要的值。值 定义
n 点的个数
nt 回归系数的个数

1. 值 错误！对象不能从编辑域代码中不能被创建。
2. 值 错误！对象不能从编辑域代码中不能被创建。 mean y 值 错误！对象不能从编辑域代码中不能被创建。 Fitted y 值 错误！对象不能从编辑域代码中不能被创建。

下面的表表示 RegressionStatistics 对象的名称，描述和语法。

名称	描述	公式	公式（UseYIntercept=true）
Ssr	平方和取决于回归统计	错误！对象不能从编辑域代码中不能被创建。	错误！对象不能从编辑域代码中不能被创建。
Sse	平方和取决于错误	错误！对象不能从编辑域代码中不能被创建。	错误！对象不能从编辑域代码中不能被创建。
Rsq	确定系数	错误！对象不能从编辑域代码中不能被创建。	错误！对象不能从编辑域代码中不能被创建。
DF	自由度数	错误！对象不能从编辑域代码中不能被创建。	错误！对象不能从编辑域代码中不能被创建。
F	F 值（F 统计）	错误！对象不能从编辑域代码中不能被创建。	错误！对象不能从编辑域代码中不能被创建。

自定义趋势线

实现一个自定义的趋势线，一个类必须实现创建 ICustomTrendLine 接口。这个类的实例被分配给一个 TrendLine 对象的 CustomTrendLine 属性。这种情况下趋势线的所有的点都必须由类完全定义，并且 TrendLineType 属性的设定是不重要的。下面的示例编码实现了一个和数据范围同步的自定义趋势线。

- Visual Basic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
```

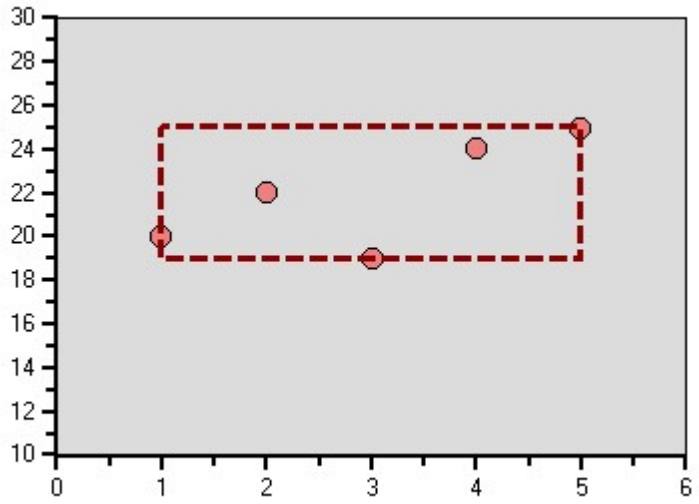
```
Handles Button1.Click
    Dim tl As C1.Win.C1Chart.TrendLine = _
c1Chart1.ChartGroups(0).ChartData.TrendsList.AddNewTrendLine()
    tl.LineStyle.Thickness = 3
    tl.LineStyle.Pattern = C1.Win.C1Chart.LinePatternEnum.Dash
    tl.CustomTrendLine = New CustomTrendLine()
    End Sub
    Public Class CustomTrendLine_ Implements C1.Win.C1Chart.ICustomTrendLine
    Private _x() As Double
    Private _y() As Double
    Public Sub Calculate(ByVal tl As C1.Win.C1Chart.TrendLine, ByVal x() As Double, _ ByVal y() As
Double)Implements C1.Win.C1Chart.ICustomTrendLine.Calculate
    If x Is Nothing Or x.Length = 0 Or y Is Nothing Or y.Length = 0 Then
    _x = Nothing
    _y = Nothing
    Return
    End If
    Dim xmin As Double = x(0), xmax = x(0)
    Dim ymin As Double = y(0), ymax = y(0)
    Dim i As Integer
    For i = 1 To x.Length - 1
    If x(i) < xmin Then
    xmin = x(i)
    ElseIf x(i) > xmax Then
    End If
    If y(i) < ymin Then
    ymin = y(i)
    ElseIf y(i) > ymax Then
    ymax = y(i)
    End If
    Next
    _x = New Double(4) {}
    _y = New Double(4) {}
    _x(0) = xmin
    _y(0) = ymin
    _x(4) = _x(0)
    _y(4) = _y(0)
    _x(2) = xmax
    _y(2) = ymax
    _x(1) = _x(0)
    _y(1) = _y(2)
    _x(3) = _x(2)
    _y(3) = _y(0)
    End Sub
    Public Function GetXValues() As Double() Implements
```

```
C1.Win.C1Chart.ICustomTrendLine.GetXValues
    Return _x
    End Function
    Public Function GetYValues() As Double() _ Implements
C1.Win.C1Chart.ICustomTrendLine.GetYValues
```

- C#

```
private void button1_Click(object sender, System.EventArgs e) {

C1.Win.C1Chart.TrendLine tl = c1Chart1.ChartGroups[0].ChartData.TrendsList.AddNewTrendLine(); tl.LineStyle.Color = Color.DarkRed; tl.LineStyle.Thickness = 3; tl.LineStyle.Pattern = C1.Win.C1Chart.LinePatternEnum.Dash; tl.CustomTrendLine = new CustomTrendLine();


} public class CustomTrendLine : C1.Win.C1Chart.ICustomTrendLine

{ private double[] _x; private double[] _y; public void Calculate( C1.Win.C1Chart.TrendLine tl, double[] x, double[] y)


{ if( x==null || x.Length==0 || y==null || y.Length==0) {

_x = null; _y = null; return; } double xmin = x[0], xmax = x[0]; double ymin = y[0], ymax = y[0]; for( int i=1; i<x.Length; i++)

{ if( x[i] < xmin) xmin = x[i]; else if( x[i] > xmax) xmax = x[i]; if( y[i] < ymin) ymin = y[i]; else if( y[i] > ymax) ymax = y[i];


}

_x = new double[5];

_y = new double[5]; _x[0] = xmin; _y[0] = ymin;

_x[4] = _x[0]; _y[4] = _y[0];

_x[2] = xmax; _y[2] = ymax;

_x[1] = _x[0]; _y[1] = _y[2];

_x[3] = _x[2]; _y[3] = _y[0];


}

public double[] GetXValues() { return _x;}

public double[] GetYValues() { return _y;}


产品网站：{+}http://www.gcpowertools.com.cn+ 咨询热线：029-88331988-600 165 of 336
public double GetY( double x) { return 0;}


public string Text { get{ return "Custom trend";}}
}
```

上面的内容做出下面的图表 围绕着数据点创建了一个自定义的红色的虚线的趋势线。



使用 PointStyles

PointStyle 提供一种机制可以使用不同的视觉效果标注特定的数据点以便和同一个数据系列的其他数据点区别开来。PointStyles 包含在 PointStylesCollection 中。一个 PointStyle
可以被提供给明确的数据点，这个数据点从系列中通过点索引来选择或者这个数据点是特殊情况下的点，例如：系列的 X 最大值，所有数据的 y 最大值等等。C1Chart 程序员还可以定义使用 Select 事件的客户化条件。PointStyle 包含和 ChartDataSeries LineStyle, SymbolStyle 以及 Offset 相同的一套 visual 属性。这些属性描述了遇到 PointStyle 的客户化条件是数据点的外观。一个
PointStyle 可以以图例项表示出来。

创建 PointStyles

PointStyles 在设计时能够很容易地通过 PointStyles 集合编辑器被创建或者通过
PointStyle 对象编程实现。
在设计时创建 PointStyles
PointStyles 在设计时可以通过 PointStyle 集合编辑器创建。使用集合编辑器，你可以新增，修改和删除点类型。
用程序创建 PointStyles
下面的代码创建了一个 PointStyle 对象的实例，并且设定 LineStyle 和 SymbolStyle属性。

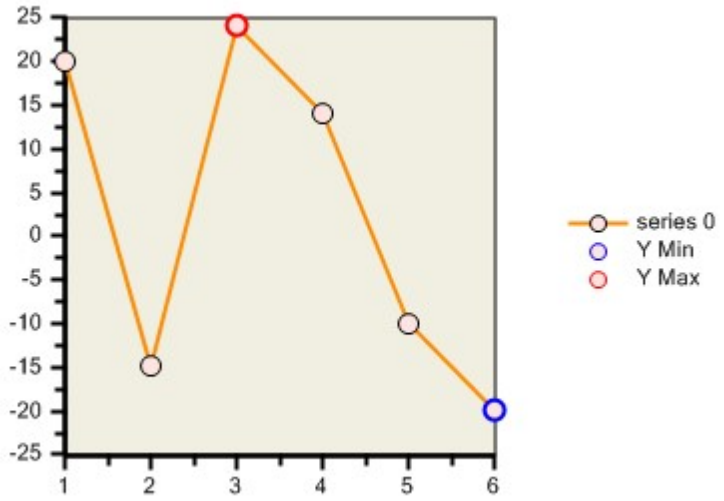
- Visual Basic

```
Dim styles As C1.Win.C1Chart.PointStylesCollection = _ c1Chart1.ChartGroups(0).ChartData.PointStylesList
Dim psmin As C1.Win.C1Chart.PointStyle = styles.AddNewPointStyle() psmin.LineStyle.Pattern =
C1.Win.C1Chart.LinePatternEnum.None psmin.SymbolStyle.Color = Color.MistyRose psmin.SymbolStyle.OutlineColor = Color.Blue
psmin.SymbolStyle.OutlineWidth = 2 psmin.SymbolStyle.Size = 10 psmin.Selection =
C1.Win.C1Chart.PointStyleSelectionEnum.SeriesMinY psmin.Label = "Y Min" psmin.LegendEntry = True
Dim psmax As C1.Win.C1Chart.PointStyle = styles.AddNewPointStyle() psmax.LineStyle.Pattern =
C1.Win.C1Chart.LinePatternEnum.None psmax.SymbolStyle.Color = Color.MistyRose psmax.SymbolStyle.OutlineColor = Color.Red
psmax.SymbolStyle.OutlineWidth = 2 psmax.SymbolStyle.Size = 10 psmax.Selection =
C1.Win.C1Chart.PointStyleSelectionEnum.SeriesMaxY psmax.Label = "Y Max" psmax.LegendEntry = True c1Chart1.Legend.Visible =
True
```

- C#

```
C1.Win.C1Chart.PointStylesCollection styles = c1Chart1.ChartGroups[0].ChartData.PointStylesList;

C1.Win.C1Chart.PointStyle psmin = styles.AddNewPointStyle(); psmin.LineStyle.Pattern =
C1.Win.C1Chart.LinePatternEnum.None; psmin.SymbolStyle.Color = Color.MistyRose; psmin.SymbolStyle.OutlineColor =
Color.Blue; psmin.SymbolStyle.OutlineWidth = 2; psmin.SymbolStyle.Size = 10; psmin.Selection =
C1.Win.C1Chart.PointStyleSelectionEnum.SeriesMinY; psmin.Label = "Y Min"; psmin.LegendEntry = true;
C1.Win.C1Chart.PointStyle psmax = styles.AddNewPointStyle(); psmax.LineStyle.Pattern =
C1.Win.C1Chart.LinePatternEnum.None; psmax.SymbolStyle.Color = Color.MistyRose; psmax.SymbolStyle.OutlineColor =
Color.Red; psmax.SymbolStyle.OutlineWidth = 2; psmax.SymbolStyle.Size = 10; psmax.Selection =
C1.Win.C1Chart.PointStyleSelectionEnum.SeriesMaxY; psmax.Label = "Y Max"; psmax.LegendEntry = true;
c1Chart1.Legend.Visible = true;
```

PointStyles 添加到了 C1Chart 的第一个系列的数据点。还添加了两个特殊点的类型表示 Y 轴的最小值的点和最大值的点。



创建客户化的 PointStyles

要定义客户化的点类型条件，用户必须设定 PointStyleSelectionEnum.Custom 的 Selection 属性，并且提供 Select 事件的事件处理程序。下面的编码创建了一个客户化的点类型。

- Visual Basic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)_
Handles Button1.Click
Dim ps As C1.Win.C1Chart.PointStyle = New C1.Win.C1Chart.PointStyle() ps.Selection =
C1.Win.C1Chart.PointStyleSelectionEnum.Custom AddHandler ps.Select, AddressOf PS_Select
c1Chart1.ChartGroups(0).ChartData.PointStylesList.Add(ps)
End Sub
Private Sub PS_Select(ByVal sender As Object, ByVal e As _ C1.Win.C1Chart.PointStyleSelectEventArgs)
Dim ps As C1.Win.C1Chart.PointStyle = CType(sender, C1.Win.C1Chart.PointStyle)
Dim ds As C1.Win.C1Chart.ChartDataSeries = _ c1Chart1.ChartGroups(0).ChartData(e.SeriesIndex)
Dim y As Double = Convert.ToDouble(ds.Y(e.PointIndex))
If (y < 0) Then Else
ps.LineStyle.Color = Color.Red
End If
e.Selected = True
End Sub
```

- C#

```
private void button1_Click(object sender, System.EventArgs e) {

C1.Win.C1Chart.PointStyle ps = new C1.Win.C1Chart.PointStyle(); ps.Selection =
C1.Win.C1Chart.PointStyleSelectionEnum.Custom; ps.Select += new
C1.Win.C1Chart.PointStyleSelectEventHandler(PS_Select); c1Chart1.ChartGroups[0].ChartData.PointStylesList.Add( ps);


} void PS_Select( object sender, C1.Win.C1Chart.PointStyleSelectEventArgs e) {
C1.Win.C1Chart.PointStyle ps = sender as C1.Win.C1Chart.PointStyle;

C1.Win.C1Chart.ChartDataSeries ds = c1Chart1.ChartGroups[0].ChartData[e.SeriesIndex]; double y = Convert.ToDouble(
ds.Y[e.PointIndex]); if( y<0) ps.LineStyle.Color = Color.Blue; else ps.LineStyle.Color = Color.Red; e.Selected =
true;


}
```

Y 值小于 0 定制了蓝色的点类型，y 值大于 0 定制了红色的点类型。