保存和导出C1Chart

将图表导出为PDF格式为导出图表为位图图像，并使用C1Pdf库将该图像导出为PDF，请使用以下代码：

C#

```
// 保存图表图像至文件流
MemoryStream ms = new MemoryStream(); chart.SaveImage(ms, ImageFormat.Png);
// 从文件流创建图像实例
var img = System.Drawing.Image.FromStream(ms);
// 创建并保存PDF文档
C1PdfDocument pdf = new C1PdfDocument(); pdf.DrawImage( img, new System.Drawing.RectangleF(0,0,img.Width,img.Height));
pdf.Save("doc.pdf");
```

导出图表图像您可以通过RenderTargetBitmap 方法导出图表的图像，如以下代码所示：

Visual Basic

```
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="54e04e83-4119-43b5-bbad-fb2e4ad0d3b1"><ac:plain-text-body><![CDATA[
```

```
Dim bm As New
RenderTargetBitmap(CInt(c1Chart1.ActualWid
CInt(c1Chart1.ActualHeight), 96, 96,
PixelFormats.[Default]) bm.Render(c1Chart1
]]></ac:plain-text-body></ac:structured-ma
Dim enc As New PngBitmapEncoder()
enc.Frames.Add(BitmapFrame.Create(bm))
Dim fs As New FileStream("chart.png",
FileMode.Create) enc.Save(fs)
```

C#

```
RenderTargetBitmap bm = new RenderTargetBitmap(
(int)c1Chart1.ActualWidth,(int)c1Chart1.ActualHeight,
96, 96, PixelFormats.Default); bm.Render(c1Chart1);
PngBitmapEncoder enc = new PngBitmapEncoder(); enc.Frames.Add(BitmapFrame.Create(bm));

FileStream fs = new FileStream("chart.png", FileMode.Create); enc.Save(fs);
```

将C1Chart保存为 .png 文件为将C1Chart保存为 .png 文件，使用下面的代码：

Visual Basic

```
' 保存图像至文件实例
Using stm = System.IO.File.Create("chart.png")
c1Chart1.SaveImage(stm, C1.WPF.C1Chart.Extended.ImageFormat.Png)

End Using
```

C#

```
// 保存图像至文件实例
using (var stm = System.IO.File.Create("chart.png"))
{ c1Chart1.SaveImage(stm, C1.WPF.C1Chart.Extended.ImageFormat.Png);

}
```

生成系列

注意：本主题的示例可以在博文 "图表自动系列生成（MVVM）" 中找到。

对于使用MVVM的开发者，系列可以完全在ViewModel中通过两个ChartData对象属性生成：SeriesItemSource 以及SeriesItemTemplate。
在一个实际的场景中，您希望为每一年绘制一个不同的数据系列，但是不同年份的数量在设计时并不确定，则您可以在ViewModel中确定年份的数量。
首先我们要着重介绍一些在绑定属性至ViewModel的元素。在突出显示的XAML标记和代码之后，是包含全部必须的标记和代码的MVVM自动生成系列的主题。
在以下的XAML标记中，您可以但看到这两个属性：

XAML

```
<c1:C1Chart Name="c1Chart1">
<c1:C1Chart.Data>
<c1:ChartData SeriesItemsSource="{Binding SeriesDataCollection}">
<c1:ChartData.SeriesItemTemplate>
<DataTemplate>
<c1:DataSeries Label="{Binding Year}" ValuesSource="{Binding Values}" />

</DataTemplate>
</c1:ChartData.SeriesItemTemplate>
</c1:ChartData>
</c1:C1Chart.Data>
```

<c1:C1ChartLegend DockPanel.Dock="Right" /> </c1:C1Chart>
SeriesItemsSource 以及 SeriesItemTemplate 属性均在ChartData对象上设置。
SeriesItemTemplate. SeriesItemsSource绑定到ViewModel，同时包括SeriesItemTemplate的Label以及ValuesSource 属性。
重点介绍ViewModel的两个章节，第一个创建SeriesData的ObservableCollection：

C#

```
public ObservableCollection<SeriesData> SeriesDataCollection
{ get { if (_seriesDataCollection == null)
{
_seriesDataCollection = new ObservableCollection<SeriesData>(); for (int i = 0; i < ViewModelData.NUM_SERIES; i++)
_seriesDataCollection.Add(new SeriesData(2010 + i));
} return _seriesDataCollection;
}
}
```

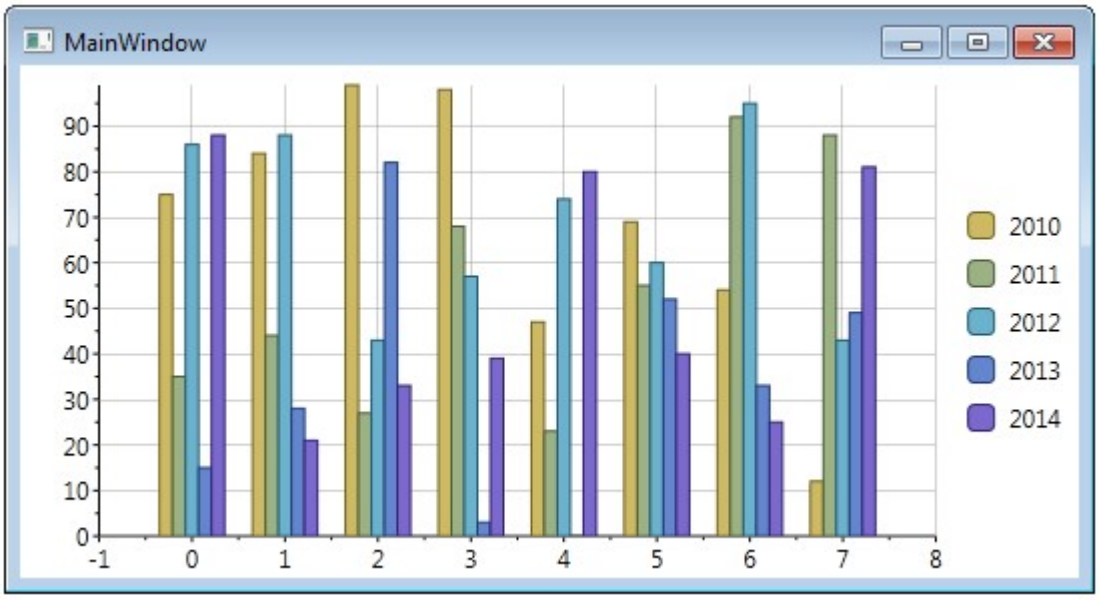
第二段代码是我们的自定义业务对象。它包含年份和数据值：

C#

```
public class SeriesData : INotifyPropertyChanged
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="567f63e4-7454-4601-95dd-153114380afa"><ac:plain-text-body><![CDATA[ { int _year; double[] _values;
]]></ac:plain-text-body></ac:structured-macro>
public SeriesData(int year)
{
_year = year;
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="e40ef625-7ed1-4ffe-9e8c-f8cb64fbd778"><ac:plain-text-body><![CDATA[ _values = new double[ViewModelData.NUM_PO
INTS]; for (int i = 0; i < ViewModelData.NUM_POINTS; i++)
]]></ac:plain-text-body></ac:structured-macro>
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="54b47048-6800-4d2c-ba46-6ecc72cc414d"><ac:plain-text-body><![CDATA[ _values[i] = ViewModelData.Rnd.Next(0,
100); }
]]></ac:plain-text-body></ac:structured-macro>
public int Year
{ get { return _year; } set { if (_year != value)
{
_year = value;
OnPropertyChanged("Year");
}
```

```
} }
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1"
ac:macro-id="a949278c-49e1-42fc-abce-ccafb821aff7"><ac:plain-text-body><![CDATA[ public double[] Values
]]></ac:plain-text-body></ac:structured-macro>
{ get { return _values; } set { if (_values != value)
{
_values = value;
OnPropertyChanged("Values");
}
}
}
```

当您运行您的程序，或运行该示例，它应该像以下图像所示：



通过MVVM自动生成序列
本主题假设您已经创建了一个新的Visual Studio工程，并为您的工程添加了适当的引用。
步骤步骤1）创建标记）创建标记
这是让您开工的XAML标记：

XAML

```
<c1:C1Chart Name="c1Chart1">
<c1:C1Chart.Data>
<c1:ChartData SeriesItemsSource="{Binding SeriesDataCollection}">
<c1:ChartData.SeriesItemTemplate>
<DataTemplate>
<c1:DataSeries Label="{Binding Year}" ValuesSource="{Binding Values}" />
</DataTemplate>
</c1:ChartData.SeriesItemTemplate>
</c1:ChartData>
</c1:C1Chart.Data>
<c1:C1ChartLegend DockPanel.Dock="Right" />
</c1:C1Chart>
```

请注意，SeriesItemsSource以及SeriesItemTemplate属性在ChartData 对象上进行设置，他们将绑定到具体的值。
步骤步骤2）创建视图模型）创建视图模型接下来，您需要创建您的项目视图。右键单击您的项目名称，选择添加新项|选择代码文件，将其命名为ViewModel.cs，并单击"确定"。
将下面的代码添加到您的代码文件中，以创建视图模型：

C#

```
using System; using System.Collections.Generic; using System.Linq; using System.Text; using System.ComponentModel; using S
ystem.Collections.ObjectModel;
namespace ChartAutomaticSeries
{ public static class ViewModelData
{ public static int NUM_SERIES = 5; public static int NUM_POINTS = 8; public static Random Rnd = new Random(); private sta
tic ChartModelData _data;
public static ChartModelData ChartData
{ get { if (_data == null)
{
_data = new ChartModelData();
} return _data;
}
}
```

```
}
<span style="color: #0000ff">public</span> <span style="color: #0000ff">class</span> ChartModelData
{ <span style="color: #0000ff">public</span> ObservableCollection<SeriesData> SeriesDataCollection
{ <span style="color: #0000ff">get</span> { <span style="color: #0000ff">if</span> (_seriesDataCollection == <span
style="color: #0000ff">null</span>)
{
_seriesDataCollection = <span style="color: #0000ff">new</span> ObservableCollection<SeriesData>(); <span style="color:
```

```
#0000ff">for</span> (<span style="color: #0000ff">int</span> i = 0; i < ViewModelData.NUM_SERIES; i++)
_seriesDataCollection.Add(<span style="color: #0000ff">new</span> SeriesData(2010 + i));
} <span style="color: #0000ff">return</span> _seriesDataCollection;
} } <span style="color: #0000ff">private</span> ObservableCollection<SeriesData> _seriesDataCollection;
}
<span style="color: #0000ff">public</span> <span style="color: #0000ff">class</span> SeriesData : INotifyPropertyChanged
{ <span style="color: #0000ff">int</span> _year; <span style="color: #0000ff">double</span>[] _values;
<span style="color: #0000ff">public</span> SeriesData(<span style="color: #0000ff">int</span> year)
{
_year = year;
_values = <span style="color: #0000ff">new</span> <span style="color: #0000ff">double</span>[ViewModelData.NUM_POINTS]; <span
style="color: #0000ff">for</span> (<span style="color: #0000ff">int</span> i = 0; i < ViewModelData.NUM_POINTS; i++)
_values[i] = ViewModelData.Rnd.Next(0, 100); }
<span style="color: #0000ff">public</span> <span style="color: #0000ff">int</span> Year
{ <span style="color: #0000ff">get</span> { <span style="color: #0000ff">return</span> _year; } <span style="color:
#0000ff">set</span> { <span style="color: #0000ff">if</span> (_year != value)
{
_year = value;
OnPropertyChanged(<span style="color: #800000">"Year"</span>);
}
} }
<span style="color: #0000ff">public</span> <span style="color: #0000ff">double</span>[] Values
{ <span style="color: #0000ff">get</span> { <span style="color: #0000ff">return</span> _values; } <span style="color:
#0000ff">set</span>
```

```
{ if (_values != value)
{
_values = value;
OnPropertyChanged("Values");
}
}
}
#region INotifyPropertyChanged Members public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{ if (PropertyChanged != null)
{
PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
}
}
#endregion
}
}
```

步骤步骤3）添加代码）添加代码
切换回您的MainWindow.xaml文件。右键单击"页面"，并从"上下文"菜单中选择"查看代码"。
编辑现有的代码，以便它类似于以下：

C#

```
public partial class MainWindow : Window

{ public MainWindow()
{ InitializeComponent();
this.DataContext = new ChartModelData();

}
}
}
```