FlexChart for UWP

2019.08.20 更新

グレープシティ株式会社

目次

概要

モダンな外観を持つ高性能な **FlexChart** および **FlexPie** for **UWP Edition** には、強力で柔軟なデータ連結と、簡単に使用できるチャート構成 API が装備されています。さらに、データ視覚化ニーズに合わせて、基本的なチャートから複合グラフまでのさまざまなチャートタイプを直接提供します。

コントロールの詳細については、次のリンクをクリックして、役に立つ総合的な情報にアクセスしてください。

- FlexChart
- FlexPie
- Sunburst
- FlexRadar
- TreeMap

FlexChart

FlexChart は強力なデータ視覚化コントロールです。ユニバーサル Windows アプリケーションに、機能豊富で見栄えのするチャートを追加できます。このコントロールを使用して、エンドユーザーはデータを読み手の心に響くように視覚化できます。

FlexChart コントロールは、多数の 2D チャートタイプ、チャート対話式操作のための組み込みツール、およびチャートレンダリングのための多様な書式を提供します。

データを使用した解説であれ複雑なデータの解釈であれ、FlexChart はすべてをシームレスに行うために役立ちます。

以下にリストするセクションで、十分に FlexChart コントロールについて理解し、使用を開始することができます。

主要な機能

FlexChart for UWP は、性能、表示、そして全体的な品質という点で、申し分のないデータ視覚化コンポーネントです。

- **凡例の自動生成**:系列の名前を指定するだけで、凡例が自動的に表示されます。
- **軸ラベルの自動回転**:長い軸ラベルを自動的に回転させることで、外観を見やすくレンダリングできます。
- **Axis Grouping**:Group the axis labels while working with any kind of data (categorical, numeric or date time) for better readability and analysis.
- **チャートのエクスポート**:アプリケーションを SVG.png、PNG などのさまざまな形式にエクスポートします。
- **Direct X のサポート**:コントロールは Direct X レンダリングエンジンをサポートしています。
- **柔軟なデータラベル**:Set offset, border, and position for data labels. The control also provides various options to manage overlapping of data labels such as automatic arrangement, hiding overlapped data labels and, rotation.
- **選択のサポート**:チャート内をクリックして、1 つのデータポイントまたはデータ系列全体を選択します。
- **null の補間**:**InterpolateNulls** プロパティを使用して、折れ線グラフと面グラフで null 値を効果的に処理します。
- **凡例の折り返し**:スペースの広さに応じて、凡例項目が複数の行と列で表示されるようにします。
- **複数のチャートタイプ**:1 つのグラフに必要なだけ系列を追加します。各系列に目的のチャートタイプを設定し、複数のチャートタイプを 1 つのチャートに統合します。
- **定義済みパレットとカスタムパレット**:多数の定義済みパレットから選択したり、カスタムパレットをチャートに適用します。
- **強力で柔軟なデータ連結**:必要に応じて、系列レベルまたはチャートレベルでデータソースを指定します。1 つのチャートで複数のデータソースを組み合わせることもできます。
- **系列の切り替え**:プロット内や凡例内の系列の表示/非表示を切り替えます。それには、**LegendToggle** プロパティを使用します。
- **使いやすさ**:オブジェクトモデルが明確なので、FlexChart コントロールは簡単に操作できます。
- **積層グラフまたは 100% 積層グラフ**:1 つのプロパティを設定するだけで、積層または 100% 積層グラフになります。
- **カテゴリ軸、数値軸、データ軸、時間軸のサポート**:int、float、string から DateTime まで、さまざまなデータ型に連結します。
- **ツールチップのカスタマイズ**:ツールチップの強力なカスタマイズ機能を利用します。

機能比較表

チャートタイプ

チャートタイプ	UWP	WPF	Win
面	○	○	○

積層面	○	○	○
100% 積層面	○	○	○
スプライン面	○	○	○
積層スプライン面	○	○	○
100% 積層スプライン面	○	○	○
StepArea	○	○	○
横棒	○	○	○
積層横棒	○	○	○
100% 積層横棒	○	○	○
バブル	○	○	○
ローソク足	○	○	○
縦棒	○	○	○
Cumulative Histogram	○	○	○
Cumulative Frequency Polygon	○	○	○
積層縦棒	○	○	○
100% 積層縦棒	○	○	○
Floating Bar	○	○	○
Histogram	○	○	○
ヒートマップ		○	○
Ranged Histogram	○	○	○
パレート図	○	○	○
ガント	○	○	○
Gaussian Curve	○	○	○
株価/Hi-Lo-Open-Close	○	○	○
折れ線	○	○	○
積層折れ線	○	○	○
100% 積層折れ線	○	○	○
スプライン	○	○	○
積層スプライン	○	○	○
100% 積層スプライン	○	○	○
折れ線シンボル	○	○	○
積層折れ線シンボル	○	○	○

100% 積層折れ線シンボル	○	○	○
スプラインシンボル	○	○	○
積層スプラインシンボル	○	○	○
100% 積層スプラインシンボル	○	○	○
Step	○	○	○
StepSymbols	○	○	○
円	○	○	○
ドーナツ円	○	○	○
分割円	○	○	○
分割ドーナツ円	○	○	○
Pareto	○	○	○
点/散布図	○	○	○
レーダー	○	○	○
極座標	○	○	○
箱ひげ図	○	○	○
誤差範囲	○	○	○
ファンネル	○	○	○
サンバースト	○	○	○
ウォーターフォール	○	○	○
2D	○	○	○
平均足	*	*	*
新値足/新値三本足	*	*	*
練行足	*	*	*
カギ足	*	*	*
カラムボリューム	*	*	*
エクイボリューム	*	*	*
ローソクボリューム	*	*	*
アームズローソクボリューム	*	*	*

* FinancialChart で使用可能

データ連結

データ連結	UWP	WPF	Win
IEnumerable を実装	○	○	○

するオブジェクト			

コア機能

コア機能	UWP	WPF	Win
空/Null のデータポイントの処理	○	○	○
ヒットテスト	○	○	○
注釈	○	○	○
レンダリングモード	ネイティブ/Direct3D	ネイティブ/Direct2D/Direct3D	ネイティブ/DirectX
傾向線	○	○	○
座標変換メソッド	○	○	○
バッチ更新	○	○	○
シリアライズのサポート		○	○
ラインマーカ	○	○	○
複数の軸	○	○	○
複数のプロット領域	○	○	○
Sunburst Drilldown	○	○	○
TreeMap Drilldown	○	○	○

チャート機能

機能	UWP	WPF	Win
注釈	○	○	○
軸の連結	○	○	○
カスタマイズ可能なデータラベル	○	○	○
カスタマイズ可能なヘッダーとフッター	○	○	○

データ操作

機能	UWP	WPF	Win
集約	○	○	○
ソート	○	○	○
TopN	○	○	○

外見

外見	UWP	WPF	Win
定義済みパレット	16	16	16

カスタムパレット	○	○	○
背景色	○	○	○
背景画像	○	○	○
背景グラデーション/ハッチスタイル	○	○	○
境界線および境界線スタイル	○	○	○
テーマ	○	○	○

財務チャート機能

財務チャート機能	UWP	WPF	Win
(ボリンジャーバンド)オーバーレイ	○	○	○
(エンベロープ)オーバーレイ	○	○	○
フィボナッチツール	○	○	○
インジケータ	○	○	○
(移動平均収束発散法)インジケータ	○	○	○
移動平均	○	○	○
オーバーレイ	○	○	○
(ストキャスティクス)インジケータ	○	○	○

チャート領域

チャート領域	UWP	WPF	Win
ヘッダー	○	○	○
フッター	○	○	○
ヘッダー/フッターの境界線	○	○	○
ヘッダー/フッターの配置	○	○	○
チャート領域の回転	○	○	○

プロット領域

プロット領域	UWP	WPF	Win
プロットマージン	○		
プロット要素のマーカー	FlexChart では、折れ線シンボル、スプラインシンボル、および散布図のチャートタイプでサポートされています。	FlexChart では、折れ線シンボル、スプラインシンボル、および散布図のチャートタイプでサポートされています。	FlexChart では、折れ線シンボル、スプラインシンボル、および散布図のチャートタイプでサポートされています。

マーカーのサイズ	○	○	○
マーカー:境界線および境界線スタイル設定	○	○	○
プロット要素の背景画像/グラデーション/ハッチスタイル	○	○	○

データラベル

データラベル	UWP	WPF	Win
オフセット	○	○	○
接続線	○	○	○
境界線および境界線スタイル設定	○	○	○
スタイル設定	○	○	○
Manage Overlapping	○	○	○
書式文字列	○	○	○
カスタムコンテンツ	○	○	○
デカルト座標グラフの位置	下/中央/左/なし/右/上	下/中央/左/なし/右/上	下/中央/左/なし/右/上
円グラフの位置	中央/内側/外側/なし	中央/内側/外側/なし	中央/内側/外側/なし

Annotations

Annotations	UWP	WPF	Win
Pre-defined Shapes	○	○	○
Position	○	○	○
Attaching Annotations	○	○	○
Offset	○	○	○
Styling	○	○	○
Tooltip	○	○	○
Customization	○	○	○

軸

軸	UWP	WPF	Win
軸:第 1 X/Y	○	○	○
軸:第 2 X/Y	○	○	○
軸:複数の第 2 X/Y	○	○	○
軸ラベル:書式文字	○	○	○

列			
軸ラベル:Hide	○	○	○
軸ラベル:スタイル設定	○	○	○
軸範囲(Min/Max)の値	○	○	○
軸:Hide	○	○	○
軸:対数	○	○	○
軸:逆転	○	○	○
軸線のスタイル設定	○	○	○
ラベル:配置	○	○	○
ラベル:角度	○	○	○
ラベル:重なっているラベルを非表示にする	○	○	○
主/副グリッド線	○	○	○
大/小目盛りマーク	○	○	○
ラベル単位/データ単位	○	○	○
タイトルおよびタイトルのスタイル設定	○	○	○
原点の設定	○	任意の値	任意の値
目盛りマークの位置	交差/内側/外側/なし	交差/内側/外側/なし	交差/内側/外側/なし
位置	上/下/左/右/自動/なし	上/下/左/右/自動/なし	上/下/左/右/自動/なし

系列

系列	UWP	WPF	Win
複数の系列	○	○	○
データ連結	○	○	○
系列レベルのチャートタイプ	○	○	○
スタイル設定	○	○	○
表示/非表示	プロット/凡例/プロットと凡例/非表示	プロット/凡例/プロットと凡例/非表示	プロット/凡例/プロットと凡例/非表示
条件付き書式設定	○	○	○

凡例

凡例	UWP	WPF	Win

タイトル	○	○	○
タイトルのスタイル	○	○	○
Custom Legend Icon	○	○	○
凡例での系列の表示/非表示の切り替え	○	○	○
方向	自動/垂直/水平	自動/垂直/水平	自動/垂直/水平
位置	左/上/右/下	左/上/右/下	左/上/右/下

マーカーシンボル

マーカーシンボル	UWP	WPF	Win
ボックス	○	○	○
ドット	○	○	○

ユーザー操作

ユーザー操作	UWP	WPF	Win
アニメーション	○	○	○
ツールチップ	○	○	○
ドリルダウン	○	○	○
系列選択	○	○	○
ポイント選択	○	○	○
ラインマーカー（十字線）	○	○	○
範囲セレクタ	○	○	○
ズーム	○	○	○
スクロール	○	○	○
軸スクロールバー	○	○	○

ツールチップ

ツールチップ	UWP	WPF	Win
自動ツールチップ	○	○	○
カスタムコンテンツ	○	○	○
遅延の表示	○	○	○
スタイル設定	○	○	○
さまざまなチャート要素のツールチップ			

円グラフ

円グラフ	UWP	WPF	Win
分割されたスライス	○	○	○
Inner Radius	○	○	○
最初のスライスの開始角度	○	○	○

エクスポート/インポートおよび印刷

エクスポート/インポートおよび印刷	UWP	WPF	Win
JPEG.png へのエクスポート	○	○	○
PNG へのエクスポート	○	○	○
SVG へのエクスポート			○
BMP へのエクスポート	○	○	
印刷のサポート	○	○	○

フットプリント

フットプリント	UWP	WPF	Win
アセンブリサイズ	218KB	183KB	229KB

クイックスタート

このクイックスタートでは、Visual Studio で単純な FlexChart アプリケーションを作成して実行する手順を説明します。

FlexChart コントロールをすばやく体験するには、以下の手順を実行します。

1. アプリケーションへの FlexChart の追加
2. データソースへの FlexChart の連結
3. アプリケーションの実行

手順 1:アプリケーションへの FlexChart の追加

この手順では、新しい Visual Studio アプリケーションを作成し、適切な参照をプロジェクトに追加します。さらに、FlexChart コントロールを作成するための XAML マークアップを追加します。

1. 新しい **ユニバーサル Windows** アプリケーションを作成します。
    1. **[ファイル]**→**[新規作成]**→**[プロジェクト]**を選択します。**[新しいプロジェクト]**ダイアログボックスが開きます。
    2. **[テンプレート]**→**[Visual C#]**→**[Windows]**→**[ユニバーサル]**を選択します。テンプレートリストで、**[空のアプリ(ユニバーサル Windows)]**を選択します。
    3. アプリケーションに名前を付け、**[OK]**をクリックします。新しいアプリケーションが作成されました。
2. **MainPage.xaml** ファイルを開き、**FlexChart** コントロールをドラッグアンドドロップします。次の参照がプロジェクトに追加されます。

- **C1.UWP.dll**
- **C1.UWP.DX.dll**
- **C1.UWP.FlexChart.dll**

参照が追加されない場合は、手動で追加してください。それには **[参照]** フォルダを **ソリューションエクスプローラー** で右クリックし、**[追加]→[新しい参照]**を選択します。

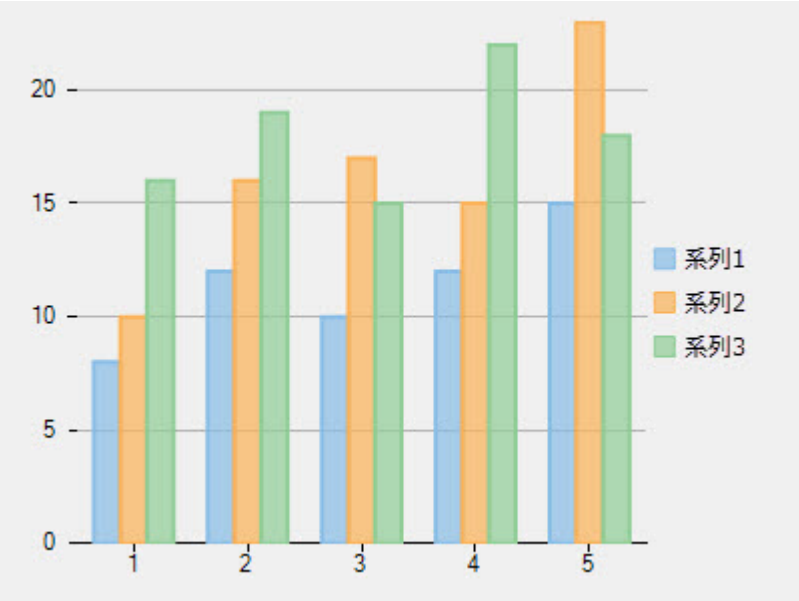
次のように、必要な名前空間とコントロールマークアップが **MainPage.xaml** に追加されます。

XAML

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:App1"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:Chart="using:C1.Xaml.Chart" xmlns:Xaml="using:C1.Xaml"
xmlns:Foundation="using:Windows.Foundation"
    x:Class="App1.MainPage"
    mc:Ignorable="d">
    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Chart:C1FlexChart x:Name="flexChart" HorizontalAlignment="Left"
Height="300" VerticalAlignment="Top" Width="300" Xaml:C1NagScreen.Nag="True">
   <Chart:Series Binding="Y" BindingX="X" SeriesName="系列1">
   <Chart:Series.ItemsSource>
  <PointCollection>
  <Foundation:Point>1,8</Foundation:Point>
  <Foundation:Point>2,12</Foundation:Point>
  <Foundation:Point>3,10</Foundation:Point>
  <Foundation:Point>4,12</Foundation:Point>
  <Foundation:Point>5,15</Foundation:Point>
  </PointCollection>
   </Chart:Series.ItemsSource>
   </Chart:Series>
   <Chart:Series Binding="Y" BindingX="X" SeriesName="系列2">
   <Chart:Series.ItemsSource>
  <PointCollection>
  <Foundation:Point>1,10</Foundation:Point>
  <Foundation:Point>2,16</Foundation:Point>
  <Foundation:Point>3,17</Foundation:Point>
  <Foundation:Point>4,15</Foundation:Point>
  <Foundation:Point>5,23</Foundation:Point>
  </PointCollection>
   </Chart:Series.ItemsSource>
   </Chart:Series>
   <Chart:Series Binding="Y" BindingX="X" SeriesName="系列3">
   <Chart:Series.ItemsSource>
  <PointCollection>
  <Foundation:Point>1,16</Foundation:Point>
  <Foundation:Point>2,19</Foundation:Point>
  <Foundation:Point>3,15</Foundation:Point>
  <Foundation:Point>4,22</Foundation:Point>
  <Foundation:Point>5,18</Foundation:Point>
  </PointCollection>
```

```
    </Chart:Series.ItemsSource>
    </Chart:Series>
     </Chart:C1FlexChart>
     </Grid>
</Page>
```

3. アプリケーションを実行します。FlexChart は次のように表示されます。



手順 2：データソースへの FlexChart の連結

この手順では、チャートをデータソースに連結します。

1. 次のようにデータソースを作成します。
    1. プロジェクトを右クリックし、**[追加]**→**[クラス]**を選択します。
    2. テンプレートの一覧から**[クラス]**を選択し、「**DataCreator.cs**」と名前を付け、**[追加]**をクリックします。
    3. 次のコードを **DataCreator.cs** クラスに追加してデータを生成します。

C# copyCode

```
namespace FlexChartUWPQuickStart
{
    class DataCreator
    {
         public static List<FruitDataItem> CreateFruit()
        {
           var fruits = new string[] { "蜜柑", "林檎", "梨", "バナナ" };
            var count = fruits.Length;
            var result = new List<FruitDataItem>();
            var rnd = new Random();

            for (var i = 0; i < count; i++)
                result.Add(new FruitDataItem()
                {
                    Fruit = fruits[i],
                    March = rnd.Next(20),
                    April = rnd.Next(20),
                    May = rnd.Next(20),
```

```
                Size=rnd.Next(5),
            });
        return result;
    }
}
public class FruitDataItem
    {
        public string Fruit { get; set; }
        public double March { get; set; }
        public double April { get; set; }
        public double May { get; set; }
    public int Size { get; set; }
    }
    public class DataPoint
    {
        public double XVals { get; set; }
        public double YVals { get; set; }
    }
}
```

2. 次のように、データをチャートに連結します。
    1. **MainPage.xaml** ファイルを開きます。次のマークアップを **<Page>** タグに追加して連結ソースを指定します。

XAML

```
DataContext="{Binding RelativeSource={RelativeSource Mode=Self}}"
```

    2. **<Grid>** タグを編集して次のマークアップを作成し、チャートにデータを提供します

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
Margin="-81,0,-361,-56">
        <Grid.RowDefinitions>
            <RowDefinition Height="93*"/>
            <RowDefinition Height="97*"/>
        </Grid.RowDefinitions>
        <Chart:C1FlexChart x:Name="flexChart" HorizontalAlignment="Left"
Width="443" ItemsSource="{Binding DataContext.Data}"
                BindingX="Fruit" Margin="198,18,0,235"
                Grid.RowSpan="2">
            <Chart:C1FlexChart.Series>
                <Chart:Series SeriesName="三月" Binding="March">
</Chart:Series>
                <Chart:Series SeriesName="四月" Binding="April">
</Chart:Series>
                <Chart:Series SeriesName="五月" Binding="May">
</Chart:Series>
            </Chart:C1FlexChart.Series>
            <Chart:C1FlexChart.AxisX>
                <Chart:Axis MajorGrid="False" Position="Bottom">
</Chart:Axis>
            </Chart:C1FlexChart.AxisX>
            <Chart:C1FlexChart.AxisY>
                <Chart:Axis AxisLine="False" Position="Left"
```

```
MajorUnit="5"></Chart:Axis>
            </Chart:C1FlexChart.AxisY>
            <Chart:C1FlexChart.SelectionStyle>
                <Chart:ChartStyle Stroke="Red"></Chart:ChartStyle>
            </Chart:C1FlexChart.SelectionStyle>
        </Chart:C1FlexChart>
    </Grid>
```

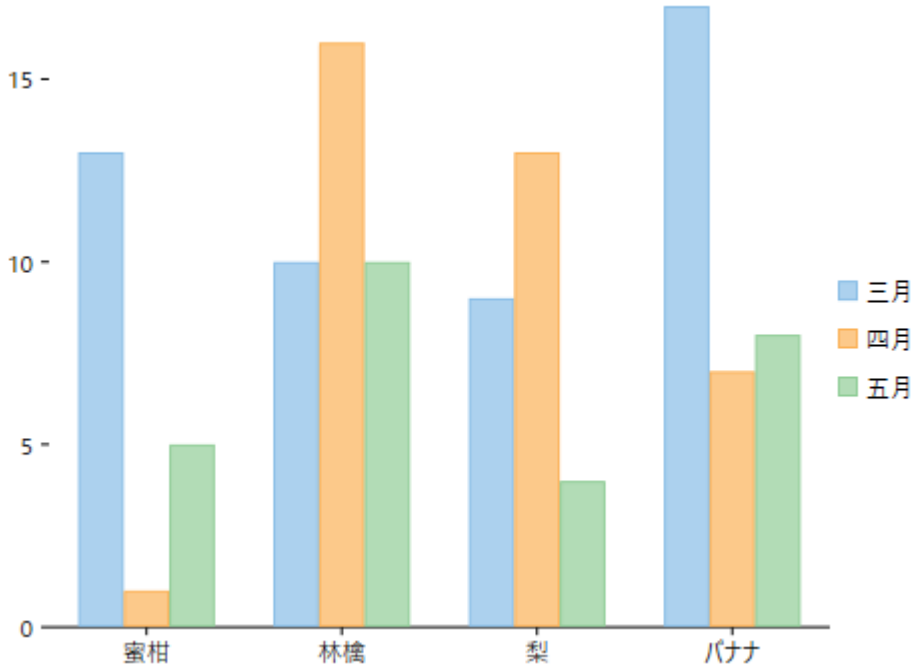
3. コードビュー（**MainPage.xaml.cs**）に切り替えます。次のコードを **MainPage** クラスに追加して、チャートにデータをプロットします。

MainPage.xaml.cs copyCode

```
List<FruitDataItem> _fruits;

public MainPage()
{
    this.InitializeComponent();
}

public List<FruitDataItem> Data
{
    get
    {
        if (_fruits == null)
        {
            _fruits = DataCreator.CreateFruit();
        }

        return _fruits;
    }
}
```

手順 3：アプリケーションの実行

ここまでで、UWP アプリケーションを作成し、アプリケーションの挙動をカスタマイズしました。次に、アプリケーションを実行します。**[F5]**キーを押してアプリケーションを実行します。FlexChart は次の図のように表示されます。



FlexChart の理解

FlexChart コントロールの使用を開始するには、FlexChart のすべての基本とタイプを十分に理解する必要があります。

以下のセクションは、コントロールの基本について説明します。

- FlexChart の基本
- FlexChart タイプ

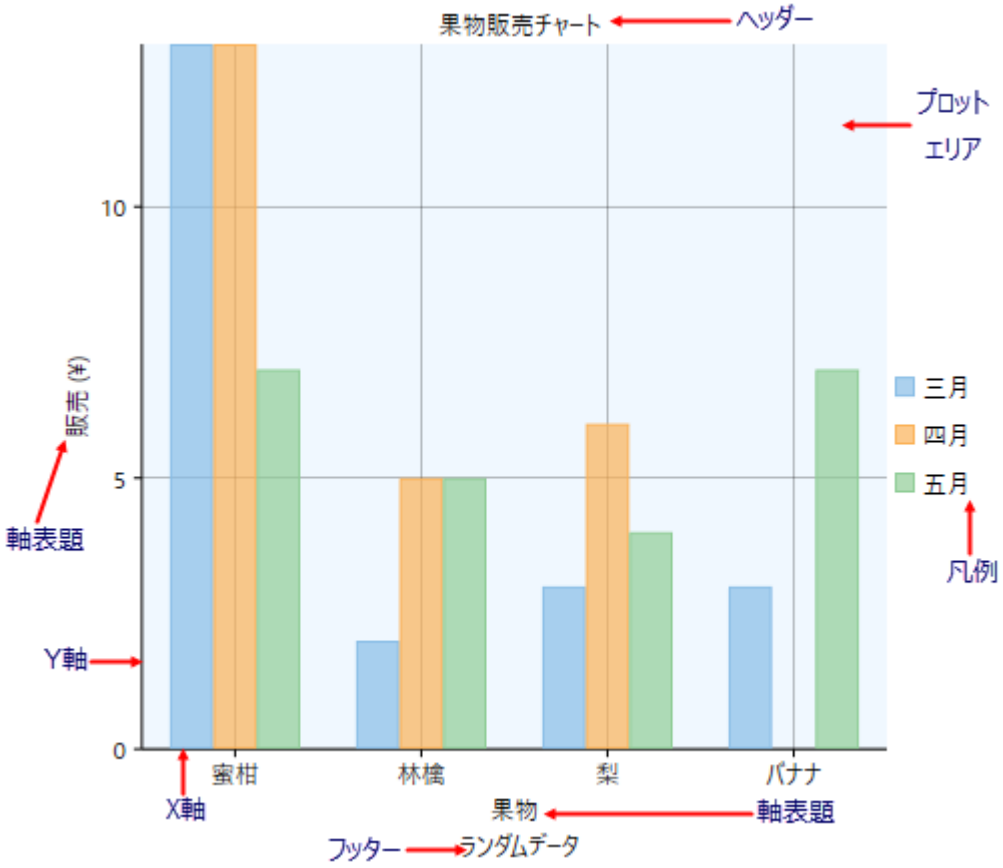
FlexChart の基本

FlexChart は次の要素で構成されます。

- ヘッダーとフッター
- 凡例
- 軸
- プロット領域
- 系列

コントロールは、これらの要素をオブジェクトによって表現し、その関連プロパティを提供する機能豊富なオブジェクトモデルを持ちます。

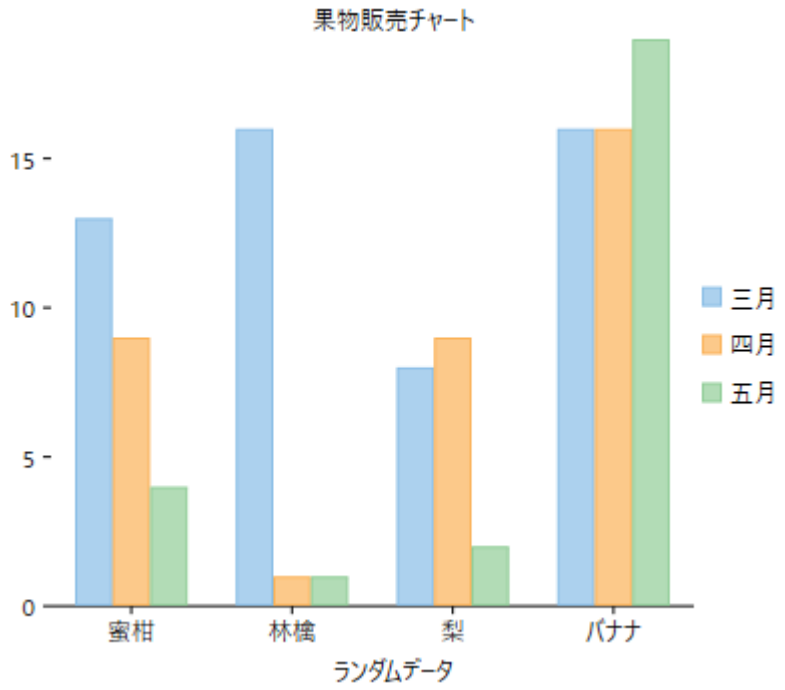
次の図に、さまざまな要素を示します。



以下のセクションでは、FlexChart のさまざまな基本要素について概説します。

ヘッダーとフッター

ヘッダーとフッターは、チャートに関する説明や関連情報を表示するために使用されます。



FlexChart では、これらの要素は **Header** プロパティと **Footer** プロパティを使用して設定されます。

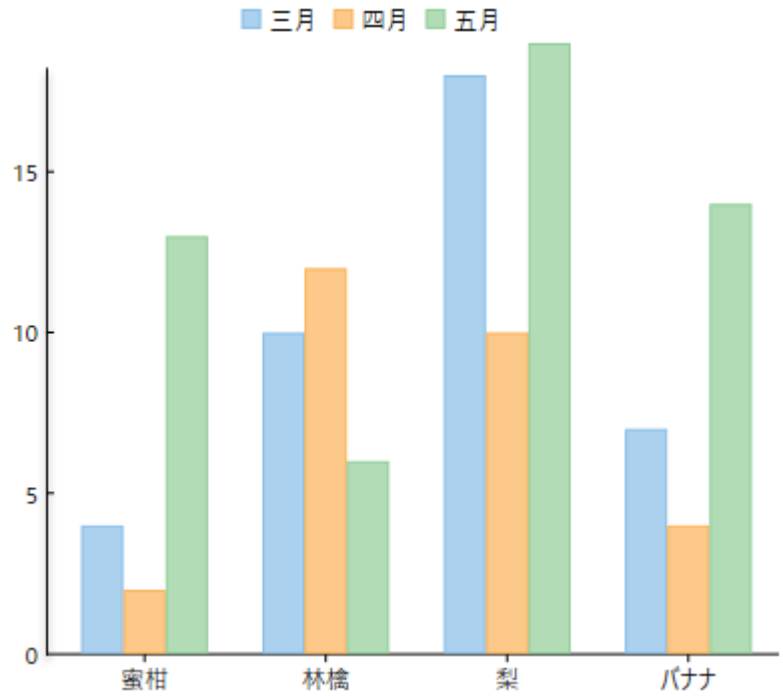
このプロパティを設定する Xaml は次のとおりです。

XAML

```
<Chart:C1FlexChart x:Name="flexChart" Header="果物販売チャート" Footer="ランダムデータ">
```

凡例

凡例は、各データ系列のエントリをチャート内に表示します。凡例には、色やシンボルとデータ系列の対応が表示されます。



FlexChart では、凡例は **FlexChartBase.Legend** プロパティを使用して設定されます。このプロパティには次のオプションがあります。

プロパティ	説明
LegendStyle	凡例のスタイルを設定するプロパティが含まれます。
LegendPosition	凡例の位置を決定します。

凡例の詳細については、「FlexChart の凡例」を参照してください。

凡例のスタイル

FlexChart では、**Chart.ChartStyle** クラスにある **LegendStyle** プロパティを使用して凡例をカスタマイズできます。

次の表に、凡例のカスタマイズに使用できるプロパティを示します。

プロパティ	説明
Fill	塗りつぶし色を設定します。
FontFamily	凡例のフォントを設定します。
FontSize	凡例のフォントのサイズを設定します。
FontStretch	フォントストレッチを設定します。
FontStyle	フォントスタイルを設定します。
FontWeight	フォントウェイトを設定します。
Stroke	ストローク色を設定します。
StrokeDashArray	ストロークの破線配列を設定します。
StrokeThickness	ストロークの太さを設定します。

このプロパティを設定する Xaml は次のとおりです。

XAML

```
<Chart:C1FlexChart.LegendStyle>
<Chart:ChartStyle FontFamily="Arial" FontStyle="Italic" Stroke="#FFC29EC4"/>
</Chart:C1FlexChart.LegendStyle>
```

凡例の位置

必要に応じて、**LegendPosition** プロパティを使用して、プロット領域を基準にして凡例を配置できます。

LegendPosition プロパティには、以下の値を設定できます。

プロパティ	説明
Auto	凡例を自動的に配置します。
Bottom	プロットの下に凡例を配置します。
Left	プロットの左に凡例を配置します。
None	凡例を非表示にします。
Right（デフォルト値）	プロットの右に凡例を配置します。
Top	プロットの上に凡例を配置します。

このプロパティを設定する Xaml は次のとおりです。

XAML

```
<Chart:C1FlexChart x:Name="flexChart" HorizontalAlignment="Left"
LegendPosition="Top">
```

凡例の切り替え

FlexChart クラスの **LegendToggle** プロパティを使用すると、凡例内の系列項目をクリックしたときにプロット内の系列の表示/非表示を切り替えることができます。**LegendToggle** プロパティのデフォルト値は False です。系列の切り替えを有効にするには、**LegendToggle** プロパティを True に設定する必要があります。

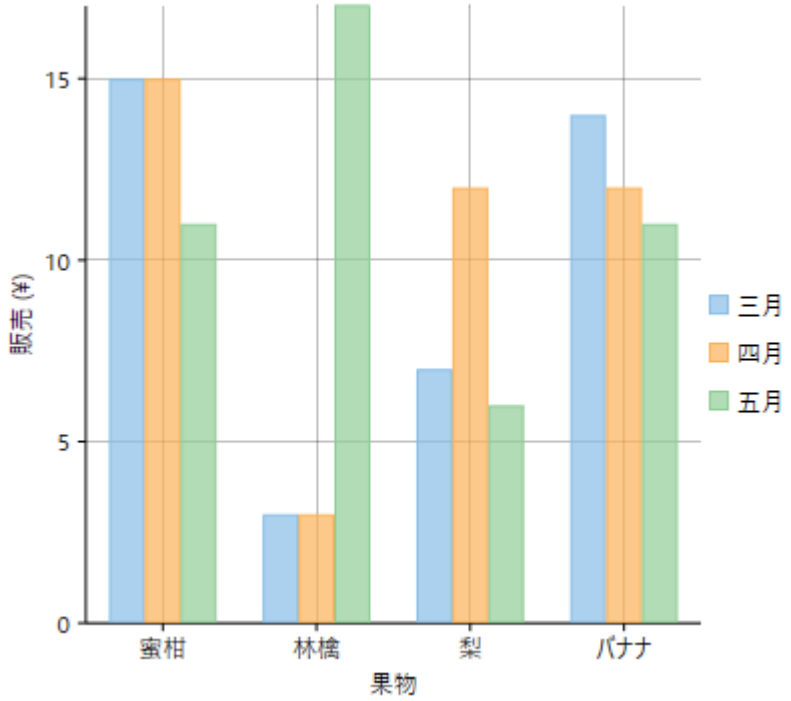
このプロパティを設定する Xaml は次のとおりです。

XAML

```
<Chart:C1FlexChart x:Name="flexChart" HorizontalAlignment="Left" LegendToggle="True">
```

軸

チャートには、X と Y という 2 つの主軸があります。もちろん、円グラフを使用する場合は例外です。



FlexChart の X 軸と Y 軸はそれぞれ **AxisX** プロパティと **AxisY** プロパティによって表現されます。どちらのプロパティも次のオプションを使用してカスタマイズできます。

レイアウトおよびスタイルのプロパティ

プロパティ	説明
AxisLine	軸線が表示されるかどうかを決定します。
Position	軸の位置を設定します。
Reversed	軸の方向を反転します。
Style	軸のスタイルを設定するプロパティが含まれます。
Title	軸の横に表示するタイトルテキストを設定します。

軸ラベルのプロパティ

プロパティ	説明
Format	軸ラベルの書式文字列を設定します。
LabelAlignment	軸ラベルの配置を設定します。
LabelAngle	軸ラベルの回転角度を設定します。
Labels	軸ラベルが表示されるかどうかを決定します。
MajorUnit	軸ラベル間の単位数を設定します。

Axis Grouping Properties

Property	Description
GroupNames	Sets the group name for the axis labels.
GroupItemsPath	Sets the group name for the axis labels in hierarchical data.

GroupSeparator	Set the axis group separator.
GroupProvider	Sets the axis group provider.

スケーリング、目盛りマーク、グリッド線のプロパティ

プロパティ	説明
MajorGrid	軸がグリッド線を含むかどうかを決定します。
MajorGridStyle	大目盛りマークに垂直に描画されるグリッド線の外観を制御するプロパティが含まれます。
MajorTickMarks	軸の目盛りマークの場所を設定します。
Max	軸の最大値を設定します。
Min	軸の最小値を設定します。
Origin	軸がその直交軸と交差する位置の値を設定します。
OverlappingLabels	何らかの理由でチャート内で重なっているラベルを管理します。

チャートの軸（Axis X および Axis Y プロパティ）を設定する Xaml は次のとおりです。

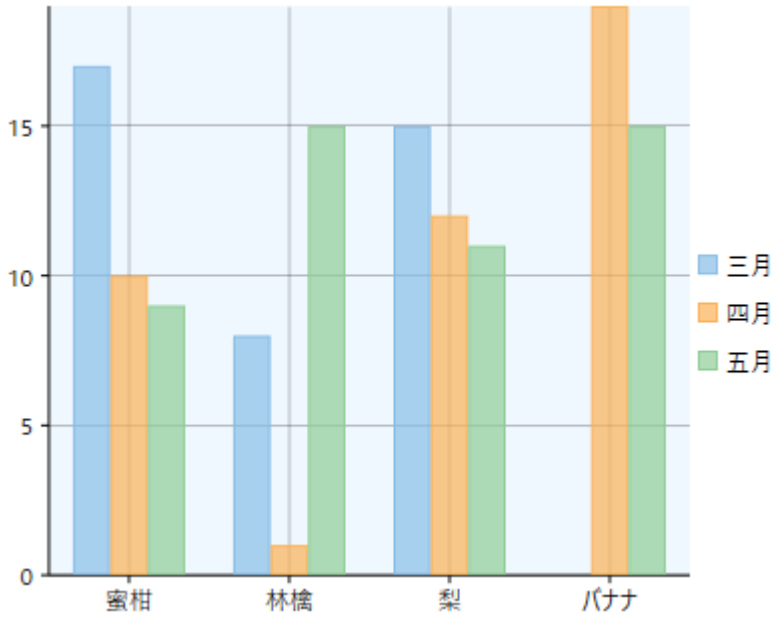
XAML

```
 <Chart:C1FlexChart.AxisX>
                <Chart:Axis MajorGrid="True" Position="Bottom" Title="果物"
MajorTickMarks="Inside"></Chart:Axis>
            </Chart:C1FlexChart.AxisX>
            <Chart:C1FlexChart.AxisY>
                <Chart:Axis Position="Left" MajorUnit="5" Title="販売 （¥）"
MajorGrid="True"/>
            </Chart:C1FlexChart.AxisY>
```

軸の詳細については、「FlexChart の軸」を参照してください。

プロット領域

プロット領域には、X 軸と Y 軸にプロットされるデータが含まれます。



FlexChart のプロット領域は、**Chart.ChartStyle** オブジェクトの次のプロパティによってカスタマイズできます。

次の表に、プロット領域のカスタマイズに使用できるプロパティを示します。

プロパティ	説明
Fill	プロット領域の色を設定します。
FontFamily	プロット領域のフォントを設定します。
FontSize	プロット領域のフォントサイズを設定します。
FontStretch	フォントストレッチを設定します。
FontStyle	フォントスタイルを設定します。
FontWeight	フォントウェイトを設定します。
Stroke	プロット領域のストロークブラシを設定します。
StrokeDashArray	プロット領域のストロークの破線配列を設定します。
StrokeThickness	プロット領域のストロークの太さを設定します。
LinePattern	プロット領域の線の点描のパターンを設定します。

チャートのプロット領域をカスタマイズする Xaml は次のとおりです。

XAML

```
<Chart:C1FlexChart.PlotStyle>
                <Chart:ChartStyle Fill="#FFF1F1F1"/>
</Chart:C1FlexChart.PlotStyle>
```

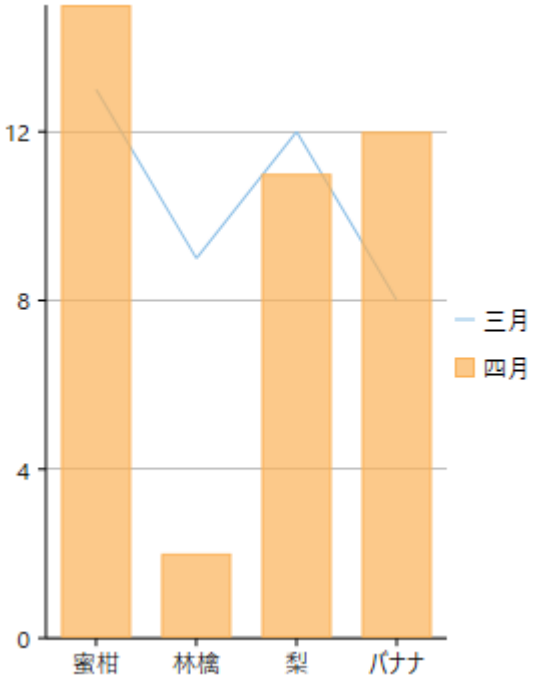
さらに、FlexChartを使用すると、複数のプロット領域を作成し、1つの系列を別のプロット領域に表示することでデータの可視性を高めることができます。

複数のプロット領域の詳細については、「複数のプロット領域」を参照してください。

系列

系列は、チャートのプロット領域内で、関連するデータポイントをグループ化します。データ系列ごとに異なるチャートタイプを設定できます。

次の図に、FlexChart のデータ系列の例を示します。



FlexChart のデータ系列は **Series** オブジェクトによって制御されます。このオブジェクトは次のプロパティで構成されます。

プロパティ	説明
AxisX	系列の X 軸を設定します。
AxisY	系列の Y 軸を設定します。
Binding	系列の Y 値を含むプロパティの名前を設定します。
BindingX	系列の X 値を含むプロパティの名前を設定します。
ChartType	系列のチャートタイプを設定します。
ItemSource	系列データを含むオブジェクトのコレクションを設定します。
SeriesName	凡例に表示される、系列のテキストを設定します。
Style	系列のスタイルを設定します。
SymbolMarker	系列の各データポイントで使用されるマーカーの形を設定します。このプロパティは、Scatter、LineSymbols、および SplineSymbols チャートタイプだけに適用されます。
SymbolSize	この系列のレンダリングに使用されるシンボルのサイズを設定します。
SymbolStyle	この系列で使用されるシンボルのスタイルを設定します。
Visibility	系列を表示するかどうかを決定し、表示する場合はその位置を設定します。

チャート系列のプロパティの設定をカスタマイズする Xaml は次のとおりです。

サンプルのタイトル
`<Chart:C1FlexChart.Series>`

```
            <Chart:Series SeriesName="三月" Binding="March" ChartType="Line"/>
            <Chart:Series SeriesName="四月" Binding="April"/>
            </Chart:C1FlexChart.Series>
```

プロパティウィンドウの **Series コレクションエディタ**からチャート系列のプロパティを設定することもできます。

系列の詳細については、「FlexChart の系列」を参照してください。

FlexChart タイプ

FlexChart for UWP は、エンドユーザーのあらゆるデータ視覚化要件を満たす包括的なチャートタイプを提供しています。

以下は、このコントロールが提供するすべてのチャートタイプのリストです。アプリケーションで使用するチャートタイプに基づいて、対応するリンクをクリックすると、主要な情報が表示されます。

- 面
- 横棒
- バブル
- 縦棒
- フローティングバー
- ファンネル
- ヒストグラム
- 折れ線
- 株価チャート（財務チャート）
- 折れ線シンボル
- 複合
- パレート図
- RangedHistogram
- 散布図
- スプライン
- スプライン面
- スプラインシンボル
- 階段グラフ

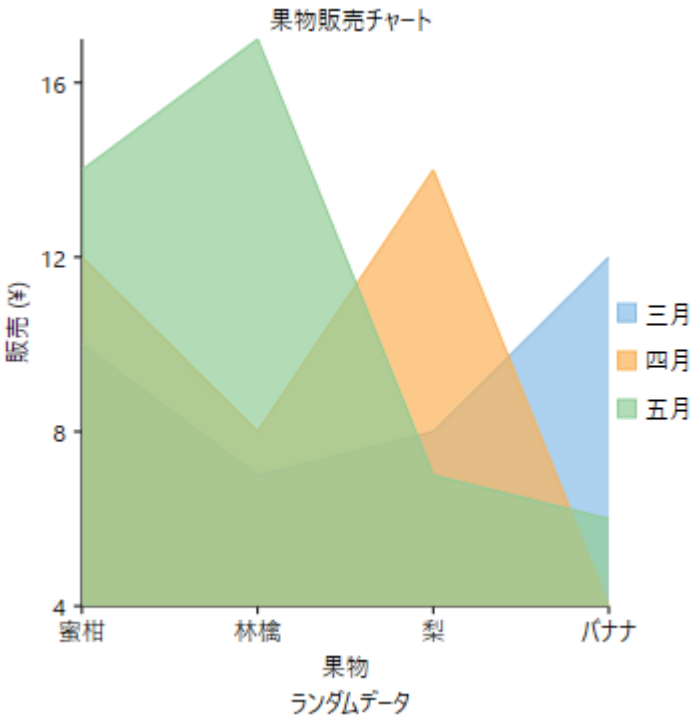
面

面グラフは、一定期間のデータの変化を表します。Y 軸のデータポイント間を接続し、系列と X 軸の間の領域を埋めることで、データ系列を表現します。さらに、データ系列は、追加されたときと同じ順番で背面から前面に表示されます。

面グラフを作成するには、**ChartType** プロパティを **Area** に設定する必要があります。

Stacking プロパティを **Stacked** または **Stacked100pc** に設定すると、積層面グラフを作成できます。

次の図は、面グラフを示します。



上のチャートは、プロットされた 3 か月間の株価によって株価が上昇していることを示しています。3 つの株価の 3 つの領域が 3 色でレンダリングされています。

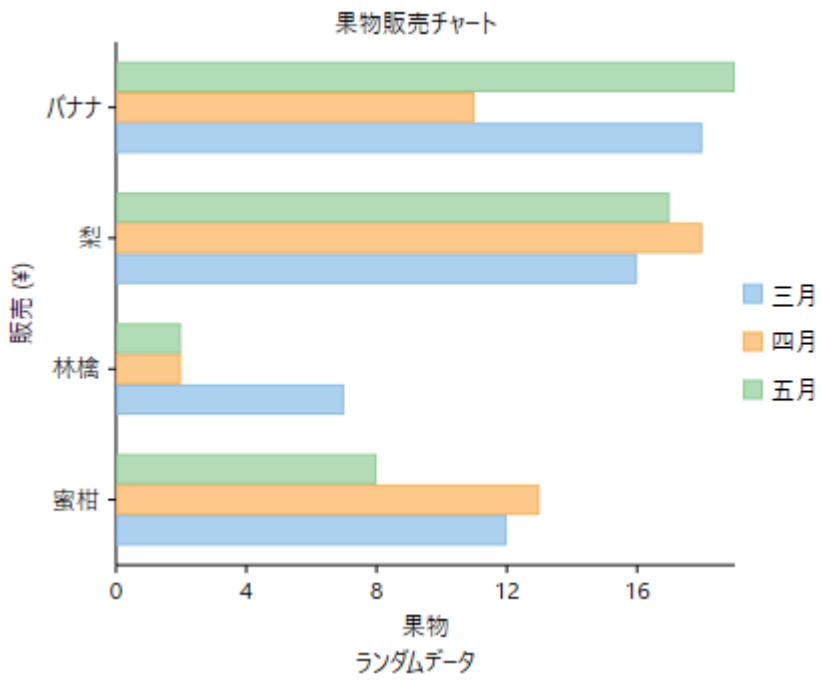
横棒

横棒グラフは、さまざまなカテゴリの値を比較したり、データ系列内の経時的な変動を表示します。このチャートタイプは、X 軸にプロットされるデータ系列を横棒として表示し、カテゴリまたは項目を Y 軸に配置します。

横棒グラフを作成するには、**ChartType** プロパティを **Bar** に設定する必要があります。

積層横棒グラフを作成するには、**Stacking** プロパティを **Stacked** または **Stacked100pc** に設定する必要があります。

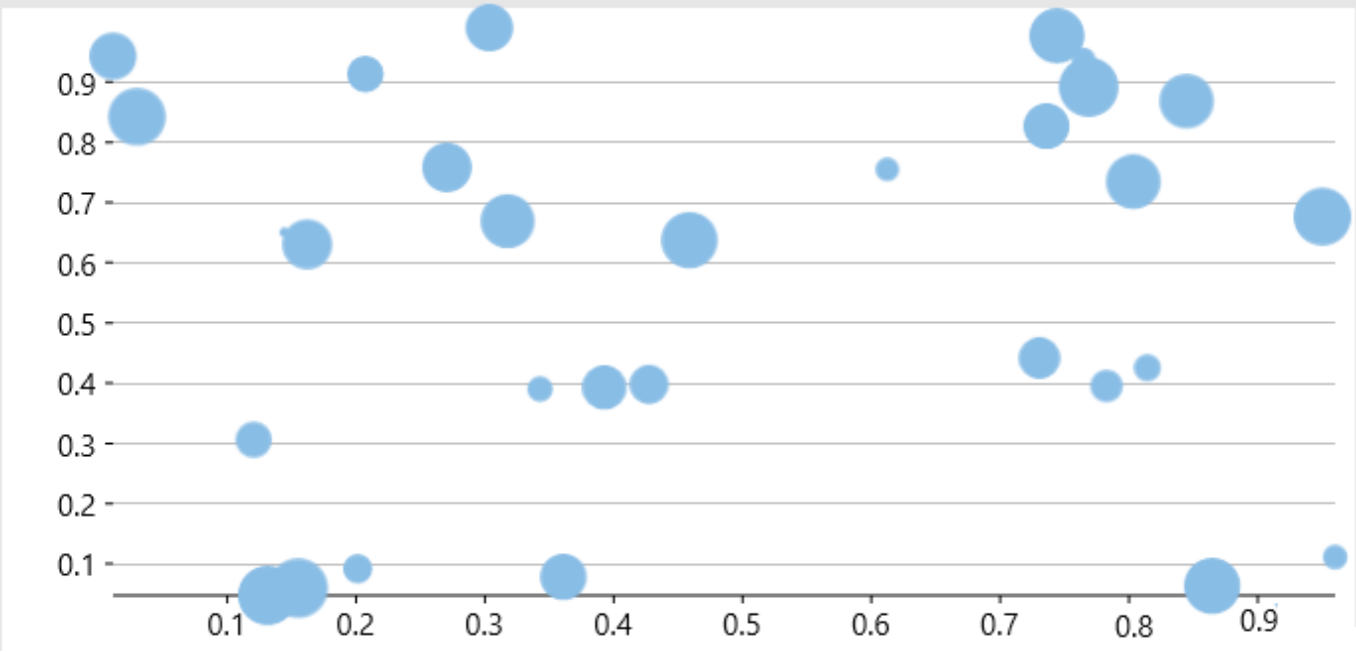
次の図は、横棒グラフを示します。



バブル

バブルチャートは、基本的に散布図グラフの 1 つのタイプですが、多次元データのグラフィカル表現に使用されます。各ポイントの付加的なデータ値をポイントのサイズを変えて表示します。チャートタイプは、データポイントをバブル（データマーカー）の形式で表現し、その X 座標と Y 座標が 2 つのデータ値で決定され、サイズが 3 番目の変数の値を示します。

したがって、X と Y のほかに、バブルサイズに対応する連結を指定する必要があります。バブルサイズに対応する連結を指定するには、binding プロパティを、各バブルの Y 値とサイズ値に使用するプロパティの名前を指定するカンマ区切りの文字列に設定します。
この例では、チャートは、「x」プロパティ、「y」プロパティ、および「size」プロパティから成るオブジェクトを含むリストに連結されます。このチャートには 1 つの系列が含まれ、その binding プロパティは文字列「**y,size**」に設定されています。



バブルチャートを作成するには、**ChartType** プロパティを **Bubble** に設定する必要があります。

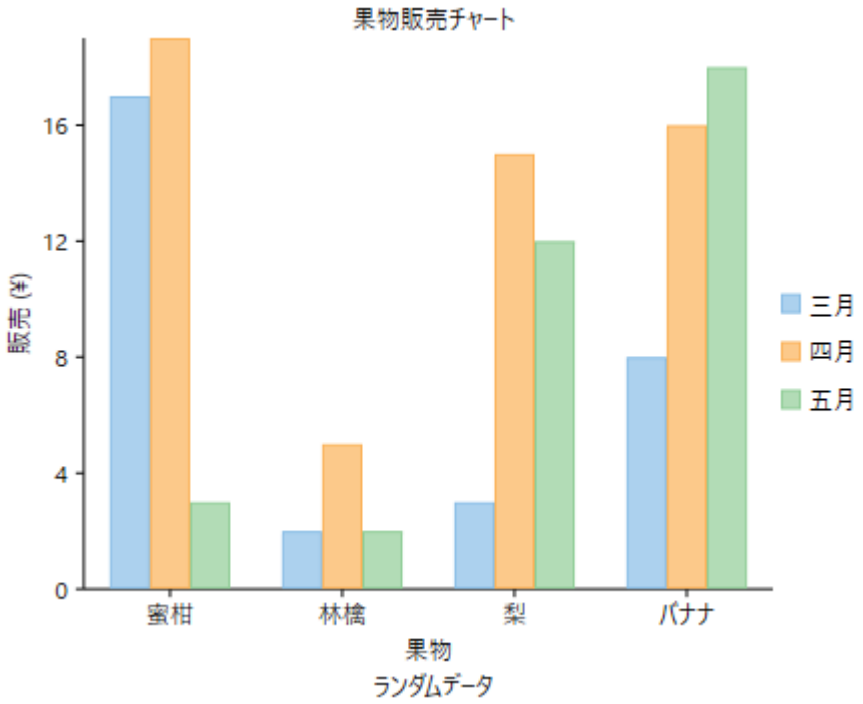
XAML

```
<Chart:C1FlexChart ItemsSource="{Binding}" ChartType="Bubble"  BindingX="X"
Binding="Y,Size" Xaml:C1NagScreen.Nag="True" Margin="0,10,-83,0">
</Chart:C1FlexChart>
```

縦棒

縦棒グラフは、横棒グラフと同様に、データ系列の経時的な変動を表現したり、異なる項目を比較します。1 つ以上の項目の値を縦棒として Y 軸に表示し、項目やカテゴリを X 軸に配置します。

縦棒グラフを作成するには、**ChartType** プロパティを **Column** に設定する必要があります。



株価(財務)

株価チャート(財務チャート)は市場価格や株価の変動を表現するために使用されますが、科学的データを表現するためにも使用できます。

FlexChart コントロールは、**Candle Chart** と **HighLowOpenClose Chart** という 2 つのタイプの株価チャートをサポートします。

これらのチャートタイプを使用するには、**ChartType** プロパティを **Candlestick** または **HighLowOpenClose** に設定する必要があります。

以下のセクションでは、この 2 つのタイプの株価チャートについて説明します。

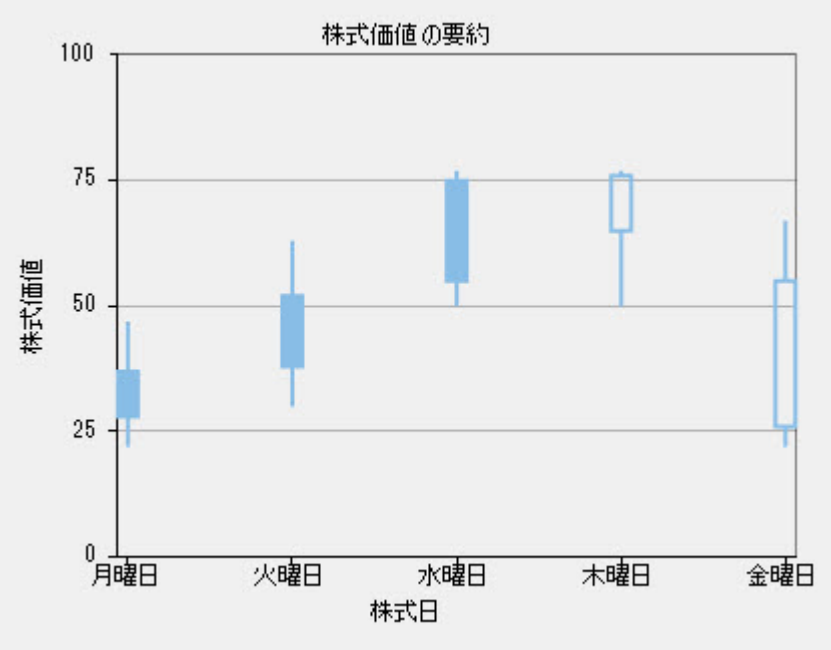
- ローソク足
- HighLowOpenClose

ローソク足

ローソク足チャートは、経時的な値の範囲を表すために横棒グラフと折れ線グラフを組み合わせて使用します。ローソクと呼ば

れるビジュアル要素で構成され、ローソクは胴体、上ヒゲ、下ヒゲの 3 つの要素から成ります。

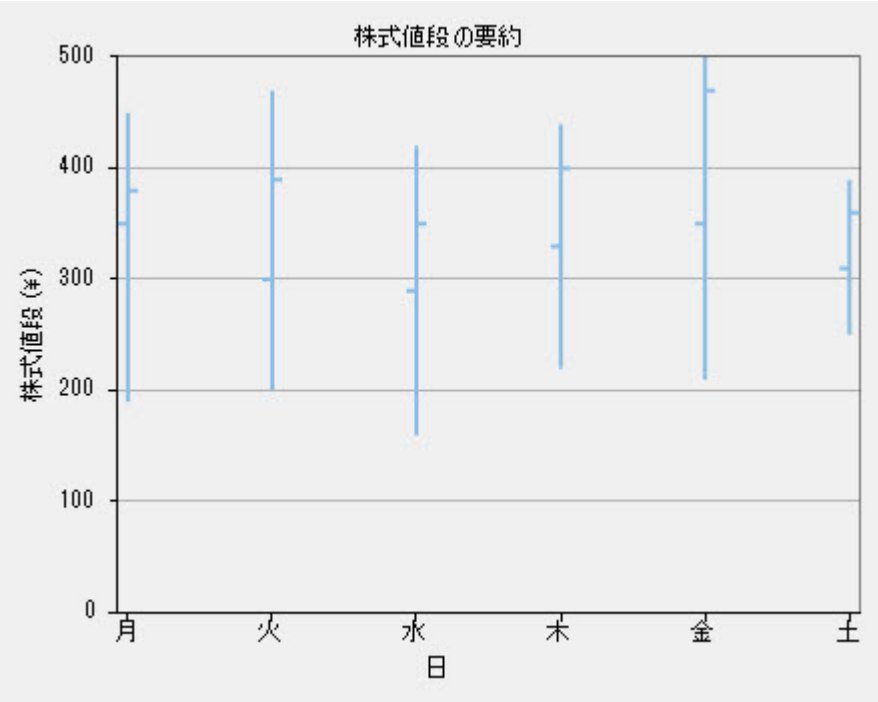
- 胴体は始値と終値を表現し、上ヒゲと下ヒゲはそれぞれ高値と安値を表現します。
- 白抜きの胴体は、株価が上昇したことを示します（終値が始値より高い）。
- 黒塗りの胴体は、株価が下降したことを示します（始値が終値より高い）。



ローソク足チャートは株価の概要を表現するために適しています。

HighLowOpenClose

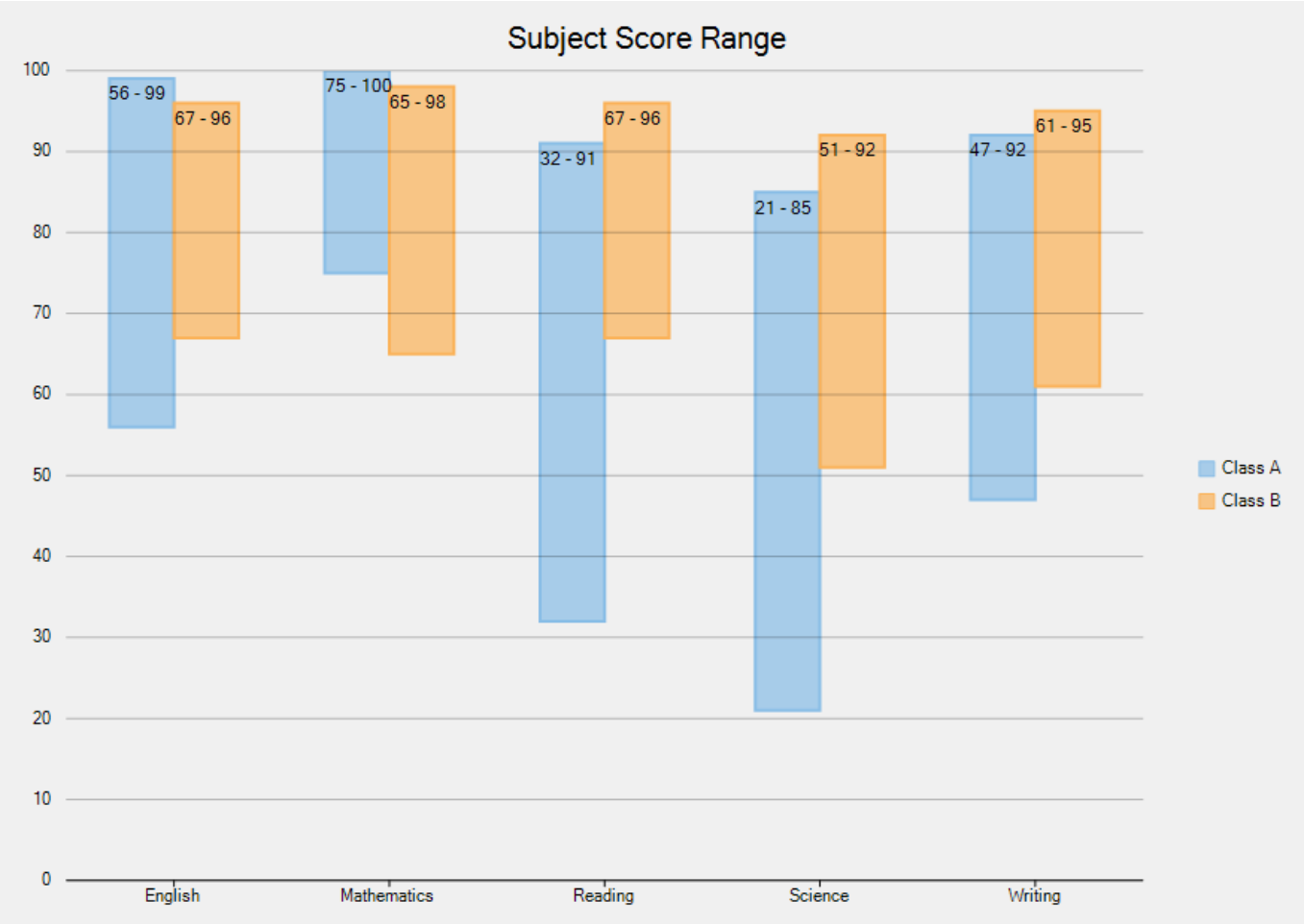
HighLowOpenClose チャートは一般に株価分析に使用されます。このチャートは 4 つの独立した値を組み合わせて、高値、安値、始値、終値のデータ値を系列内のデータポイントごとに提供します。



フローティングバー

Floating bar chart provides an interesting way to represent data in charts. In this type of chart, a single or multiple bars apparently floats between a minimum and maximum value instead of being connected to the axis. It displays information as a range of data by plotting two Y-values(low and high) per data point. The Y-axis shows the values, and the X-axis shows the category they belong to. Floating bars can be useful to show highs and lows in a data set, such as daily high and low temperatures, stock prices, blood pressure readings, etc.

In FlexChart, Floating bar chart can be implemented using the **Series** class. To begin with, create a new Series object and specify its properties. Then, use the **SymbolRendering** event provided by the Series class to plot the data points on the chart.



To implement Floating bar chart using the FlexChart control, see **FloatingBarChart** sample. The samples are available in the default installation folder - *Documents\ComponentOne Samples*

ファンネル

ファンネルグラフを使用すると、1 次元プロセスの連続的な段階を表現できます。たとえば、販売見込み客、有望見込み客、売買交渉、成約などの売買プロセスの段階を通して見込み客を追跡するとします。

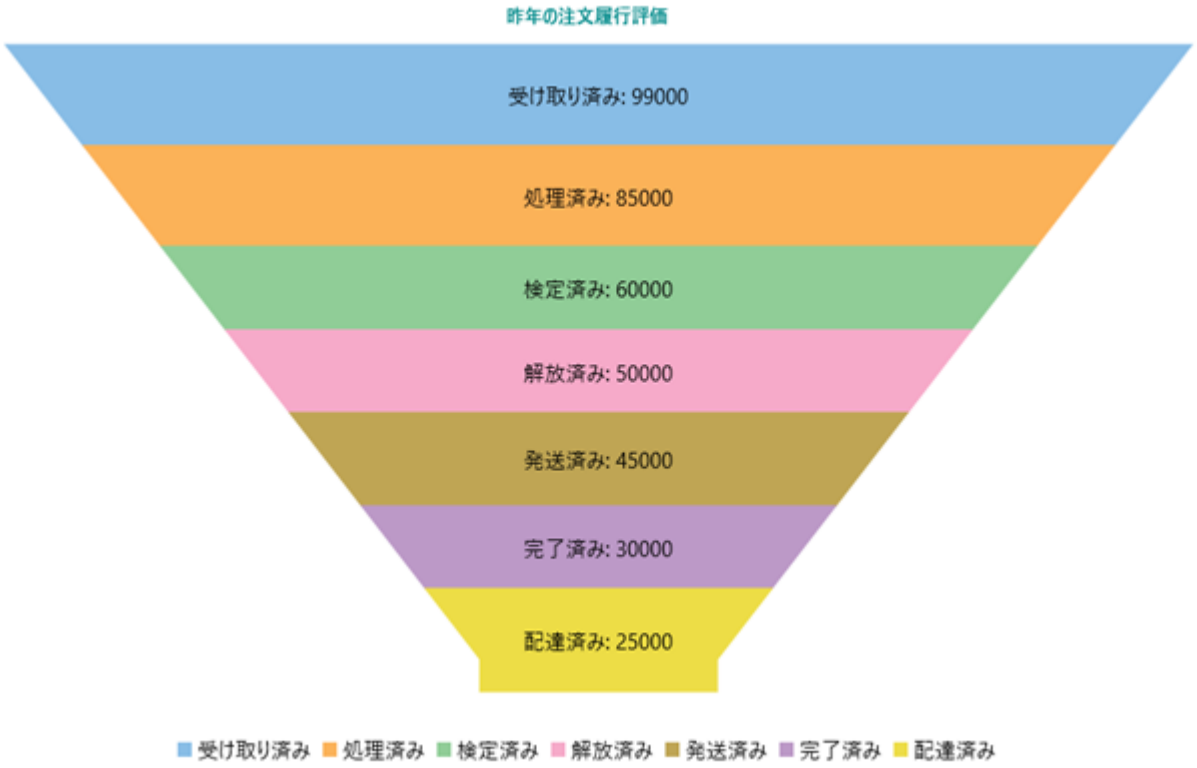
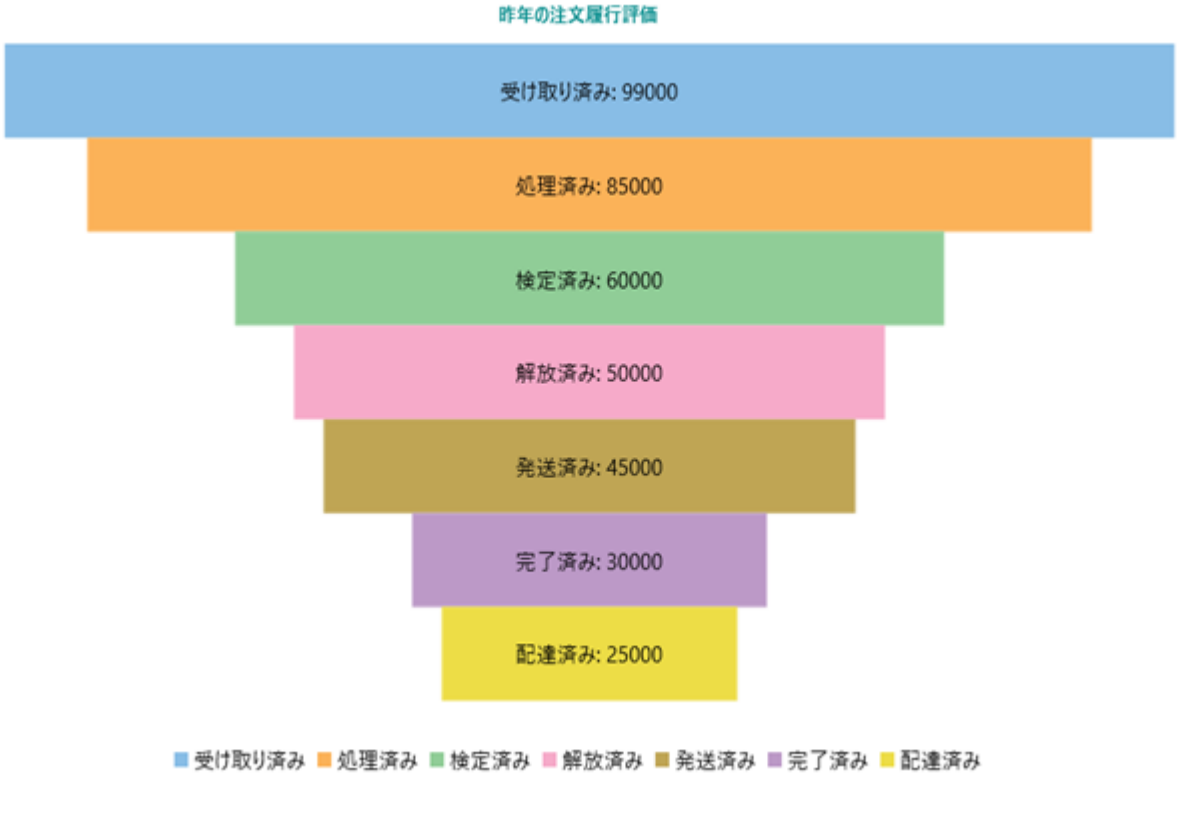
このプロセスの各段階で、全体に対する割合（パーセンテージ）を表します。そのため、チャートは、最初に最大の段階があり、段階が進むごとに小さくなる漏斗の形状になります。

ファンネルグラフは、どの段階にどの程度の割合で値が減少しているかが顕著にわかる場所として、プロセス内に潜在的な問題がある領域を特定します。

FlexChart には、次の 2 つの形式のファンネルグラフが用意されています。

- **台形グラフ**:平行な 2 辺を含みます。
- **積層横棒グラフ**:水平の棒の形式で、関連する値を互いの値の上に配置します。

次の図は、受注処理評価プロセスの 7 つの段階の注文数を示す台形グラフと積層横棒グラフを示します。

台形グラフ	
積層横棒グラフ	

FlexChart の **ChartType** プロパティを **ChartType** 列挙の **Funnel** に設定して、ファンネルグラフを使用します。**FunnelType** プロパティを **FunnelChartType** 列挙の Default または Rectangle に設定して、ファンネルグラフのタイプを台形グラフと積層横棒グラフのいずれかに指定します。

さらに、台形グラフに設定されている場合は、**FunnelNeckWidth** および **FunnelNeckHeight** プロパティを設定して、ファンネルグラフのネックのサイズを変更します。これらのプロパティは、**FlexChart** クラスの **Options** プロパティからアクセスできる **ChartOptions** クラスにあります。

次のコードは、受注処理の7つの段階にわたる受注金額の値を含むデータを作成するクラス**DataCreator**を作成します。

- **Visual Basic**

```
Class DataCreator
    Public Shared Function CreateFunnelData() As List(Of DataItem)
        Dim data = New List(Of DataItem)()
        data.Add(New DataItem("受け取り済み", 99000))
        data.Add(New DataItem("処理済み", 85000))
        data.Add(New DataItem("検定済み", 60000))
        data.Add(New DataItem("解放済み", 50000))
        data.Add(New DataItem("発送済み", 45000))
        data.Add(New DataItem("完了済み", 30000))
        data.Add(New DataItem("配達済み", 25000))
        Return data
    End Function
End Class

Public Class DataItem
    Public Sub New(order__1 As String, value__2 As Integer)
        Order = order__1
        Value = value__2
    End Sub

    Public Property Order() As String
        Get
            Return m_Order
        End Get
        Set
            m_Order = Value
        End Set
    End Property
    Private m_Order As String
    Public Property Value() As Integer
        Get
            Return m_Value
        End Get
        Set
            m_Value = Value
        End Set
    End Property
    Private m_Value As Integer
End Class
```

- **C#**

```
class DataCreator
{
    public static List<DataItem> CreateFunnelData()
    {
        var data = new List<DataItem>();
        data.Add(new DataItem("受け取り済み", 99000));
        data.Add(new DataItem("処理済み", 85000));
```

```
            data.Add(new DataItem("検定済み", 60000));
            data.Add(new DataItem("解放済み", 50000));
            data.Add(new DataItem("発送済み", 45000));
            data.Add(new DataItem("完了済み", 30000));
            data.Add(new DataItem("配達済み", 25000));
            return data;
        }
    }

    public class DataItem
    {
        public DataItem(string order, int value)
        {
            Order = order;
            Value = value;
        }

        public string Order { get; set; }
        public int Value { get; set; }
    }
```

次のスニペットは、チャートタイプを Funnel に設定し、ファンネルのネックのサイズを指定し、チャートのヘッダー、凡例、データラベルを設定します。

XAML

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:FunnelChart"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:Chart="using:C1.Xaml.Chart"
    xmlns:Foundation="using:Windows.Foundation"
    x:Class="FunnelChart.MainPage"
    DataContext="{Binding RelativeSource={RelativeSource Mode=Self}}"
    mc:Ignorable="d" Width="972.674"
    Height="708.449">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="29*"/>
            <ColumnDefinition Width="34*"/>
            <ColumnDefinition Width="297*"/>
            <ColumnDefinition Width="0*"/>
        </Grid.ColumnDefinitions>

        <Chart:C1FlexChart x:Name="flexChart"
                        BindingX ="Order"
                        ChartType="Funnel"
                        ItemsSource="{Binding Data}"
                        HorizontalAlignment="Left"
                        Height="529"
                        Margin="-146.333,179,0,0"
                        VerticalAlignment="Top"
                        Width="939"
                        Header="昨年の注文履行評価"
                        HeaderAlignment="Center"
                        LegendPosition="Bottom" Grid.Column="2">
            <Chart:C1FlexChart.HeaderStyle>
                <Chart:ChartStyle FontFamily="Arial"
                                FontSize="13"
```

```
                                FontWeight="Bold"
                                Stroke="DarkCyan"/>
            </Chart:C1FlexChart.HeaderStyle>
            <Chart:Series Binding="Value">
            </Chart:Series>
            <Chart:C1FlexChart.DataLabel>
                <Chart:DataLabel Content="{}{Order}: {y}"
                                 Position="Center"/>
            </Chart:C1FlexChart.DataLabel>
            <Chart:C1FlexChart.Options>
                <Chart:ChartOptions FunnelType="Default"
                                    FunnelNeckHeight="0.05"
                                    FunnelNeckWidth="0.2"/>
            </Chart:C1FlexChart.Options>
        </Chart:C1FlexChart>
    </Grid>

</Page>
```

Code

MainPage.xaml.vb copyCode

```
Partial Public Class MainPage
    Inherits Page
    Private _data As List(Of DataItem)

    Public Sub New()
        InitializeComponent()
    End Sub

    Public ReadOnly Property Data() As List(Of DataItem)
        Get
            If _data Is Nothing Then
                _data = DataCreator.CreateFunnelData()
            End If

            Return _data
        End Get
    End Property
End Class
```

MainPage.xaml.cs copyCode

```
public partial class MainPage : Page
{
    private List<DataItem> _data;

    public MainPage()
    {
        InitializeComponent();
    }

    public List<DataItem> Data
    {
        get
```

```
        {
            if (_data == null)
            {
                _data = DataCreator.CreateFunnelData();
            }

            return _data;
        }
    }
}
```

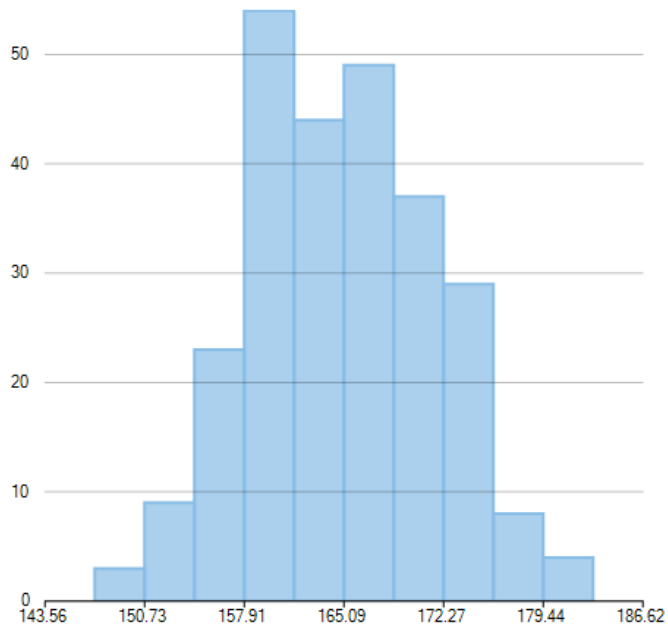
ヒストグラム

ヒストグラムは、定義されているクラスの間隔(ビン)に対するデータの度数分布をプロットします。これらのビンは、生データ値を重複のない連続した間隔に分割することで作成されます。特定のビンに入る値の数に基づき、連続的な X 軸に沿った長方形の柱として度数がプロットされます。

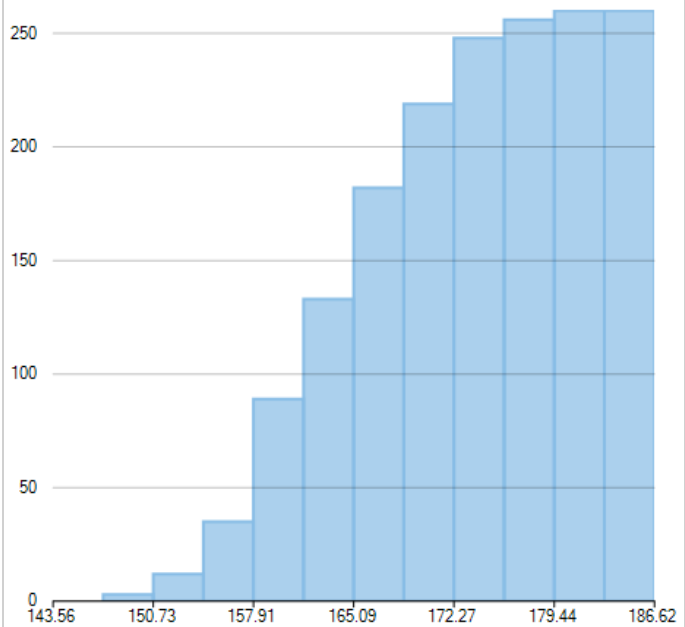
The following representations can created with the help of a histogram.

- **Frequency Polygon**
- **Gaussian Curve**

The following images show a histogram and a cumulative histogram created using FlexChart.



Histogram



Cumulative Histogram

ヒストグラムを作成するには、**Histogram** 系列を追加し、**ChartType** プロパティを **Histogram** に設定する必要があります。

一度、BindingXをX軸にプロットされる元の生データ値に設定して関連するデータを指定すると、FlexChartはデータの度数分布を生成し、ヒストグラムにプロットします。

関連するデータを指定すると、チャートは、データをグループ化する間隔を自動的に計算します。ただし、必要に応じて、**BinWidth** プロパティを設定することで、この間隔の幅を指定することもできます。

次のコードスニペットに示すように、FlexChart でデータの度数分布を生成し、ヒストグラムにプロットし、関連するデータを提供します。

- **Xaml**

```
<Chart:C1FlexChart x:Name="flexChart"
                   ChartType="Histogram"
                   ItemsSource="{Binding DataContext.Data}"
                   Binding="Y"
                   BindingX="X">
    <Chart:C1FlexChart.AxisX>
        <Chart:Axis Format="0.00"></Chart:Axis>
    </Chart:C1FlexChart.AxisX>
```

```
    <Chart:Histogram x:Name="histogramSeries" SeriesName="度数"/>
</Chart:C1FlexChart>
```

Frequency Polygon

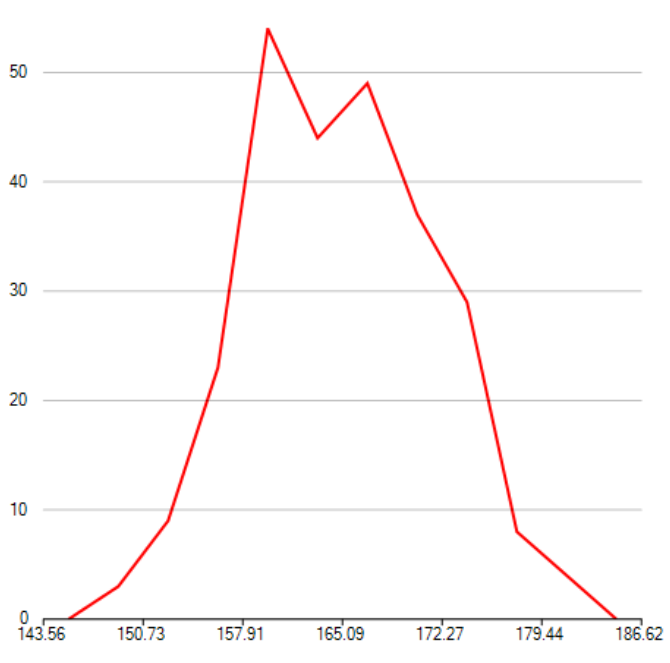
A frequency polygon shows a frequency distribution representing the overall pattern in the data. It is a closed two-dimensional figure of straight line segments - created by joining the mid points of the top of the bars of a histogram.

Use the following steps to create a frequency polygon using histogram chart.

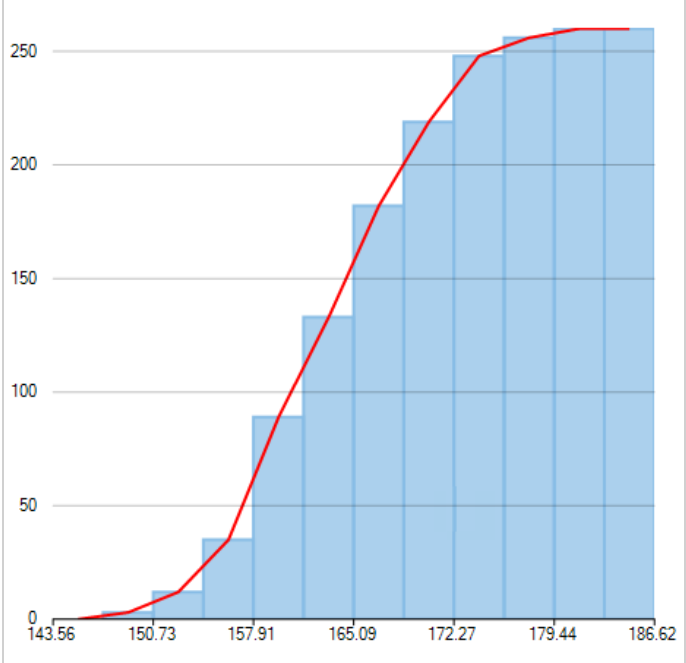
1. Set the **AppearanceType** property to FrequencyPolygon. This property accepts value from the **HistogramAppearance** enumeration.
2. Set the style for frequency polygon using the **FrequencyPolygonStyle** property.

Moreover, you can also create a cumulative frequency polygon by setting the **CumulativeMode** property to **true**.

The following images show a frequency polygon and a cumulative frequency polygon created using FlexChart.



Frequency Polygon



Cumulative Frequency Polygon

Use the following code snippet to create a frequency polygon.

In XAML

```
<c1:Histogram x:Name="histogramSeries" SeriesName="Frequency" CumulativeMode="True"
                  AppearanceType="FrequencyPolygon" />
<c1:Histogram.FrequencyPolygonStyle>
        <c1:ChartStyle Stroke="Red" StrokeThickness="2"/>
</c1:Histogram.FrequencyPolygonStyle>
```

In Code

```
histogramSeries.AppearanceType = HistogramAppearance.FrequencyPolygon;
histogramSeries.FrequencyPolygonStyle = new ChartStyle()
{Stroke = new SolidColorBrush(Color.FromRgb(255, 0, 0))};
// 累積頻度ポリゴンを作成する
histogramSeries.CumulativeMode = true;
```

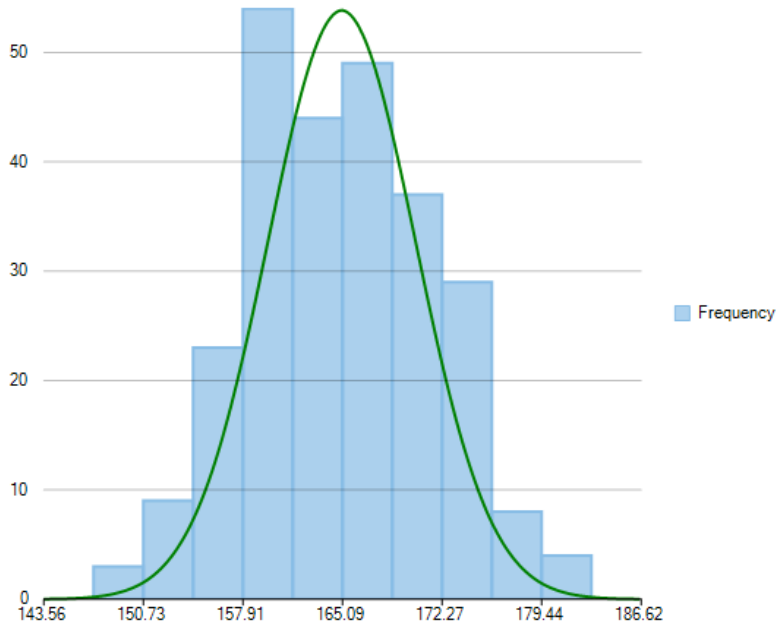
Gaussian Curve

Gaussian curve is a bell shaped curve, also known as normal curve, which represents the probability distribution of a continuous random variable. It represents a unimodal distribution as it only has one peak. Moreover, it shows a symmetric distribution as fifty percent of the data set lies on the left side of the mean and fifty percent of the data lies on the right side of the mean.

Use the following steps to create a Gaussian curve using histogram chart.

1. Set the **AppearanceType** property to Histogram. This property accepts value from the **HistogramAppearance** enumeration.
2. Set the **NormalCurve.Visible** property to **true** to create a Gaussian curve.
3. Set the style for Gaussian curve using the **NormalCurve.LineStyle** property.

Following image illustrates a Gaussian curve created using FlexChart, which depicts probability distribution of scores obtained by students of a university in half yearly examinations.



Use the following code snippet to create a Gaussian curve.

In XAML

```
<c1:Histogram x:Name="histogramSeries" SeriesName="Frequency"
                          AppearanceType="Histogram" />
<c1:Histogram.NormalCurve>
        <c1:NormalCurve>
        <c1:NormalCurve.LineStyle>
            <c1:ChartStyle Stroke="Green" StrokeThickness="2"/>
            </c1:NormalCurve.LineStyle>
    </c1:NormalCurve>
</c1:Histogram.NormalCurve>
```

In code

```
histogramSeries.AppearanceType = HistogramAppearance.Histogram;
histogramSeries.NormalCurve.Visible = true;
histogramSeries.NormalCurve.LineStyle = new ChartStyle() {Stroke = new SolidColorBrush(Color.FromRgb(0, 128, 0))};
```

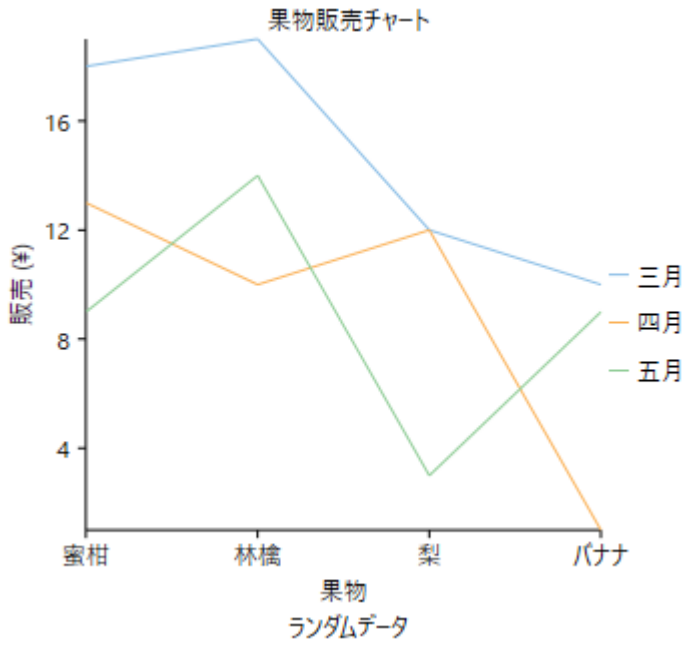
先頭に戻る

折れ線

折れ線グラフは、系列内の異なるデータポイントを直線で接続することで、一定期間の傾向を表示します。入力を X 軸に沿って等間隔に並ぶカテゴリ情報として取り扱います。

折れ線グラフは、**ChartType** プロパティを **Line** に設定することで作成できます。

積層折れ線グラフを作成するには、**Stacking** プロパティを **Stacked** または **Stacked100pc** に設定する必要があります。

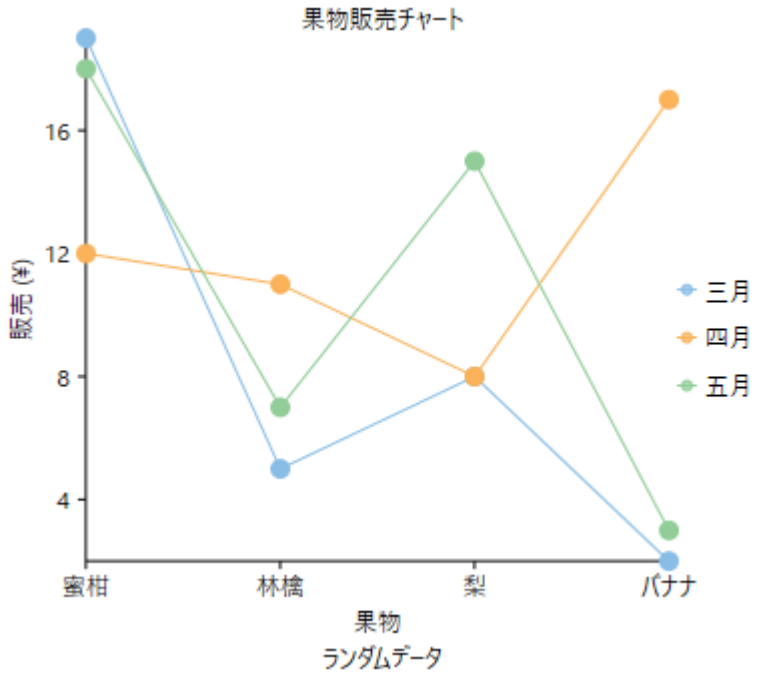


折れ線シンボル

折れ線シンボルグラフは、折れ線グラフと散布図グラフを組み合わせたグラフです。等間隔に並ぶデータの傾向を表示し、同じイベントに関連付けられた 2 つの変数の関係を視覚化します。シンボルを使用してデータポイントをプロットし、データポイント間を直線で接続します。

折れ線シンボルグラフを作成するには、**ChartType** プロパティを **LineSymbols** に設定する必要があります。

Stacking プロパティを **Stacked** または **Stacked100pc** に設定すると、積層折れ線シンボルグラフを作成できます。

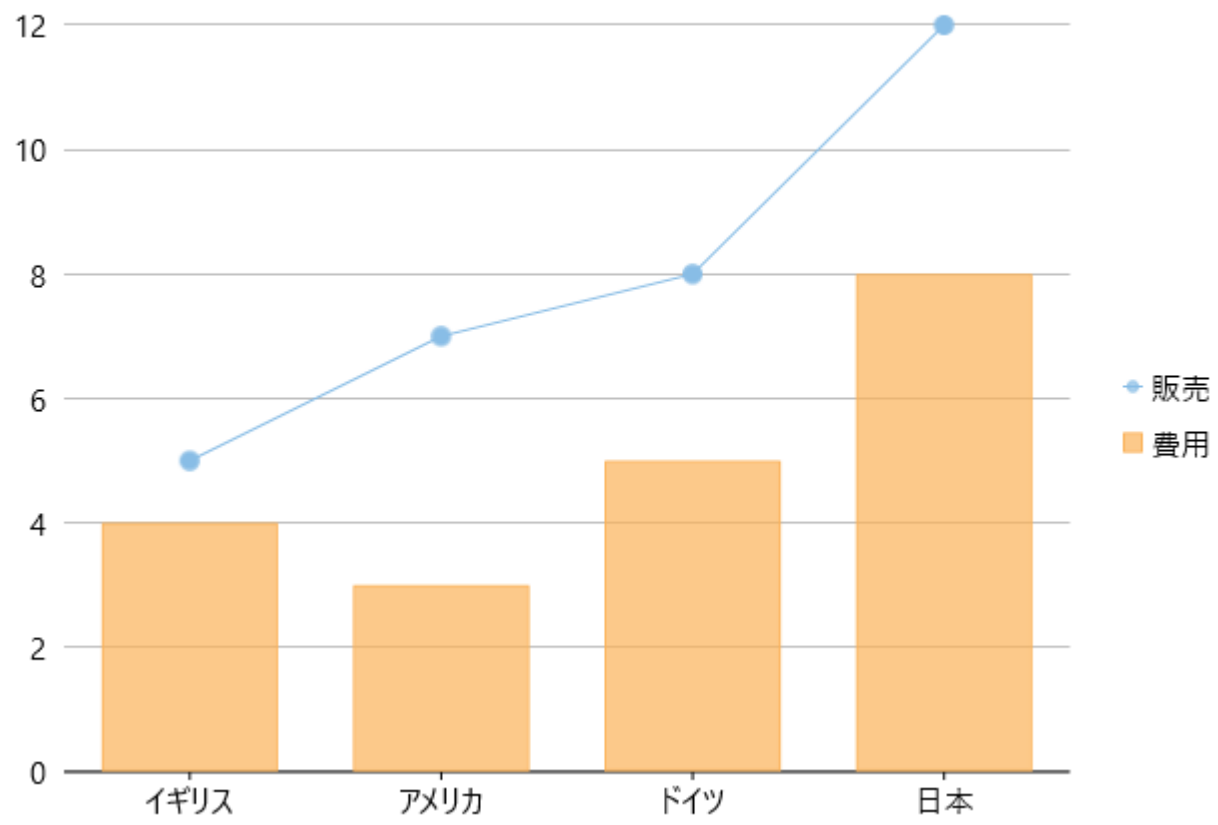


複合

FlexChart では、主に次の 2 つの利点がある複合チャートを作成できます。

- **チャートタイプの組み合わせ**:面と棒、棒と折れ線、棒と散布図など、複数のチャートを組み合わせて 1 つのチャートにすることができます。さまざまなチャートタイプを使用して 1 つのチャートに複数のメトリックをプロットし、エンドユーザーが簡単にデータを解釈できるようにします。FlexChart で系列ごとにチャートタイプを指定して、複数のチャートタイプを組み合わせます。系列のチャートタイプを指定するには、SeriesBase クラスの **ChartType** プロパティを設定します。このプロパティを設定すると、チャートで設定されている**ChartType**プロパティがオーバーライドされます。
- **複数のデータセットのプロット**:系列のデータソースを指定して、複数のデータセットのデータを 1 つのチャートにプロットします。プロットするデータが複数の場所にある場合に便利です。系列のデータソースを指定するには、**Series** クラスの DataSource プロパティを設定します。このプロパティを設定すると、チャートで設定されている DataSource プロパティがオーバーライドされます。

次の図に、棒グラフと折れ線グラフ(シンボル付き)のチャートタイプを組み合わせた複合チャートを示します。このチャートでは、シリンダー容積、エンジン排気量、トランスミッション速度の 3 つのパラメータに基づいて、5 つの自動車を比較します。



- **XAML**

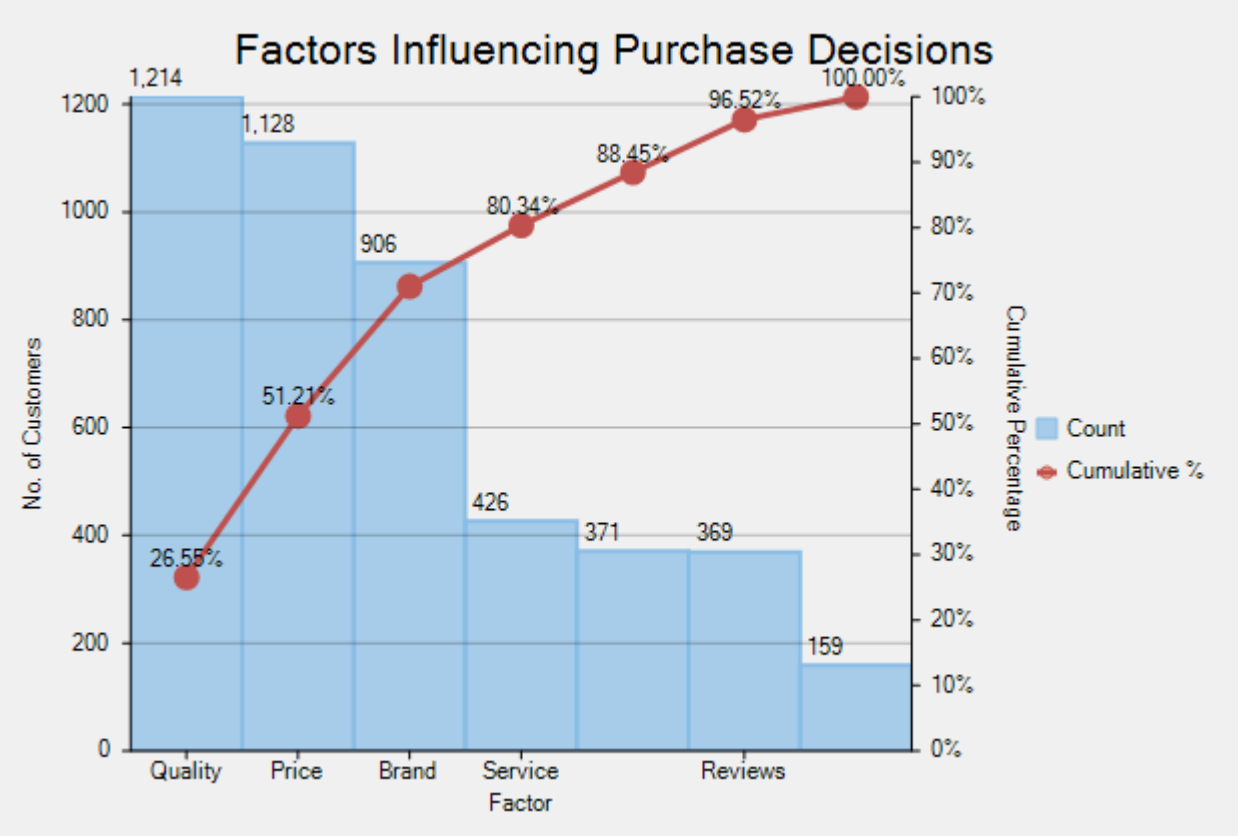
```
<Chart:C1FlexChart x:Name="flexChart"
                   BindingX="Country"
                   ChartType="Column"
                   ItemsSource="{Binding DataContext.Data}" Margin="10,0,723,10">
    <Chart:Series SeriesName="販売"
                  Binding="Sales"
                  ChartType="LineSymbols"/>
    <Chart:Series SeriesName="費用"
                  Binding="Expenses"/>
</Chart:C1FlexChart>
```

パレート図

Pareto chart is a type of chart that contains both bar and a line chart. It is a vertical bar chart in which values are plotted in decreasing order of relative frequency from left to right. The categories or factors that represent the bigger bars on the left are more important than those on the right. The line chart plots the cumulative total percentage of frequencies that are represented by the bars.

Pareto chart is essentially used in scenarios where the data is broken into different categories, and when the developer needs to highlight the most important factors from a given set of factors. For example, quality control, inventory control, and customer grievance handling are some areas where Pareto chart analysis can be frequently used.

In FlexChart, Pareto chart can be easily created by combining RangedHistogram chart with any of Line, Spline, LineSymbol, or SplineSymbol chart. First, plot the relative frequency on a RangedHistogram in descending order. Then, calculate the cumulative relative frequency in percentage using original data to create another series which is plotted on any of the Line, Spline, LineSymbol, or SplineSymbol chart. This forms Pareto line of the chart which helps in identifying the added contribution of each category.



To implement Pareto chart using the FlexChart control, see **FlexChartExplorer** sample. The samples are available in the default installation folder - *Documents\ComponentOne Samples*

RangedHistogram

RangedHistogram は、新しい Excel 形式のヒストグラムで、**範囲付き** X 軸に対する Y 軸の度数分布を視覚化するために役立ちます。ヒストグラムチャートタイプと同様に、ビンは、生データ値を重複のない連続した間隔に分割することで作成されます。特定のビンに入る値の数に基づき、X 軸に沿った長方形の柱として度数がプロットされます。

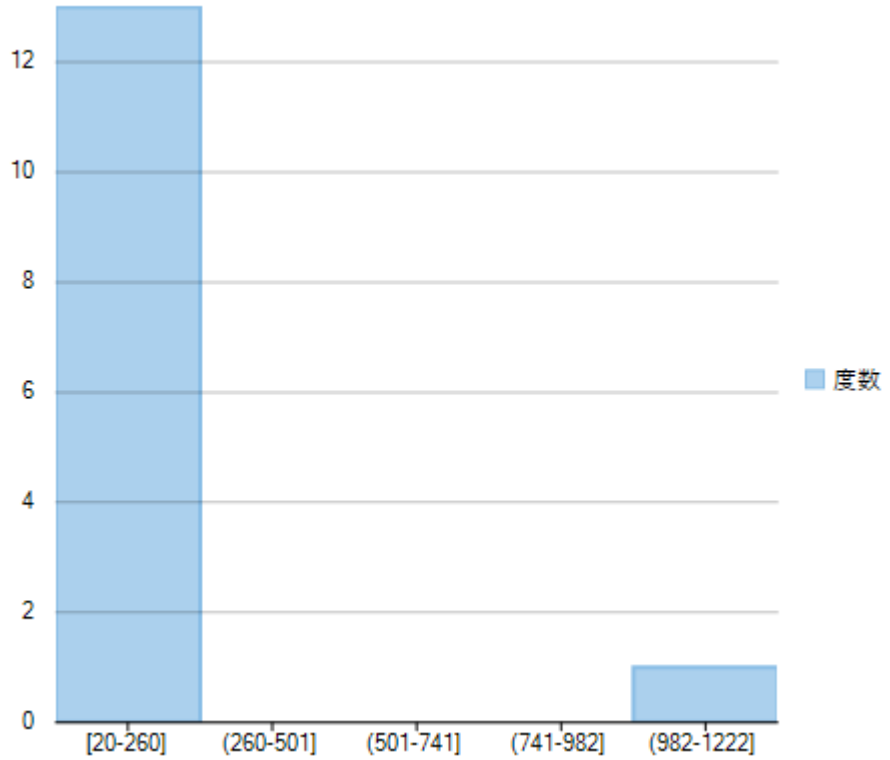
RangedHistogram は、提供されたデータの度数分布を非カテゴリモードまたはカテゴリモードでプロットします。

非カテゴリモード

非カテゴリモードでは、元のデータポイントがいくつかの間隔または範囲に分類されます。これらの間隔が X 軸にプロットされ、対応する範囲の度数分布が Y 軸に示されます。FlexChart は、データをグループ化する間隔を自動的に計算します。

ただし、この動作は、**BinMode** プロパティで **HistogramBinning** を指定することで制御できます。さらに、**BinWidth**、**NumberOfBins**、**UnderflowBin** および **OverflowBin** の値を設定し、**ShowUnderflowBin** と **ShowOverflowBin** を指定できます。

次の図に、ある小売店の製品販売数の度数分布を非カテゴリモードで示します。



特定のデータの RangedHistogram を非カテゴリモードで作成するには、次のコードスニペットに示すように、**RangedHistogram** 系列を追加し、**ChartType** プロパティを **RangedHistogram** に設定する必要があります。

- **Xaml**

```
<Chart:C1FlexChart x:Name="flexChart"
                       ChartType="RangedHistogram"
                   ItemsSource="{Binding DataContext.Data}"
                   Binding="Value">
    <Chart:RangedHistogram x:Name="RangedhistogramSeries"
                           SeriesName="度数"
                           BinMode="NumberOfBins"
                           NumberOfBins="5"
                           OverflowBin="89"
                           UnderflowBin="20"
                           ShowOverflowBin="True"
                           ShowUnderflowBin="True"/>
</Chart:C1FlexChart>
```

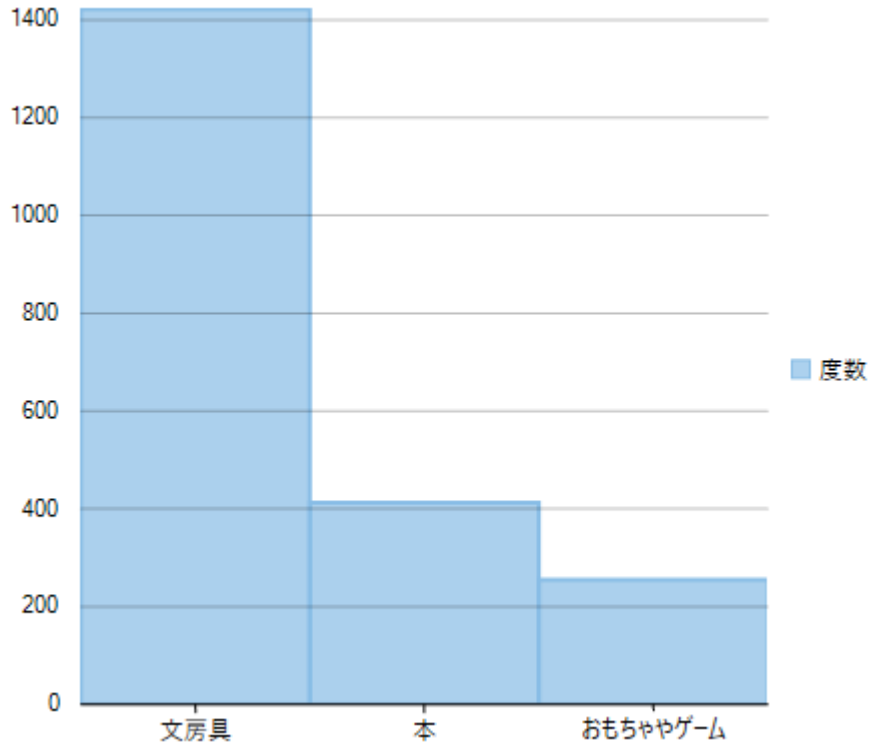
カテゴリモード

カテゴリモードでは、度数データが元のデータに基づいて提供されるいくつかのカテゴリに排他的にグループ化され（X 軸にプロット）、対応するカテゴリの累積的度数が Y 軸に描画されます。**BindingX** プロパティを設定すると、RangedHistogram 系列のカテゴリモードが有効になります。

このモードでは、RangedHistogram 系列の BinMode、BinWidth、NumberOfBins、OverflowBin、UnderflowBin の各プロパティは無視されます。

次の図に、ある小売店の 3 つのカテゴリの製品（文房具、本、おもちゃとゲーム）の販売数の度数分布をカテゴリモードで示し

ます。



特定のデータの RangedHistogram をカテゴリモードで作成するには、次のコードスニペットに示すように、**RangedHistogram** 系列を追加し、**ChartType** プロパティを **RangedHistogram** に設定し、**BindingX** プロパティを設定する必要があります。

- **Xaml**

```
<Chart:C1FlexChart x:Name="flexChart"
                   ChartType="RangedHistogram"
                   ItemsSource="{Binding DataContext.Data}"
                   Binding="Value"
                   BindingX="Name">
  <Chart:RangedHistogram x:Name="RangedhistogramSeries"
                         SeriesName="度数" />
</Chart:C1FlexChart>
```

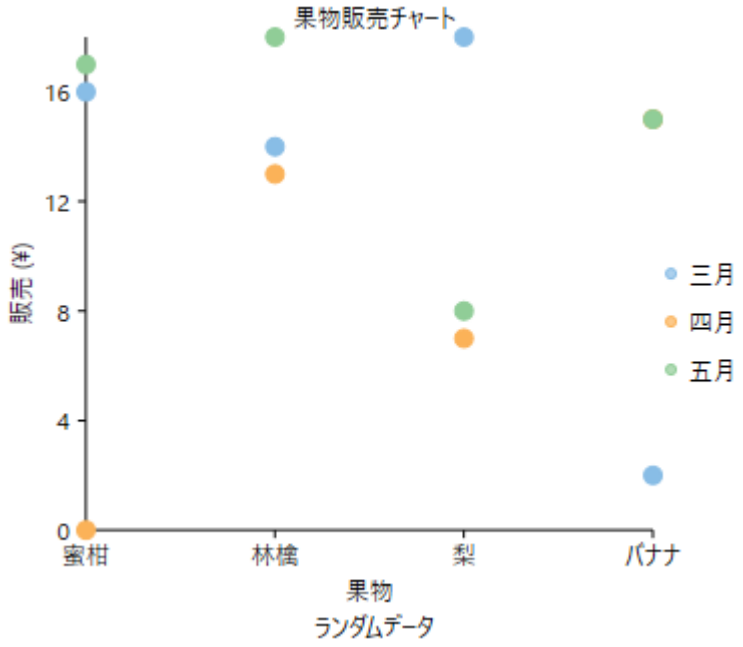
従来のヒストグラムとは異なり、RangedHistogram と同じ X 軸を使用して他のチャートタイプをプロットすることはできません。

散布図

散布図グラフは別名 XY グラフと呼ばれ、複数のデータ系列の項目間の関係を表します。簡単に言えば、X 値と Y 値を 2 つの軸にプロットしたものです。データポイントは接続されず、異なるシンボルを使用してカスタマイズできます。通常、このチャートタイプは科学的データを表現するために使用され、予測データや結果データに含まれる集中データのばらつきを強調できます。

散布図グラフを作成するには、**ChartType** プロパティを **Scatter** に設定する必要があります。

Stacking プロパティを **Stacked** または **Stacked100pc** に設定すると、積層散布図グラフを作成できます。

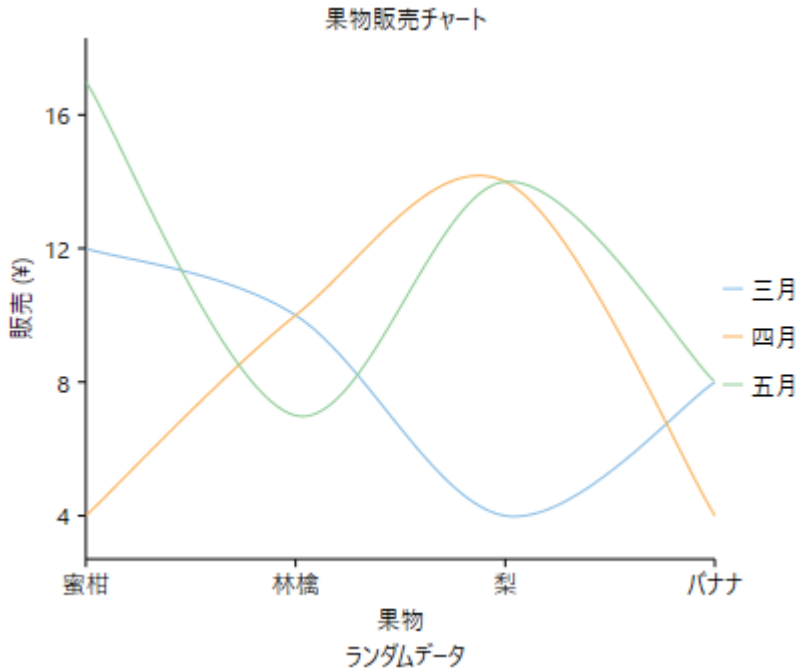


スプライン

スプライングラフは折れ線グラフと似ていますが、直線ではなくスプラインを使用してデータポイント間を接続する点が異なります。このグラフは折れ線グラフの代わりに使用されますが、より具体的に言えば、曲線フィッティングを使用する必要があるデータの表現に使用されます。

スプライングラフを作成するには、**ChartType** プロパティを **Spline** に設定する必要があります。

Stacking プロパティを **Stacked** または **Stacked100pc** に設定すると、積層スプライングラフを作成できます。

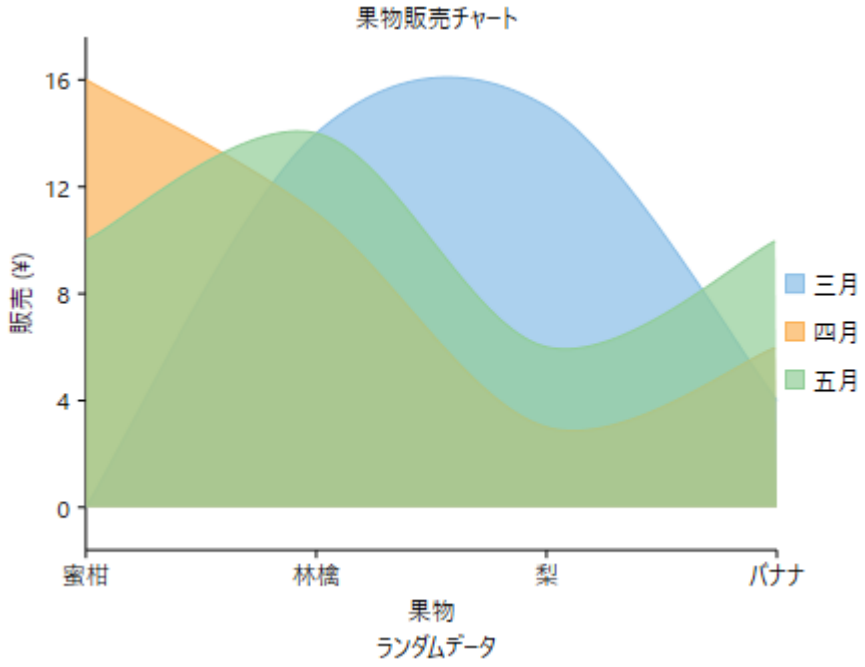


スプライン面

スプライン面グラフは、データポイントの接続方法が異なる以外は、面グラフと同じです。 スプライン面グラフは、直線ではなくスプラインを使用してデータポイント間を接続し、スプラインで囲まれた領域を塗りつぶします。

ChartType プロパティを **SplineArea** に設定すると、スプライン面グラフを作成できます。

積層スプライン面グラフを作成するには、**Stacking** プロパティを **Stacked** または **Stacked100pc** に設定します。

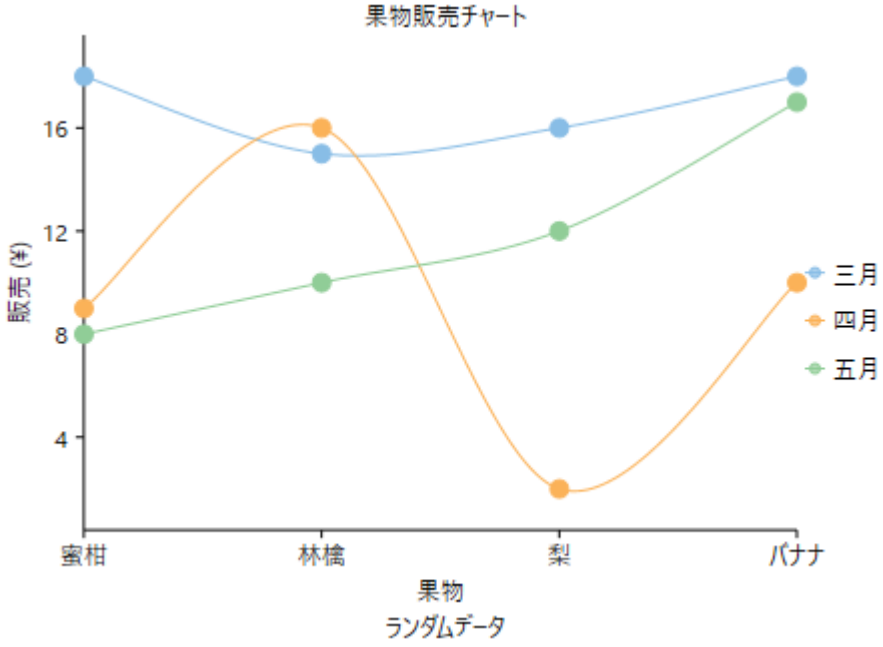


スプラインシンボル

スプラインシンボルグラフは、スプライングラフと散布図グラフを組み合わせたグラフです。シンボルを使用してデータポイントをプロットし、データポイント間をスプラインで接続します。

スプラインシンボルグラフを作成するには、**ChartType** プロパティを **SplineSymbols** に設定します。

Stacking プロパティを **Stacked** または **Stacked100pc** に設定すると、積層スプラインシンボルグラフを作成できます。



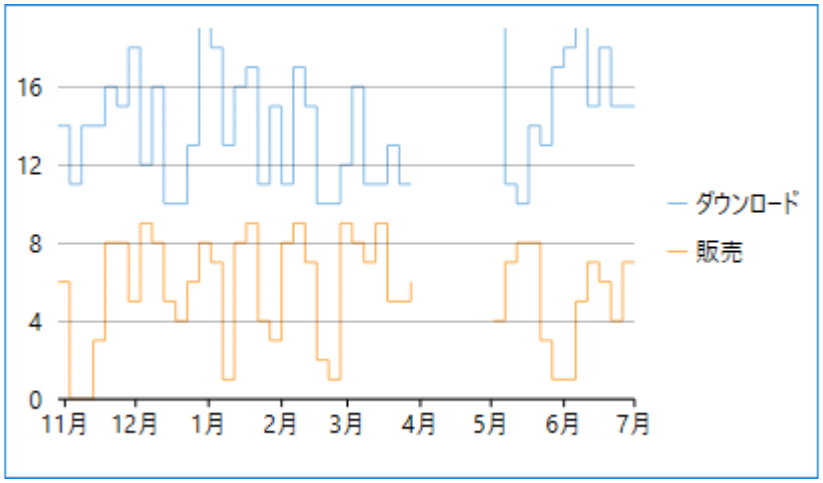
階段グラフ

階段グラフは、Y 軸方向に不連続の急な変化を示すデータを、水平線と垂直線を使用して表示します。このチャートは、不規

則に急な変化を示すが、次の変化までは一定値を維持するデータを表示する際に便利です。階段グラフを使用すると、データの傾向と、その傾向が一定値を維持する期間を判断できます。

あるソフトウェアの週ごとの売上とダウンロード数を視覚化して比較する例を考えます。これらの値は不連続に変化するため、階段グラフを使用して視覚化できます。以下の図に示すように、これらのチャートは売上の変化を描画するだけでなく、変化の正確なタイミングと売上が一定だった期間も示します。さらに、チャートを見るだけで、それぞれの変化の大きさが簡単にわかります。

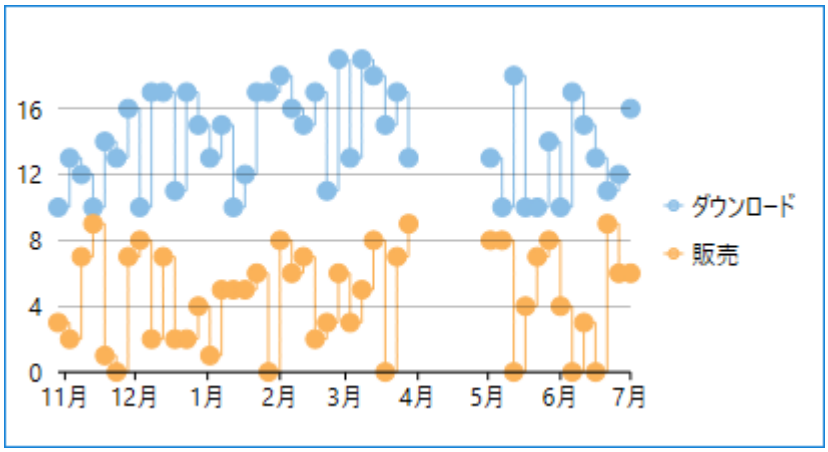
FlexChart は階段グラフ、階段シンボルグラフ、階段面グラフ、塗りつぶし階段グラフをサポートします。次の表で、これらのチャートタイプについて詳細に説明します。



階段グラフ

階段グラフは折れ線グラフに似ていますが、折れ線グラフは、連続するデータポイントを最短距離で接続するのに対して、階段グラフは、水平線と垂直線で接続します。これらの水平線と垂直線により、チャートは階段のような外観になります。

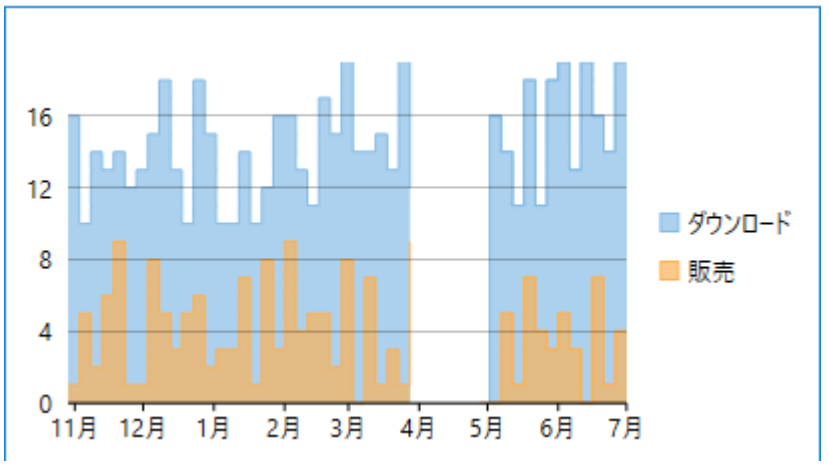
折れ線グラフは変化とトレンドを描写しますが、階段グラフでは、変動の大きさと断続的なパターンも判断できます。



階段シンボルグラフ

階段シンボルグラフは、階段グラフと散布図を組み合わせたグラフです。FlexChart は、シンボルを使用してデータポイントをプロットし、データポイント間を水平および垂直の階段型折れ線で接続します。

ここでは、シンボルを使用してデータポイントがマークされ、断続的な変化の始まりを明示できるので便利です。



階段面グラフは、階段グラフと面グラフを組み合わせたグラフです。面グラフに似ていますが、データポイントを接続する方法が異なります。FlexChart は、水平および垂直の階段型折れ線を使用してデータポイントをプロットし、X 軸と階段型折れ線の間の領域を塗りつぶします。

これらは階段グラフをベースにしており、一般に 2 つ以上の数量の不連続の断続的な変化を比較するために使用されます。これにより、チャートは積層グラフの外観になり、複数の系列の関連するデータポイントが積み重なったように表示されます。

階段面グラフ	たとえば、図で示すように、特定の期間のソフトウェアのダウンロード数と販売数を簡単に比較できます。

階段グラフを作成するには、チャートで適切なデータを生成し、**ChartType** プロパティを **Step** に設定する必要があります。ただし、階段面グラフまたは階段シンボルグラフを作成するには、**ChartType** プロパティをそれぞれ **StepArea** または **StepSymbols** に設定する必要があります。

- **XAML**

```
<Chart:C1FlexChart x:Name="flexChart"
                   BindingX="Date"
                   ItemsSource="{Binding DataContext.Data}"
                   HorizontalAlignment="Stretch"
                   VerticalAlignment="Stretch"
                   ChartType="Step" Margin="10,135,15,135">
    <Chart:C1FlexChart.AxisX>
        <Chart:Axis/>
    </Chart:C1FlexChart.AxisX>
    <Chart:C1FlexChart.Series>
        <Chart:Series SeriesName="Downloads" Binding="Downloads"></Chart:Series>
        <Chart:Series SeriesName="Sales" Binding="Sales"></Chart:Series>
    </Chart:C1FlexChart.Series>
</Chart:C1FlexChart>
```

C#

```
flexChart.ChartType = C1.Chart.ChartType.Step;
```

FlexChart の操作

FlexChart for UWP をアプリケーションの開発に使用するには、コントロールが提供するいくつかの機能の利用方法を理解する必要があります。

このセクションでは、FlexChart が提供する機能に関して、重要な概念情報をタスクベースで説明します。

以下のリンクから、FlexChart のさまざまな操作方法を説明するセクションにアクセスできます。

- データ
- 外観
- エンドユーザー操作
- FlexChart の要素
- Trend Lines
- エクスポート

データ

グラフにまず第一に必要なものはデータです。データなしでは、グラフは視覚化どころか何も表示できません。したがって、グラフを操作する際の最初の仕事は、グラフにデータを表示することであり、それによってデータを操作し、解釈できるようになります。

グラフデータに関して言えば、データは、主に 2 つの段階で指定、表現、および解釈されます。

- データの提供
- データのプロット

これらのセクションにアクセスすると、FlexChart についてこれらの段階の詳細を参照できます。

データの提供

グラフにデータをプロットするには、まずグラフにデータを提供する必要があります。

グラフにデータを提供するために最もよく使用されている手法は、データ連結です。

以下のリンクをクリックすると、FlexChart にデータを連結する方法を参照できます。

- データソースを使用したデータ連結

データソースを使用したデータの連結

データを連結するということは、1 つ以上のデータコンシューマーをデータプロバイダに同期して接続するということです。データが連結されると、グラフは、連結されたすべてのデータを指定された系列のデータソースとして使用し、系列とグラフプロパティに従ってデータをグラフ面上に表現します。

データソースの内容と実際のグラフの間には少し距離があるため、データを集約しないとプロットできないことがよくあります。ただし、プロットするデータが既にデータビューや別のデータソースオブジェクトとして用意されていることもあります。その場合は、グラフを直接データソースオブジェクトに連結できます。

FlexChart コントロールをデータソースに連結するには、まず **ItemsSource** プロパティをデータソースオブジェクトに設定する必要があります。次に、グラフの各系列をデータソースオブジェクト内のフィールドに連結する必要があります。それには、**BindingX** プロパティと **Binding** プロパティを使用します。

> 連結ソースを指定するには、**DataContext = "{Binding RelativeSource={RelativeSource Mode=Self}}"** マークアップを **MainPage.xaml** ファイルの **<Page>** タグに追加する必要があります。

データ連結を実装する完全な機能を備えたプログラムを示すコードは次のとおりです。このコードは、**DataCreator.cs** クラスを使用してグラフのデータを生成します。

XAML

- **Tab Caption**

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:Data_Binding"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:Chart="using:C1.Xaml.Chart" xmlns:Foundation="using:Windows.Foundation"
    x:Class="Data_Binding.MainPage"
    DataContext="{Binding RelativeSource={RelativeSource Mode=Self}}"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

        <Chart:C1FlexChart x:Name="flexChart" BindingX="Fruit"
        ItemsSource="{Binding DataContext.Data}" ChartType="Bar">
            <Chart:C1FlexChart.Series>
                <Chart:Series SeriesName="三月" Binding="March"/>
                <Chart:Series SeriesName="四月" Binding="April"/>
                <Chart:Series SeriesName="五月" Binding="May"/>
            </Chart:C1FlexChart.Series>
        </Chart:C1FlexChart>
```

```
    </Grid>
</Page>
```

Code

DataCreator.cs copyCode

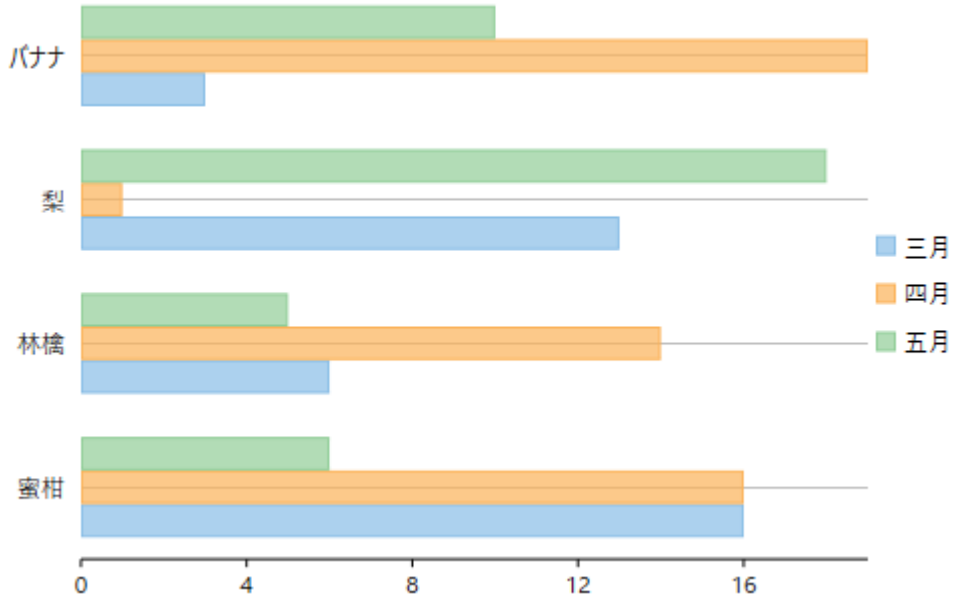
```
    class DataCreator
    {
        public static List<FruitDataItem> CreateFruit()
        {
var fruits = new string[] { "蜜柑", "林檎", "梨", "バナナ" };
            var count = fruits.Length;
            var result = new List<FruitDataItem>();
            var rnd = new Random();
            for (var i = 0; i < count; i++)
                result.Add(new FruitDataItem()
                {
                    Fruit = fruits[i],
                    March = rnd.Next(20),
                    April = rnd.Next(20),
                    May = rnd.Next(20),
                });
            return result;
        }

    }
    public class FruitDataItem
    {
        public string Fruit { get; set; }
        public double March { get; set; }
        public double April { get; set; }
        public double May { get; set; }
    }

    public class DataPoint
    {
        public double XVals { get; set; }
        public double YVals { get; set; }
    }
```

MainWindow.xaml.cs copyCode

```
public sealed partial class MainPage : Page
{
    private List<FruitDataItem> _fruits;

    public MainPage()
    {
        this.InitializeComponent();

    }
```

```
    public List<FruitDataItem> Data
    {
        get
        {
            if (_fruits == null)
            {
                _fruits = DataCreator.CreateFruit();
            }

            return _fruits;
        }
    }
}
```

コードを実行すると、次の出力が表示されます。



データのプロット

FlexChart for UWP は、関連する値が **BindingX** および **Binding** プロパティに設定されている場合に、フィールドまたはデータ配列の形式で連結されたデータをプロットします。

使用するチャートタイプに従って、**BindingX** および **Binding** プロパティに値を設定する必要があります。たとえば、散布図グラフの場合は、BindingX プロパティと Binding プロパティにそれぞれ 1 つの値（フィールド）を設定する必要があります。一方、バブルチャートの場合は、BindingX プロパティに 1 つの値（フィールド）を設定し、Binding プロパティに 2 つの値（Y 値を指定するフィールドと、バブルのサイズを指定するフィールド）を設定する必要があります。

次のコードスニペットを参照してください。

散布図グラフの場合

- **XAML**

```
<Chart:C1FlexChart x:Name="flexChart" BindingX="Country"
ItemsSource="{Binding DataContext.Data}" ChartType="Scatter">
    <Chart:C1FlexChart.Series>
```

```
        <Chart:Series SeriesName="販売" Binding="Sales"/>
        <Chart:Series SeriesName="費用" Binding="Expenses"/>
    </Chart:C1FlexChart.Series>
</Chart:C1FlexChart>
```

バブルチャートの場合

- **XAML**

```
<Chart:C1FlexChart x:Name="flexChart" BindingX="X"
ItemsSource="{Binding DataContext.Data}" ChartType="Bubble">
    <Chart:C1FlexChart.Series>
        <Chart:Series SeriesName="Bubble" Binding="Y,Size"/>
    </Chart:C1FlexChart.Series>
</Chart:C1FlexChart>
```

データがプロットされたら、それを使用して目的に合わせてデータを視覚化できます。

以下のセクションでは、系列をカスタマイズして不規則なデータをプロットする方法について説明します。

- 系列の表示または非表示
- Null 値の補間

系列の表示または非表示

系列がチャートに表示されたら、表示された系列をカスタマイズして、より効率的な管理を行うことができます。

FlexChart for UWP では、プロット領域、凡例、またはその両方で系列を表示または非表示にして系列をカスタマイズできます。

チャートのスペースには限りがあるため、チャートに表示される系列が大量にある場合は、系列の管理が確実に必要になります。

FlexChart では、系列の **Visibility** プロパティを使用して系列を管理できます。**Visibility** プロパティは、**SeriesVisibility** 列挙型の値を受け取ります。

このプロパティを以下に示す値に設定して、系列を表示または非表示にすることができます。

値	説明
SeriesVisibility.Visible	系列はプロットと凡例の両方に表示されます。
SeriesVisibility.Plot	系列はプロットには表示されますが、凡例には表示されません。
SeriesVisibility.Legend	系列は凡例には表示されますが、プロットには表示されません。
SeriesVisibility.Hidden	系列はプロットと凡例のどちらにも表示されません。

次のコードスニペットは、Visibility プロパティを設定する方法を示しています。

XAML

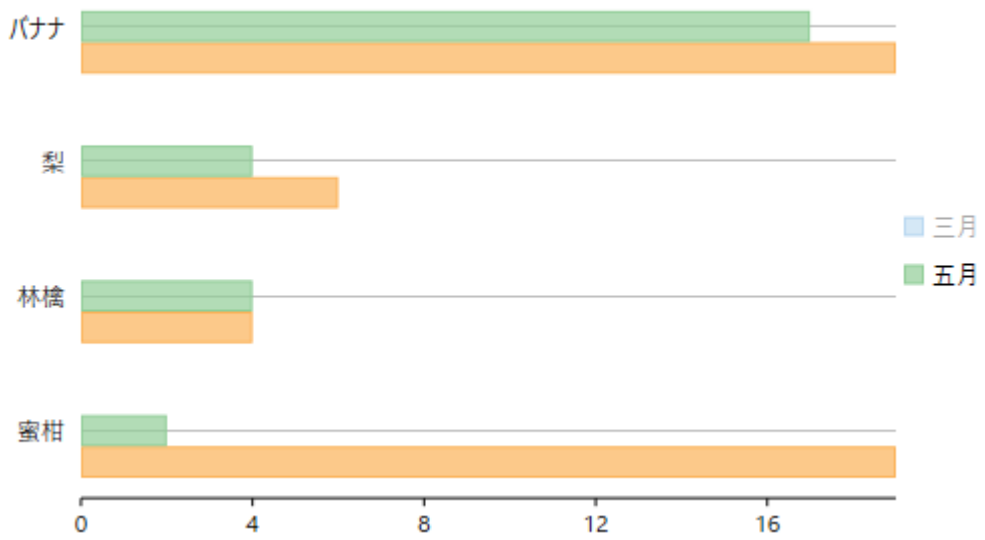
- **タブキャプション**

```
<Chart:Series SeriesName="三月" Binding="March" Visibility="Legend"/>
<Chart:Series SeriesName="四月" Binding="April" Visibility="Plot"/>
<Chart:Series SeriesName="五月" Binding="May"/>
```

コード

C# copyCode

```
series1.SeriesName = "三月";
series2.SeriesName = "四月";

series1.Visibility = C1.Chart.SeriesVisibility.Legend;
series2.Visibility = C1.Chart.SeriesVisibility.Plot;
```



Null 値の補間

データをプロットする目的で **FlexChart** に連結されるデータテーブルのデータフィールドには、null 値が含まれることがよくあります。null 値を含むデータをプロットすると、FlexChart にギャップができます。プロットされたデータにギャップがあると、データが不完全で、整合性がないように見えます。

FlexChart では、**InterpolateNulls** プロパティを使用することで、このような整合性のなさを解決することができます。**InterpolateNulls** プロパティを設定して、データ内の null 値によってできたギャップを自動的に埋めることができます。

InterpolateNulls プロパティは、折れ線グラフと面グラフにのみ適用できます。

次に、**InterpolateNulls** プロパティの設定方法を示します。

- **C#**

```
flexChart.Options.InterpolateNulls = true;
```

外観

チャートの外観は、チャートの全体的なルックアンドフィールを決定します。見栄えがよくわかりやすい外観は、読み手の目をビジュアル表現されたデータに引きつけます。データの解釈も容易になります。

FlexChart の外観は、さまざまな方法でカスタマイズできます。以下のセクションで、これらについて説明します。

- 色
- フォント
- 系列のシンボルスタイル

色

色を使用して、チャートの見た目の印象を向上させることができます。対話式に色を選択する、チャートパレットを設定する、RGB 値を指定する、色相/彩度/輝度を指定する、透明色を使用するなどして、色をカスタマイズできます。

FlexChart では、チャート全体の色だけでなく、次の要素の色もカスタマイズできます。

- 系列
- ヘッダーとフッター
- 凡例
- プロット領域
- ラベル

以下のリンクをクリックして、色のさまざまな使用方法を参照してください。

- 対話式の色の選択
- チャートパレットの設定
- RGB 色の指定
- 色相/彩度/輝度の指定
- 透明色の使用

対話式の色の選択

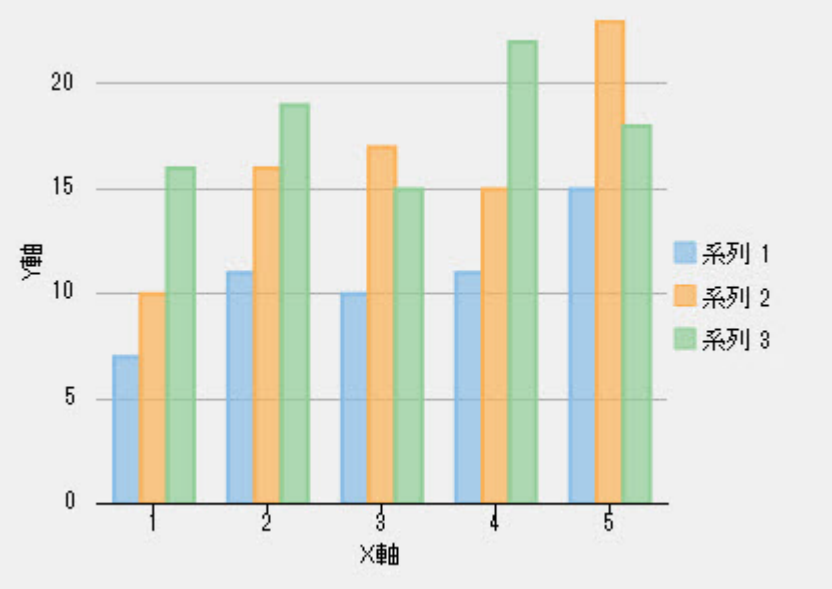
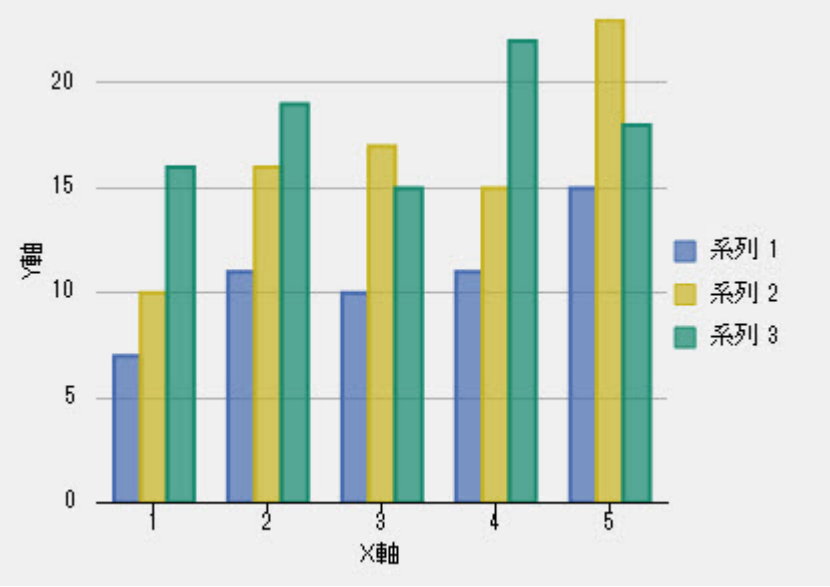
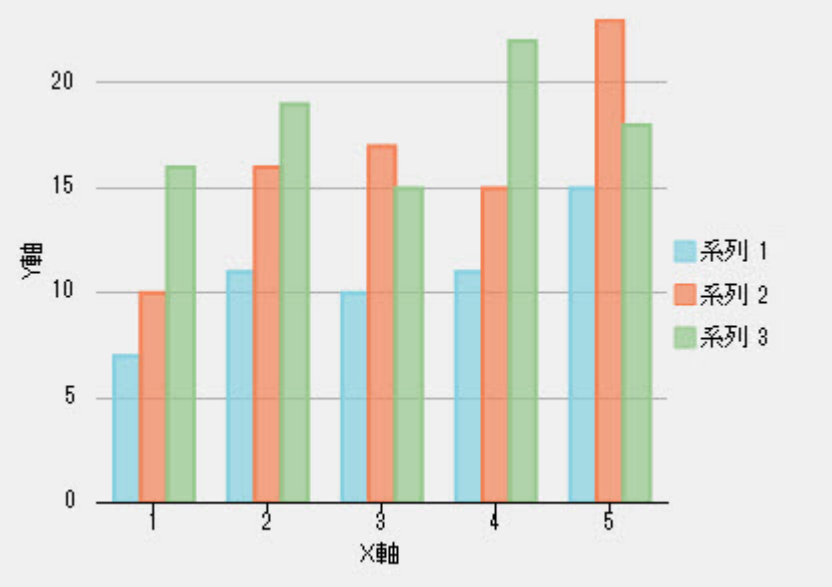
Windows の標準の色指定ダイアログのように動作する .NET の色指定ダイアログを使用して、対話式に色を選択できます。色は、Windows の基本色またはカスタム色から選択できます。また、全カラースペクトルから対話式に選択することもできます。

FlexChart: パレットの設定

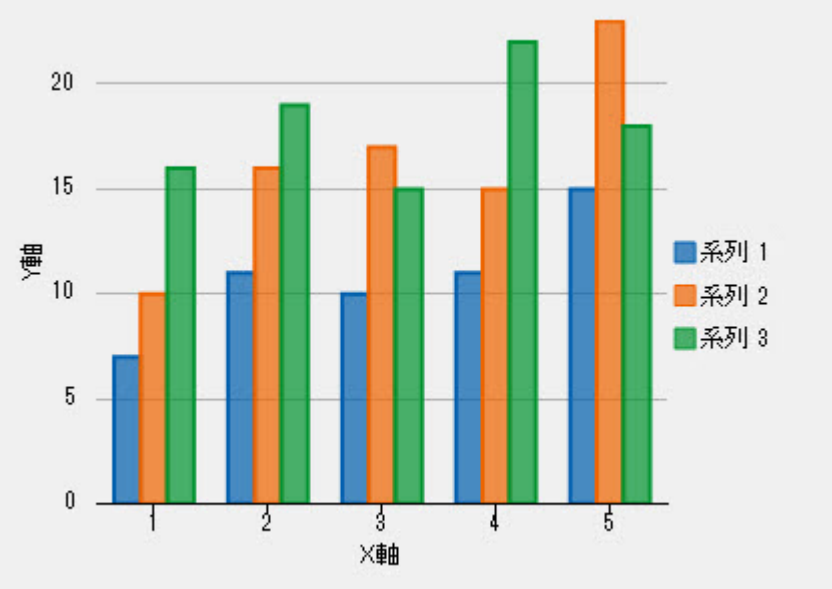
Palette プロパティを使用して、好みの **FlexChart** パレットを設定できます。デフォルトでは、標準のチャートパレットを指定する **Palette.Standard** 設定が使用されます。

次に、FlexChart で使用できるパレットを示します。

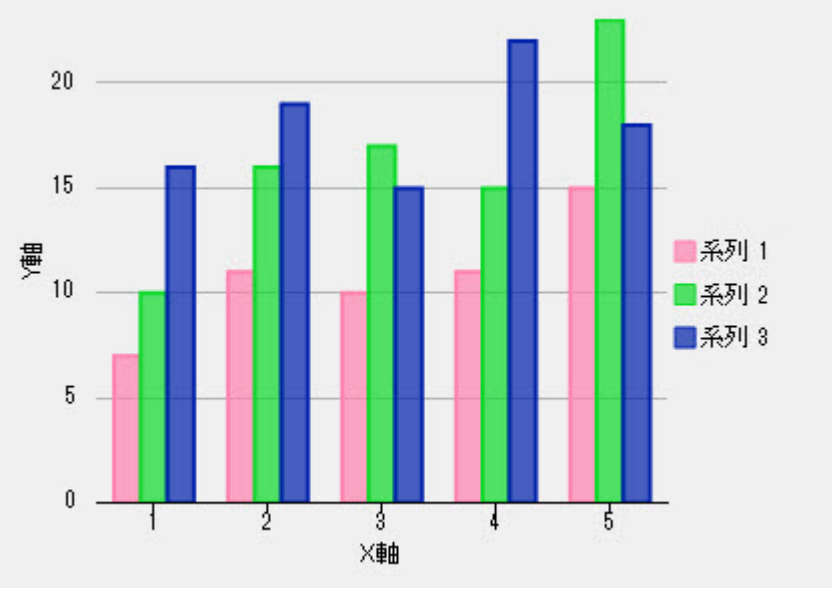
パレット設定	プレビュー

Standard（デフォルト）	
Cocoa	
Coral	

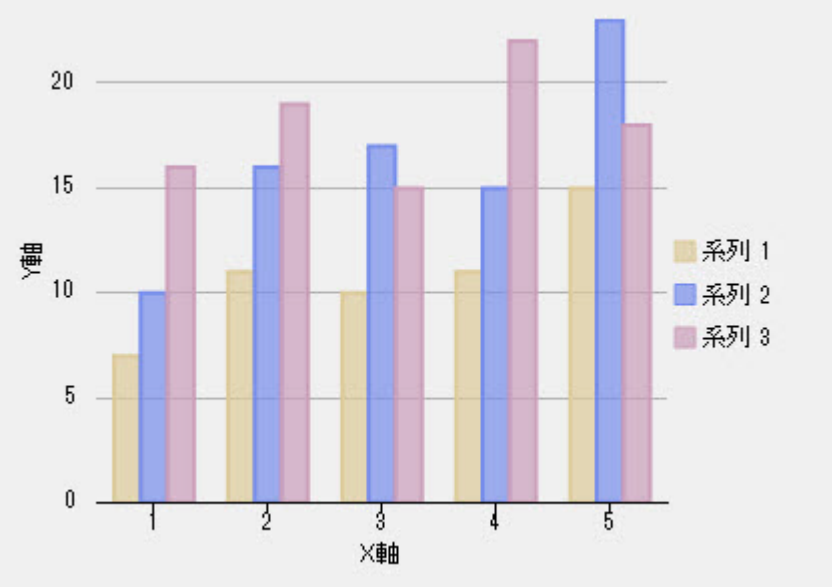
Dark

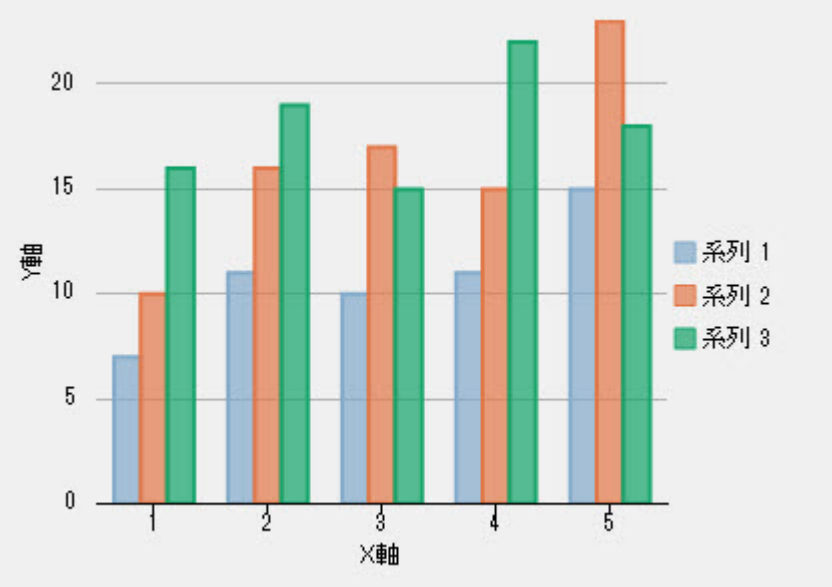
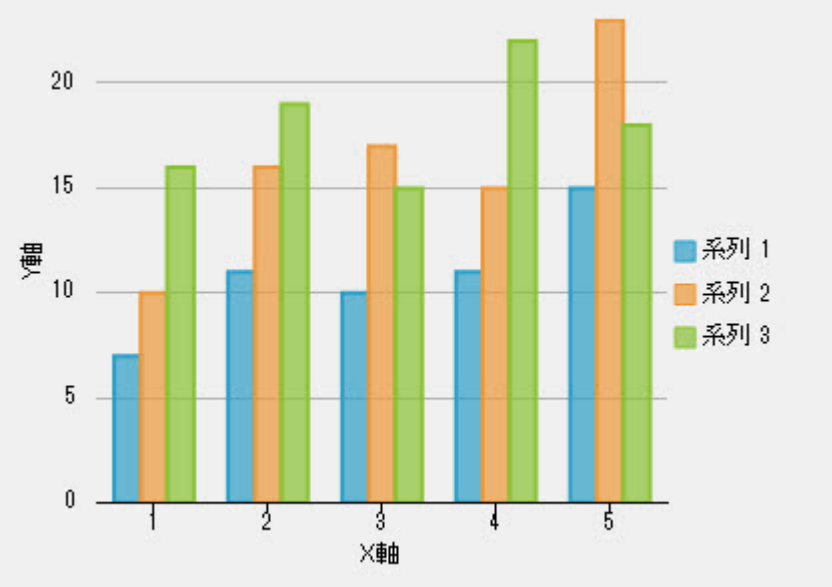
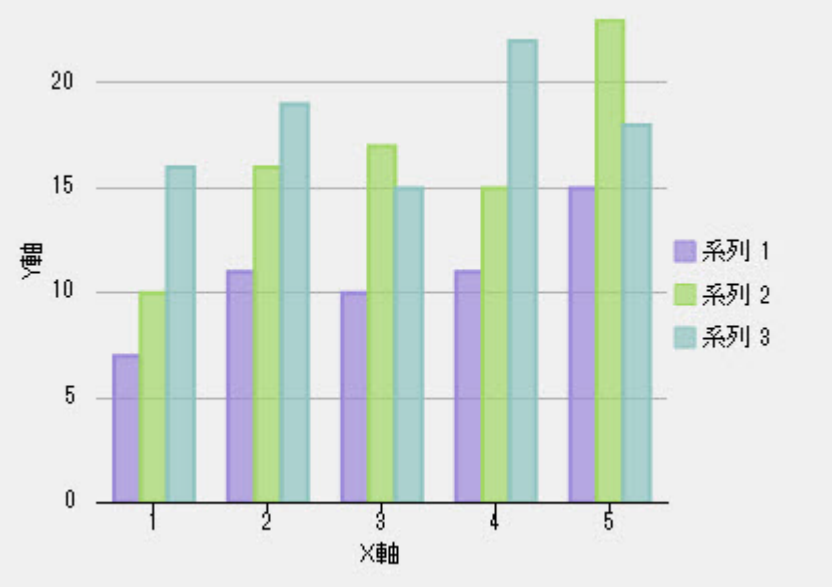


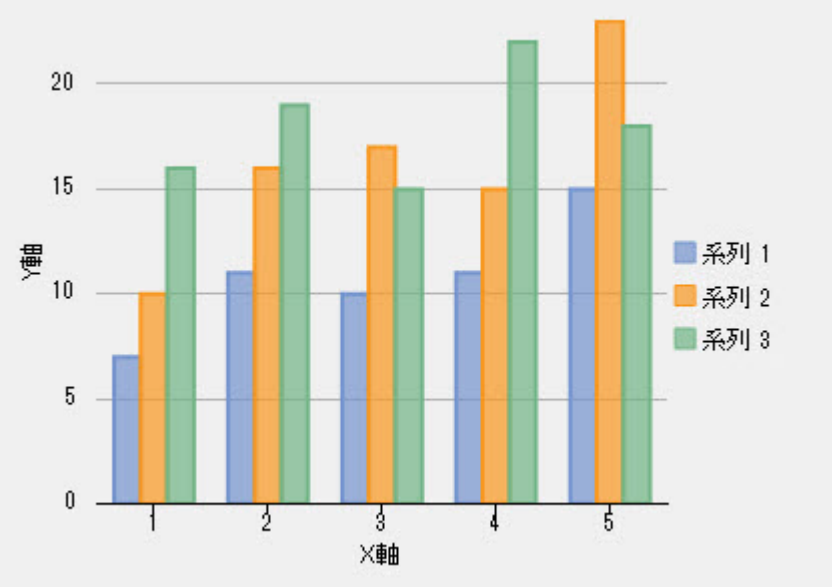
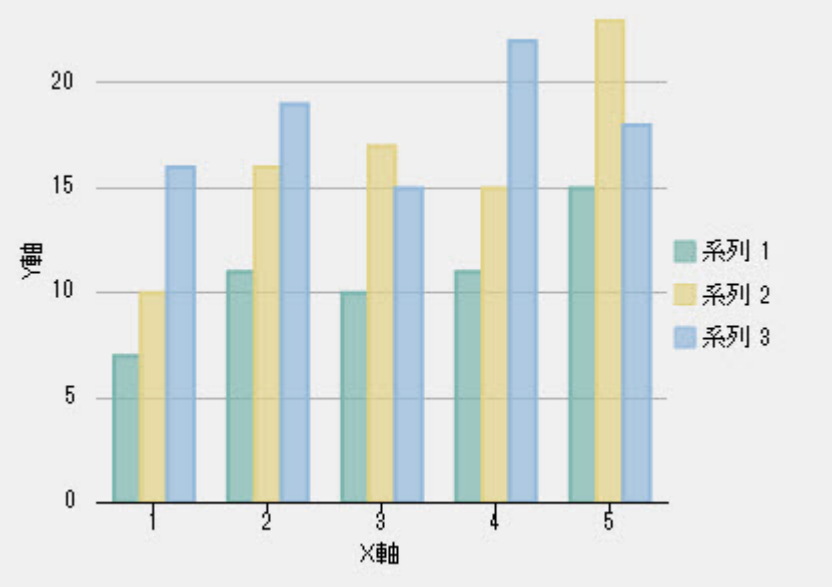
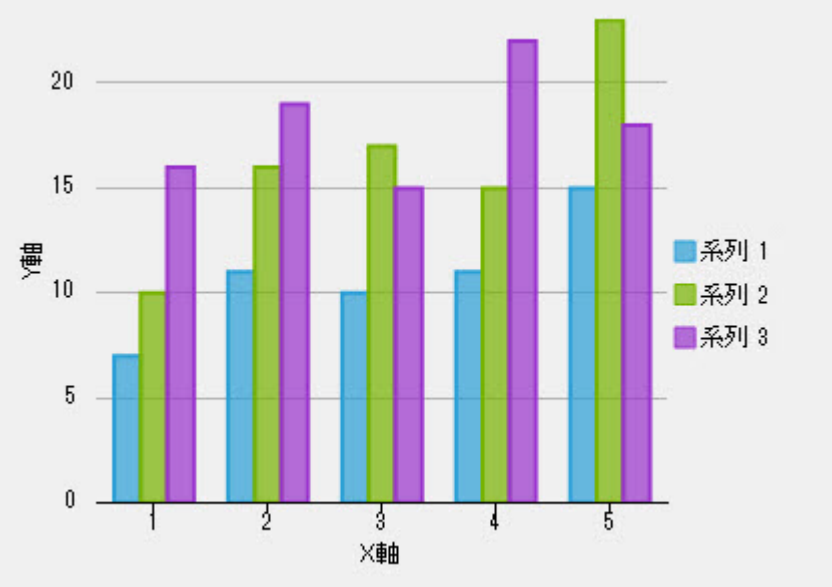
Highcontrast

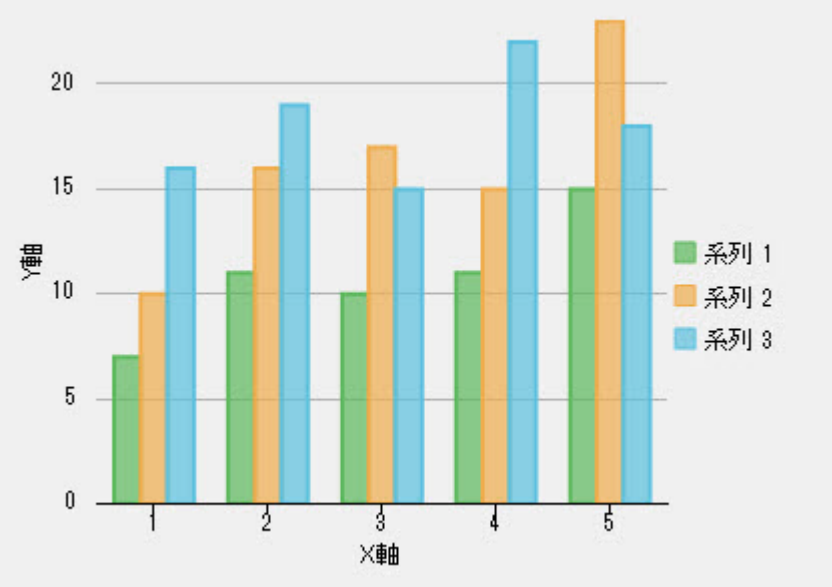
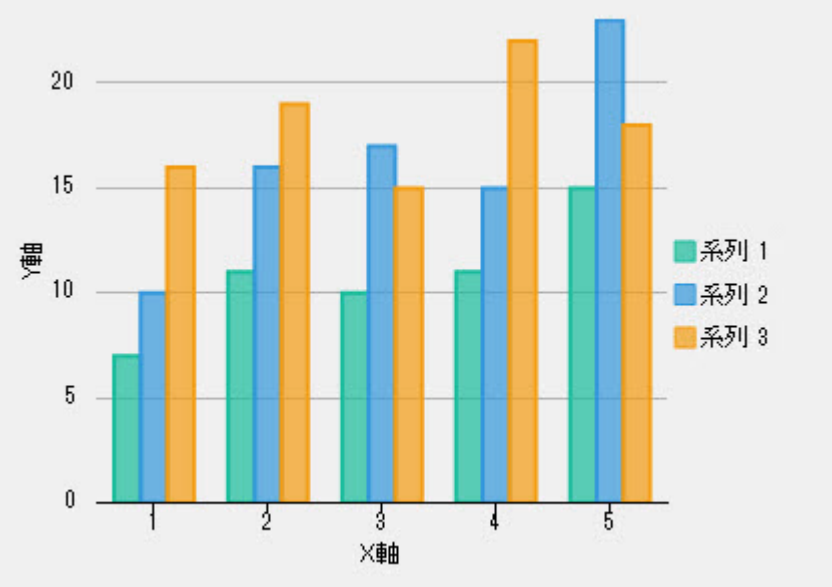
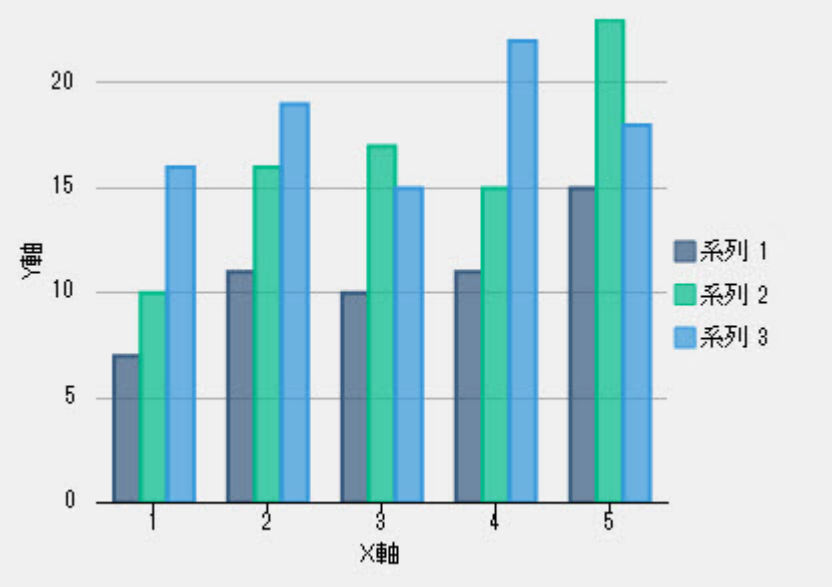


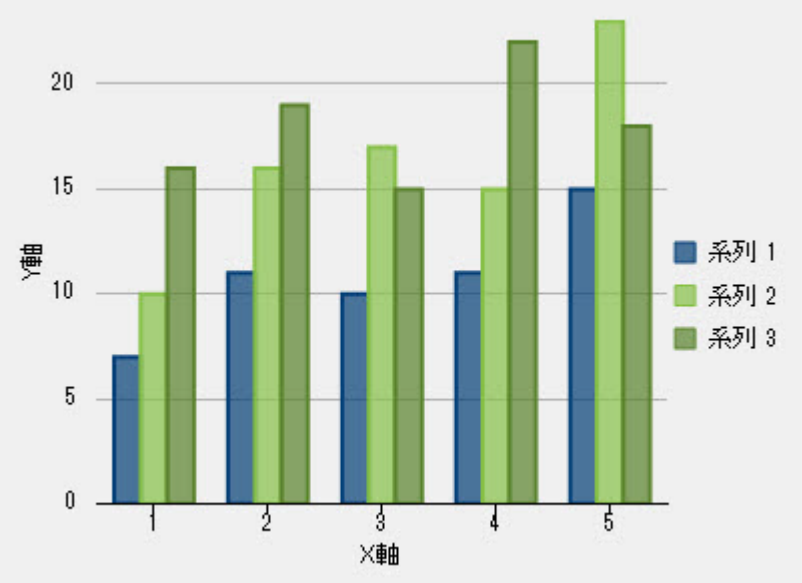
Light



Midnight	
Modern	
Organic	

Slate	
Zen	
Cyborg	

Superhero	
Flatly	
Darkly	

Cerulan	
Custom	現在指定されているパレットをカスタムグループにコピーします。

RGB 色の指定

RGB 要素で色を指定することもできます。この方法は、別の RGB 色と色を合わせる場合に便利です。RGB 色の値は、1 色に含まれる赤、緑、青の各要素の 16 進数値を組み合わせたものです。各要素の最小値は「00」、最大値は「ff」です。たとえば、「#ff00ff」はマゼンタ色を指定します（赤色の最大値、青色の最大値、緑色なしの組み合わせ）。

色相/彩度/輝度の指定

色は、RGB 要素で指定できるほか、色相、彩度、輝度で表すこともできます。色相、彩度、輝度は、RGB カラースキームの完全な構成情報です。色相は、赤/緑/青で構成されるカラーホイール内の特定の色調を表します。彩度は、灰色から純色までの色相の強度です。輝度は、色調の明るさまたは暗さです。

透明色の使用

チャート自身を除くすべての要素の背景と前景を「Transparent（透明）」にできます。

背景または前景が透明な場合は、その外側にある要素の色が背景に使用されます。たとえば、ヘッダーの背景を Transparent に設定すると、チャート自身の背景がヘッダーに使用されます。

つまり、要素の背景色が透明な場合、要素の背景は描画されません。また、要素の前景色が透明な場合、前景（たとえば、タイトルのテキスト）は描画されません。

透明色のプロパティは、設計時には、Visual Studio のプロパティウィンドウで Control、Header、Footer、Legend、ChartArea、および ChartLabels オブジェクトの［スタイル］ノードにあります。

フォント

さまざまなチャート要素に対してフォントをカスタマイズすると、チャートの印象を向上させることができます。要素のフォントサイズを調整して、チャート全体のサイズに対して最適なサイズを選択できます。

FlexChart でフォントを変更またはカスタマイズする場合、**ChartStyle** オブジェクトで提供されている次のプロパティを使用できます。

プロパティ	説明
FontFamily	フォントファミリを設定します。
FontSize	フォントサイズを設定します。
FontStretch	フォントストレッチを設定します。
FontStyle	フォントスタイルを設定します。
FontWeight	フォントウェイトを設定します。

系列のシンボルスタイル

要件によっては、チャート内の系列の外観をカスタマイズする必要があります。

FlexChart では、**SymbolMarker** プロパティと **SymbolSize** プロパティを使用して、チャート内の系列をカスタマイズできます。

SymbolMarker プロパティを使用すると、系列の各データポイントに使用されるマーカーの図形を設定できます。**SymbolSize** プロパティを使用すると、系列のレンダリングに使用されるシンボルのサイズ（ピクセル単位）を設定できます。

次の表に、これらのプロパティが各チャートタイプにどのように影響するかを示します。

値	SymbolMarker の効果	SymbolSize の効果
ChartType.Column	効果なし	効果なし
ChartType.Bar	効果なし	効果なし
ChartType.Line	効果なし	効果なし
ChartType.Scatter	シンボルマーカーを変更	シンボルサイズを変更
ChartType.LineSymbols	シンボルマーカーを変更	シンボルサイズを変更
ChartType.Area	効果なし	効果なし
ChartType.Spline	効果なし	効果なし
ChartType.SplineSymbols	シンボルマーカーを変更	シンボルサイズを変更
ChartType.SplineArea	効果なし	効果なし
ChartType.Bubble	シンボルマーカーを変更	効果なし
ChartType.Candlestick	効果なし	シンボルサイズを変更
ChartType.HighLowOpenClose	効果なし	シンボルサイズを変更

SymbolSize プロパティはバブルチャートに効果がありませんが、プロパティウィンドウの［オプション］ノードにある **BubbleMaxSize** プロパティと **BubbleMinSize** プロパティを設定することで、バブルチャートのバブルのサイズを変更できます。

エンドユーザー操作

チャートの機能については、いくつかの特別なツールを使用しないと対応できない特別な要件がある場合があります。

そのような要件に対応するために、**FlexChart** では、一連の変換メソッドと対話式の組み込みツールを提供しています。これらのツールは、アプリケーションをさらにカスタマイズおよび開発するために役立ちます。

エンドユーザー操作の詳細については、以下のセクションを参照してください。

- ツールチップ
- 軸スクロールバー
- 範囲セレクタ
- ラインマーカー

ツールチップ

ツールチップは、チャートのデータポイントまたは系列の上にマウスポインタを合わせると表示されるポップアップです。次のように、さまざまな状況でチャートデータに関する詳細で有益な情報を提供します。

- **単一系列のグラフ**: データ値と系列名がツールチップに表示されます。
- **複合チャート**: 1 つのカテゴリに対して複数の系列の複数のデータ値がツールチップに表示されます。
- **円グラフ**: セグメントの名前とパーセンテージまたは値がツールチップに表示されます。

FlexChart のデフォルトでは、ツールチップにデータポイントの Y 値が表示されます。ただし、定義済みのパラメータと書式を使用して、ツールチップにカスタムコンテンツを作成したり書式設定することができます。さらに、複合チャートを使用する場合は、共有ツールチップを作成することもできます。

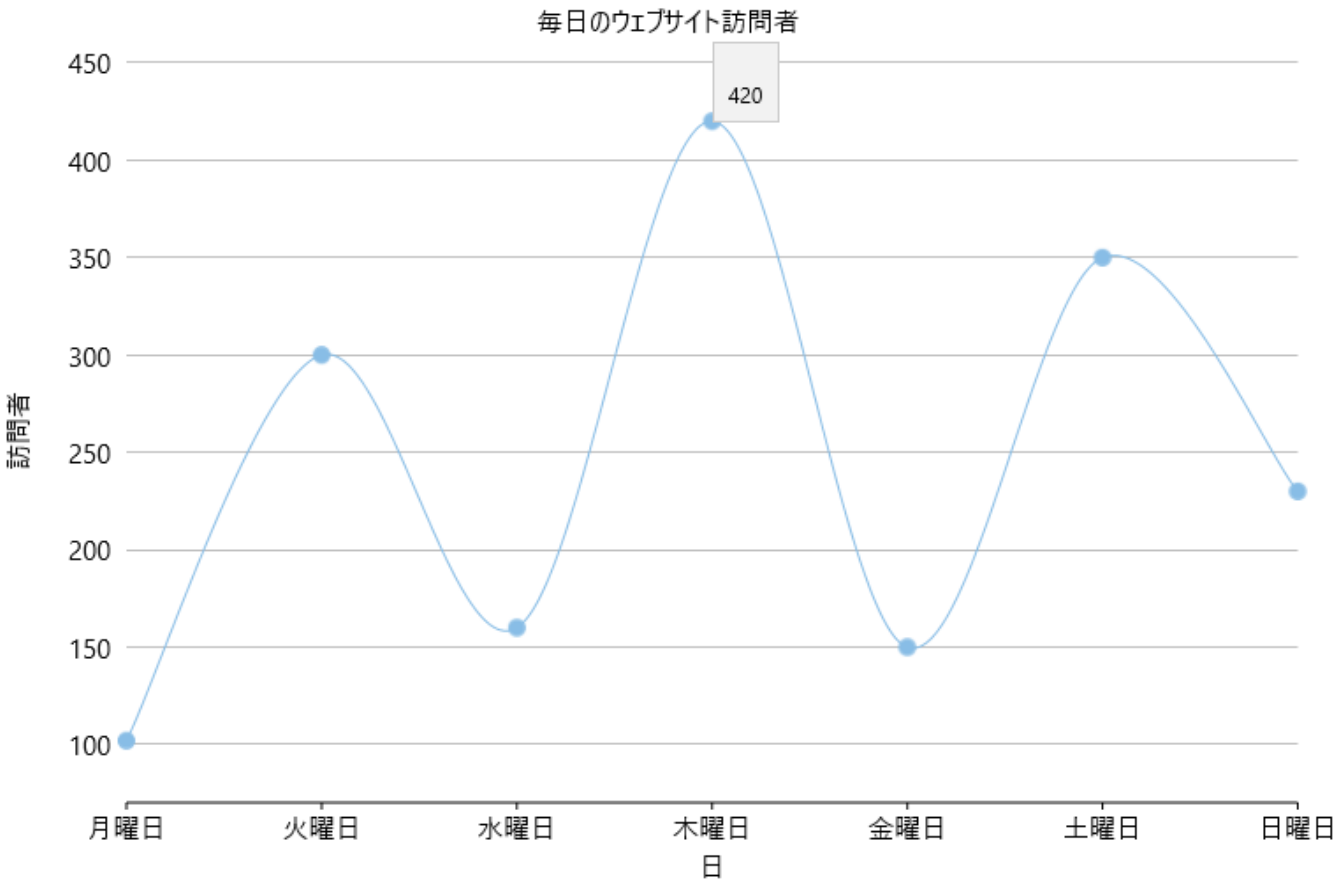
FlexChart のツールチップの詳細については、次のトピックを参照してください。

- デフォルトツールチップ
- ツールチップコンテンツのカスタマイズ
- ツールチップコンテンツの書式設定
- 共有ツールチップ

デフォルトツールチップ

FlexChart では、データポイントまたは系列にマウスポインタを合わせると、デフォルトのツールチップが表示されます。デフォルトのツールチップには、マウスポインタを合わせたデータポイントの Y 値が表示されます。ツールチップにカスタムコンテンツが存在しない場合は、基底のデータを使用して、デフォルトのツールチップが生成されます。

次の図は、データポイントのデータ値を示すデフォルトのツールチップです。



ツールチップコンテンツのカスタマイズ

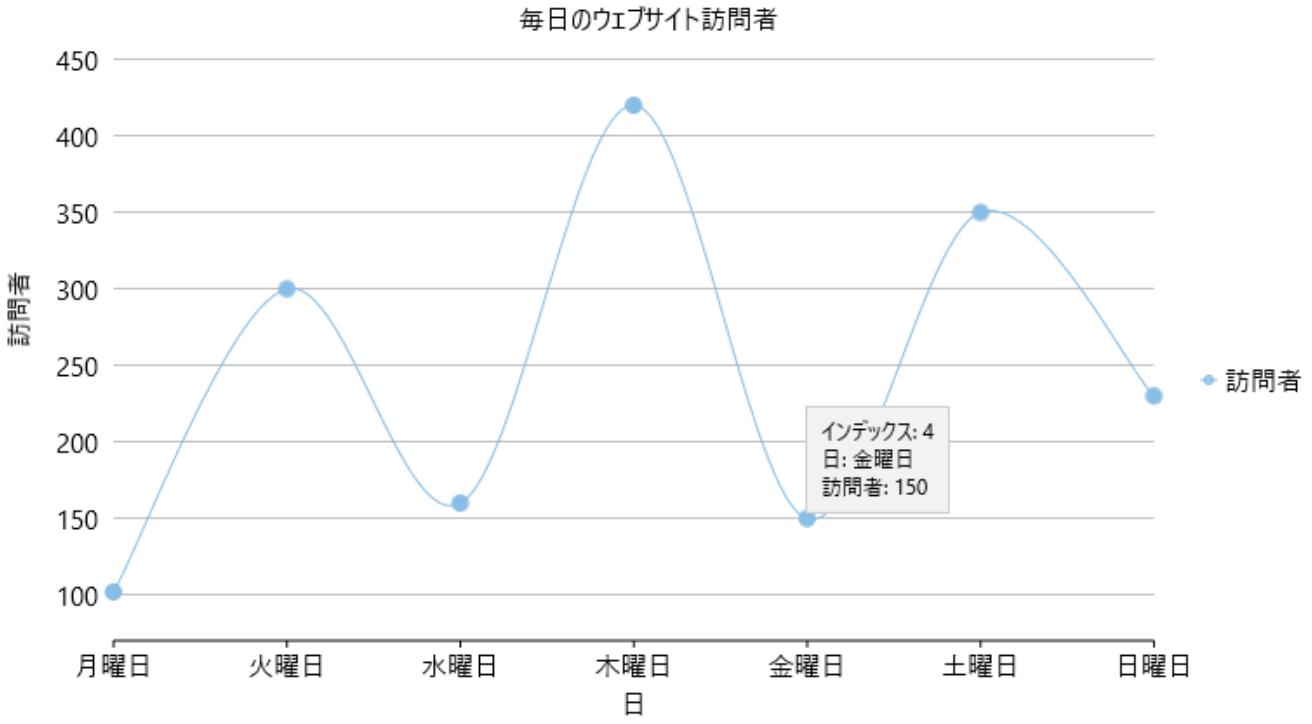
FlexChart では、ツールチップの **ToolTipContent** プロパティに定義済みのパラメータを設定することで、ツールチップコンテンツを簡単にカスタマイズできます。

ツールチップをカスタマイズするには、**FlexChartBase**クラスの ToolTipContent プロパティのテンプレート文字列に定義済みのパラメータを設定します。

次の表に、ツールチップコンテンツのカスタマイズに適用できる定義済みのパラメータを示します。

パラメータ	説明
x	データポイントの X 値を示します。
y	データポイントの Y 値を示します。
value	データポイントの Y 値を示します。
name	データポイントの X 値を示します。
seriesName	系列の名前を表示します。
pointIndex	データポイントのインデックスを示します。

次の図は、インデックスとデータポイント値を示すカスタマイズされたツールチップコンテンツです。



次のコードは、特定の週の毎日の Web サイトの訪問者数のデータを比較して表示します。このコードは、ToolTipContent プロパティを設定してツールチップコンテンツをカスタマイズする方法を示します。

XAML

```
<Chart:C1FlexChart Name="flexChart"
 ToolTipContent="インデックス: {pointIndex}&#13;日: {name}&#13;{seriesName}: {Visitors}">
</Chart:C1FlexChart>
```

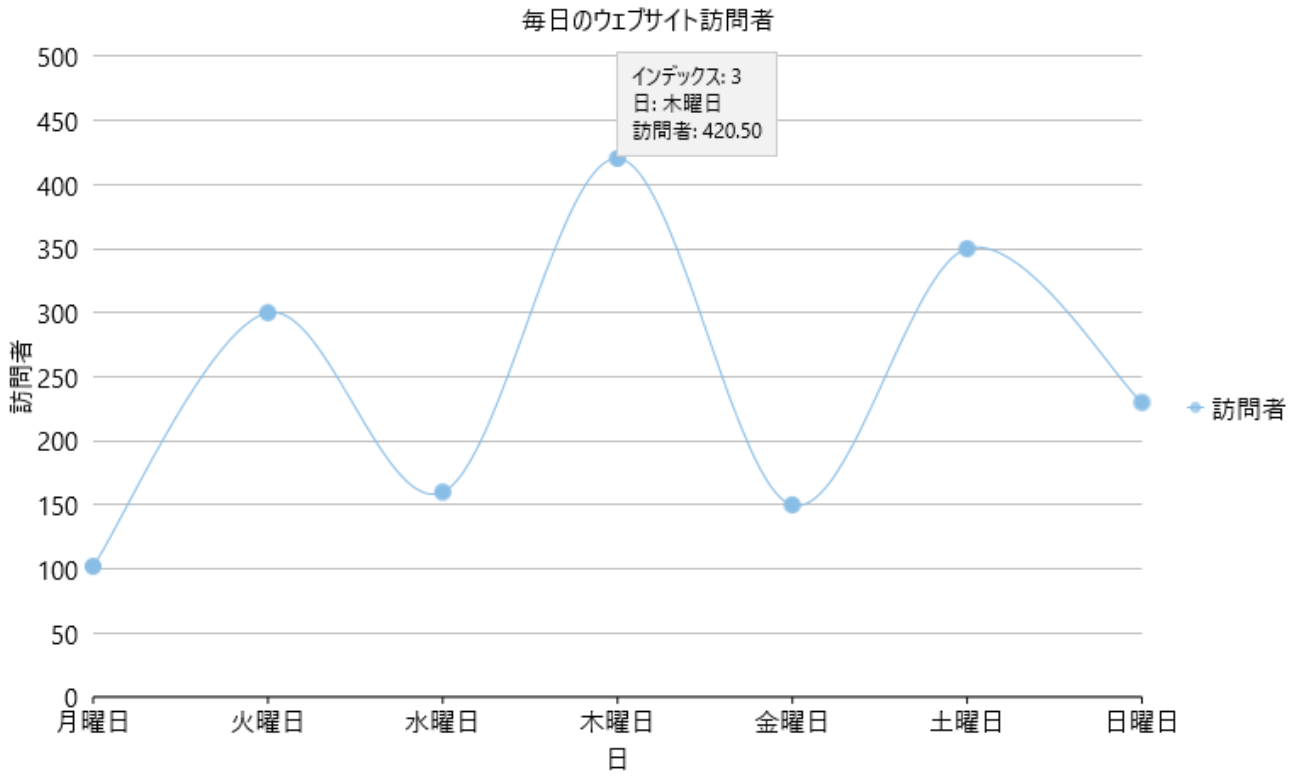
ツールチップコンテンツの書式設定

FlexChart では、数字の区切り文字、通貨記号、日時形式を表示して、ツールチップをさらに詳細に表示できます。

FlexChart では、標準およびカスタムの書式文字列を使用して、ツールチップのカスタムコンテンツを書式設定できます。書式文字列は、.NET に用意されているさまざまな数値および日時形式と同じです。

書式文字列の詳細については、数値および日時書式文字列を参照してください。

次の図は、インデックスと書式設定されたデータポイント値を示すカスタマイズされたツールチップコンテンツです。



次のコードは、特定の週の毎日の Web サイトの訪問者数のデータを比較して表示します。このコードは、ToolTipContent プロパティを設定してツールチップコンテンツを書式設定する方法を示します。

XAML

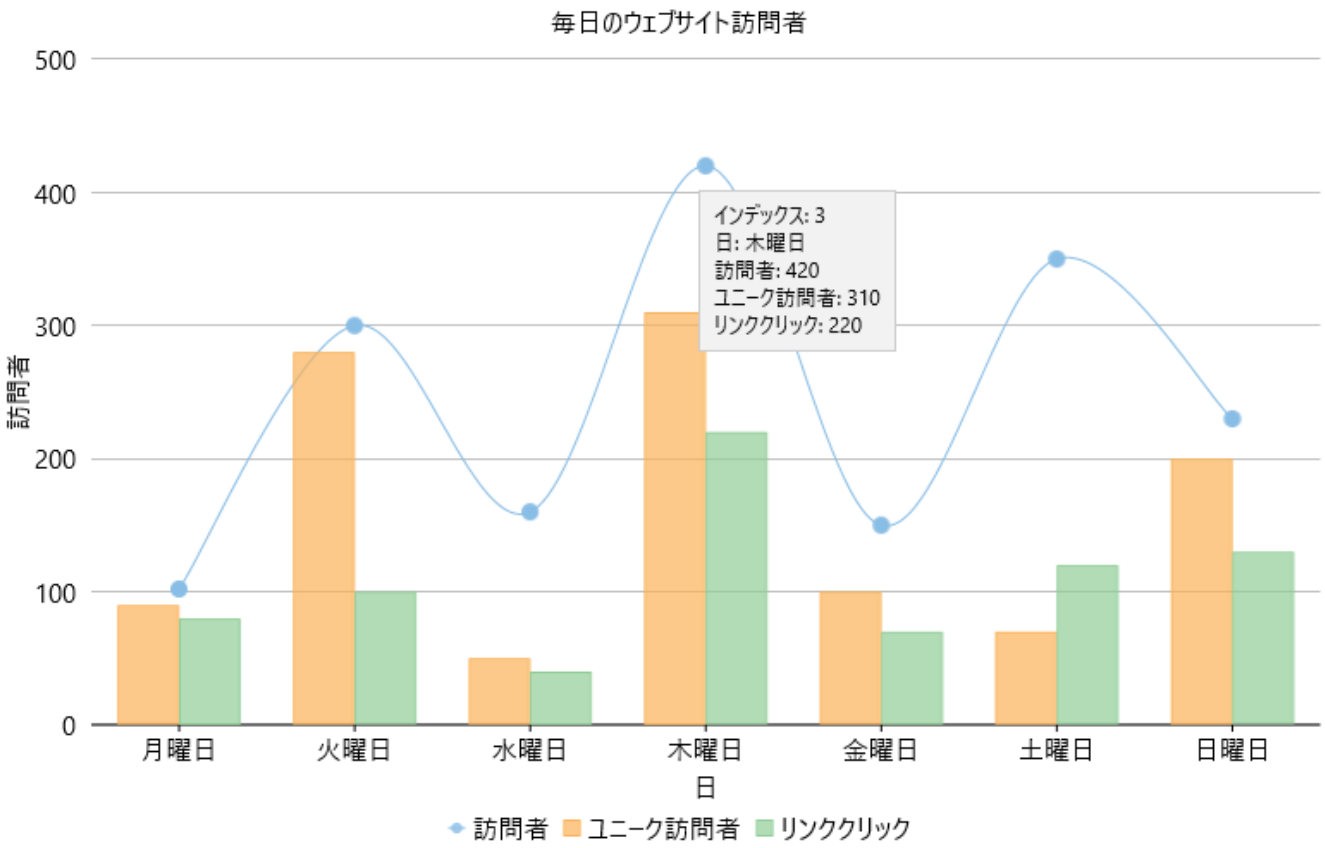
```
<Chart:C1FlexChart Name="flexChart"
   ToolTipContent="インデックス: {pointIndex}&#13;日: {name}&#13;{seriesName}: {Visitors:F}">
</Chart:C1FlexChart>
```

共有ツールチップ

共有ツールチップは、チャート内の 1 つの X 軸のすべてのデータ値を 1 つのツールチップに強調表示します。

複合チャートタイプを含むチャートでは、1 つのツールチップに共通の X 値に対する複数の Y 値を表示することがよくあります。このような場合は、**ToolTipContent** プロパティを適切に設定して、FlexChart ツールチップを共有ツールチップとして使用できます。

次の図は、1 つの X 値にすべての系列の Y 値を表示する共有ツールチップです。



次のコードは、特定の週の毎日の Web サイトの訪問者数のデータを比較して表示します。このコードは、ToolTipContent プロパティを設定してツールチップコンテンツを書式設定する方法を示します。

XAML

```
<Chart:C1FlexChart Name="flexChart"
   ToolTipContent="インデックス: {pointIndex}&#13;日: {name}&#13;訪問者:
   {Visitors}&#13;ユニーク訪問者: {UniqueVisitors}&#13;リンククリック: {LinkClick}">
</Chart:C1FlexChart>
```

軸スクロールバー

軸スクロールバーを使用すると、軸の値をスクロールすることで、特定の範囲を選択できます。チャートに大量の値またはデータが存在すると、コンパクトなユーザーインタフェースでは特に、データの解釈が難しくなります。軸スクロールバーは、特定の範囲内にあって関連性の高いデータを容易に解釈できるようにすることで、この問題を解決します。

FlexChart では、主軸（X 軸と Y 軸）と第 2 軸の両方に軸スクロールバーを追加できます。軸に軸スクロールバーを追加するには、**C1.Xaml.Chart.Interaction.C1AxisScrollbar** クラスのインスタンスを作成する必要があります。

C1AxisScrollbar クラスには、Boolean 値を受け取る **ScrollButtonsVisible** プロパティがあり、増やすボタンと減らすボタンの表示/非表示を設定できます。スクロールバーの現在の下限値と上限値を設定するには、C1RangeSlider クラスで提供されている **LowerValue** プロパティと **UpperValue** プロパティをそれぞれ使用します。下限値と上限値は、スクロールバーがサイズ変更されたり移動されると変化します。LowerValue プロパティと UpperValue プロパティのいずれかが変化すると、C1RangeSlider クラスで提供される **ValueChanged** イベントが発生します。

次のコードスニペットを参照してください。

XAML

```
<Chart:Axis.Scrollbar>
    <Interaction:C1AxisScrollbar x:Name="axisYScrollbar"
    Width="30" ScrollButtonsVisible="False"/>
</Chart:Axis.Scrollbar>
```

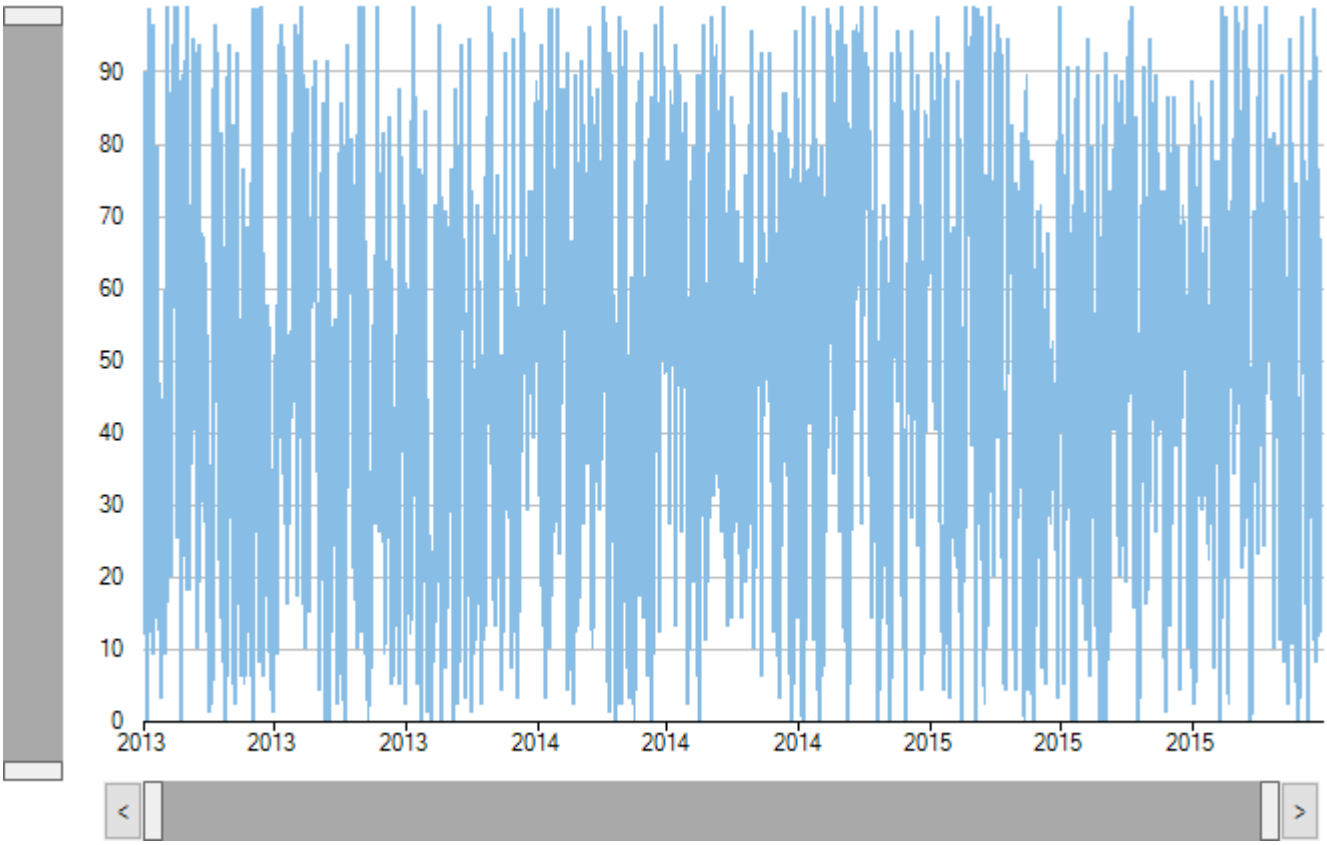
コード

C# copyCode

```
public class AxisScrollbarModel
{
    Random rnd = new Random();

    public List<DataItem> Data
    {
        get
        {
            var pointsCount = rnd.Next(1, 30);
            var pointsList = new List<DataItem>();
            for (DateTime date = new DateTime(DateTime.Now.Year - 3, 1, 1);
                 date.Year < DateTime.Now.Year; date = date.AddDays(1))
            {
                pointsList.Add(new DataItem()
                {
                    Date = date,
                    Series1 = rnd.Next(100)
                });
            }

            return pointsList;
        }
    }

    public string Description
    {
        get
        {
            return Strings.Description;
        }
    }

    public string Title
    {
        get
        {
            return Strings.Title;
        }
    }
}
```

VB copyCode

```
Public Class AxisScrollbarModel
    Private rnd As New Random()

    Public ReadOnly Property Data() As List(Of DataItem)
        Get
            Dim pointsCount As Object = rnd.[Next](1, 30)
            Dim pointsList As New List(Of DataItem)()
            Dim [date] As New DateTime(DateTime.Now.Year - 3, 1, 1)
            While [date].Year < DateTime.Now.Year
                pointsList.Add(New DataItem() With {
                    .[date] = [date],
                    .Series1 = rnd.[Next](100)
                })
                [date] = [date].AddDays(1)
            End While

            Return pointsList
        End Get
    End Property

    Public ReadOnly Property Description() As String
        Get
            Return Strings.Description
        End Get
    End Property

    Public ReadOnly Property Title() As String
        Get
            Return Strings.Title
        End Get
    End Property
End Class
```



範囲セレクタ

チャートをスクロールする方法としては従来からスクロールバーがありますが、範囲セレクタはより新しい手法として、選択した範囲がデータ全体のどの位置にあるかを視覚化できるようにします。

FlexChart の範囲セレクタでは、下限値スクロールボックスと上限値スクロールボックスを使用して数値データの範囲を選択できます。これらのスクロールボックスは、範囲の開始値と終了値を定義します。範囲バーでスクロールボックスを左（または下）にドラッグすると値が減少し、右（または上）にドラッグすると値が増加します。

FlexChart に範囲セレクタを追加するには、**C1.Xaml.Chart.Interaction.C1RangeSelector** クラスのインスタンスを作成します。**C1RangeSelector** クラスは、**C1.Xaml.C1RangeSlider** クラスを継承します。C1RangeSlider で提供される **LowerValue** プロパティと **UpperValue** プロパティを使用して、範囲セレクタの下限値と上限値をそれぞれ設定できます。LowerValue プロパティと UpperValue プロパティのいずれかが変更されると、**ValueChanged** イベントが発生します。

範囲セレクタを水平方向または垂直方向に設定するには、**Orientation** プロパティを使用します。このプロパティが変更されると、**OrientationChanged** イベントが発生します。

次に、実装方法を示すコードスニペットを示します。

XAML

```
<Chart:C1FlexChart.Layers>
    <Interaction:C1RangeSelector x:Name="rangeSelector" ValueChanged="OnRangeSelectorValueChanged"/>
</Chart:C1FlexChart.Layers>
```

コード

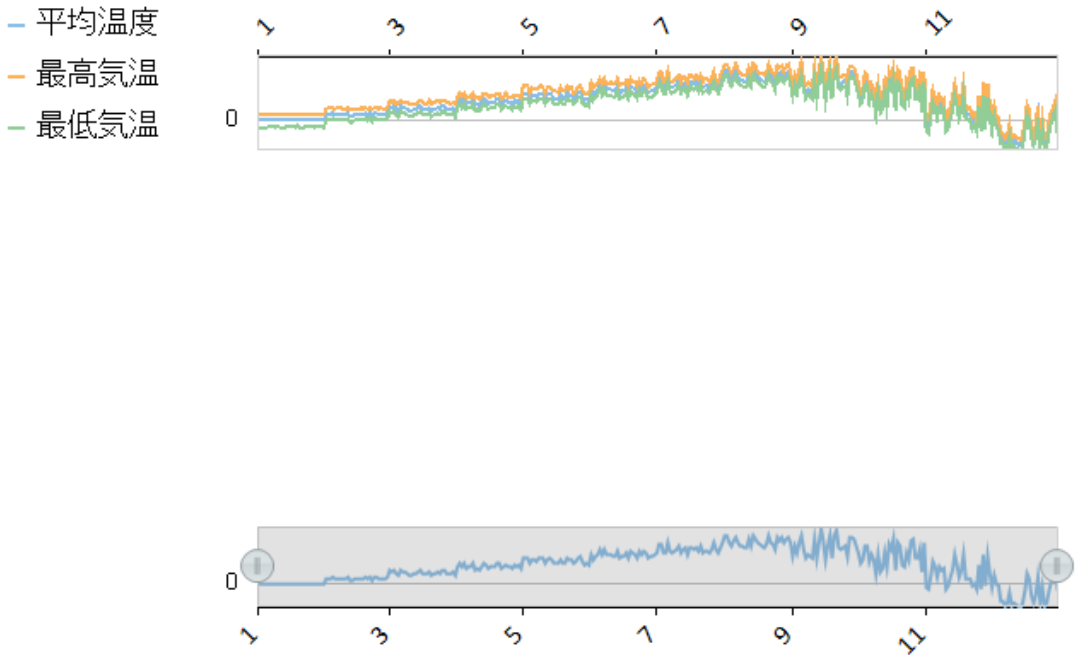
C# copyCode

```
void OnRangeSelectorValueChanged(object sender, System.EventArgs e)
{
    chartPrecipitation.AxisX.Min = rangeSelector.LowerValue;
    chartPrecipitation.AxisX.Max = rangeSelector.UpperValue;
    chartPressure.AxisX.Min = rangeSelector.LowerValue;
```

```
    chartPressure.AxisX.Max = rangeSelector.UpperValue;
    chartTemperature.AxisX.Min = rangeSelector.LowerValue;
    chartTemperature.AxisX.Max = rangeSelector.UpperValue;
}
```

VB copyCode

```
Sub OnRangeSelectorValueChanged(sender As Object, e As System.EventArgs)
    chartPrecipitation.AxisX.Min = rangeSelector.LowerValue
    chartPrecipitation.AxisX.Max = rangeSelector.UpperValue
    chartPressure.AxisX.Min = rangeSelector.LowerValue
    chartPressure.AxisX.Max = rangeSelector.UpperValue
    chartTemperature.AxisX.Min = rangeSelector.LowerValue
    chartTemperature.AxisX.Max = rangeSelector.UpperValue
End Sub
```



ラインマーカー

ラインマーカーは、プロット上で水平線または垂直線をドラッグすることで、チャート内の特定の位置の正確なデータ値を添付ラベルを使用して表示します。折れ線グラフや面グラフに多数のデータがある場合や、複数の系列のデータを 1 つのラベルに表示したい場合などに便利です。Drag、Move などの組み込み操作を使用すると、ラインマーカーをドラッグしてチャート内のデータポイントをより正確に選択できます。

FlexChart でラインマーカーを作成して使用するには、**C1.Xaml.Chart.Interaction.C1LineMarker** クラスのインスタンスを作成する必要があります.

C1LineMarker で提供されている **Lines** プロパティを使用して、LineMarker の線の表示/非表示を設定します。Lines プロパティは、**LineMarkerLines** 列挙に含まれる次の値を受け取ります。

- **Both**：垂直線と水平線の両方を表示します。
- **Horizontal**：水平線を表示します。
- **Vertical**：垂直線を表示します。
- **None**：線を表示しません。

C1LineMarker クラスでは、ラインマーカーの配置を設定するための **Alignment** プロパティも提供されています。さらに、**Interaction** プロパティを **LineMarkerInteraction** 列挙に含まれる次の値のいずれかに設定して、ラインマーカーの操

作モードを設定することができます。

- **Drag**：ユーザーが線をドラッグすると、ラインマーカーが移動します。
- **Move**（デフォルト）：ラインマーカーはポインタと共に移動します。
- **None**：ユーザーがクリックして位置を指定します。

Interaction プロパティを Drag に設定する場合は、**DragContent** プロパティと **DragLines** プロパティを設定して、ラインマーカーの線にリンクされた内容や値がドラッグ可能かどうかを指定する必要があります。Furthermore, you can set the initial position of the line marker relative to the plot area with the help of **VerticalPosition** and **HorizontalPostion** properties. The acceptable range for these properties is [0,1].

次のコードスニペットに、実装方法を示します。

XAML

```
<Chart:C1FlexChart.Layers>
    <Interaction:C1LineMarker x:Name="lineMarker" DragThreshold="30"
    PositionChanged="OnLineMarkerPositionChanged"/>
</Chart:C1FlexChart.Layers>
```

コード

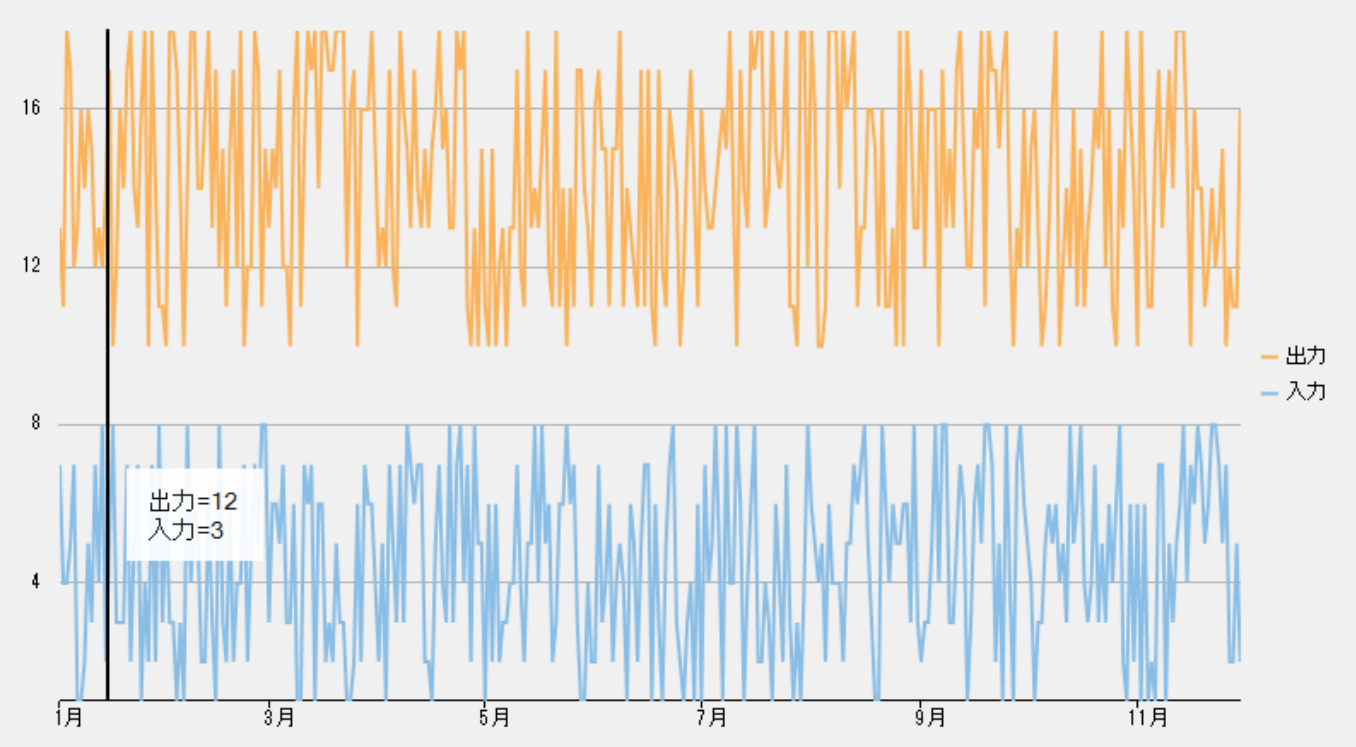
C# copyCode

```
private void OnLineMarkerPositionChanged(object sender, PositionChangedArgs e)
{
    if (flexChart != null)
    {
        var info = flexChart.HitTest(e.Position);
        int pointIndex = info.PointIndex;
        var tb = new TextBlock();
        if (info.X == null)
            return;

        tb.Inlines.Add(new Run()
        {
            Text = info.X.ToString()
        });
        for (int index = 0; index < flexChart.Series.Count; index++)
        {
            var series = flexChart.Series[index];
            var value = series.GetValues(0)[pointIndex];
            var fill = (int)((IChart)flexChart).GetColor(index);
            string content = string.Format("{0}{1} = {2}", "\n",
            series.SeriesName, value.ToString());
            tb.Inlines.Add(new Run()
            {
                Text = content,
                Foreground = new SolidColorBrush() { Color = FromArgb(fill) }
            });
        }
        tb.IsHitTestVisible = false;
        lineMarker.Content = tb;
    }
```

```
}
```

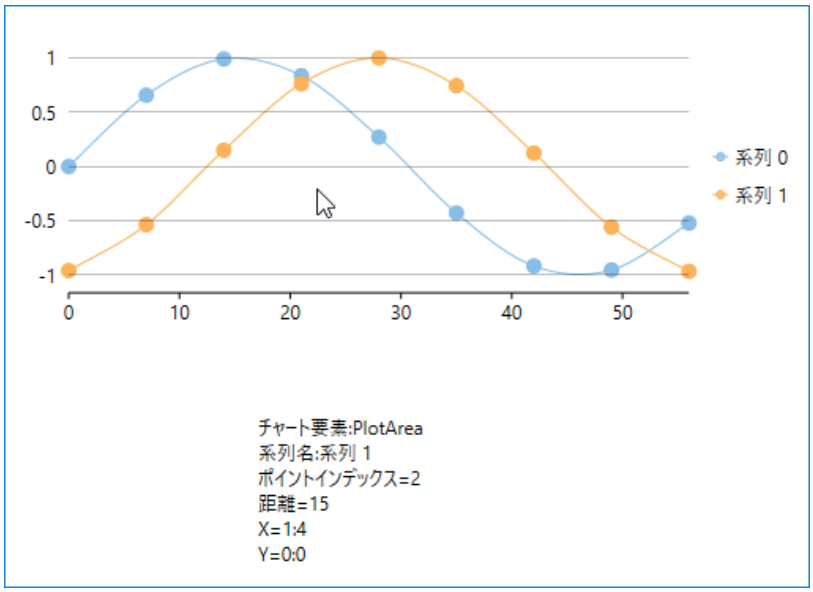
VB copyCode

```
    Private Sub OnLineMarkerPositionChanged(sender As Object, e As
PositionChangedArgs)
        If flexChart IsNot Nothing Then
            Dim info As HitTestInfo = flexChart.HitTest(e.Position)
            If info.Item Is Nothing Then
                Return
            End If
            Dim pointIndex As Integer = info.PointIndex
            Dim tb As New TextBlock()
            tb.Inlines.Add(New Run() With {
                .Text = info.X.ToString()
            })
            Dim index As Integer = 0
            While index < flexChart.Series.Count
                Dim series As Series = flexChart.Series(index)
                Dim value As Object = series.GetValues(0)(pointIndex)
                Dim fill As Integer = CType((CType(flexChart,
IChart)).GetColor(index), Integer)
                Dim content As String = String.Format("{0}{1} = {2}",
vbLf, series.SeriesName, value.ToString())
                tb.Inlines.Add(New Run() With {
                    .Text = content,
                    .Foreground = New SolidColorBrush() With {
                        .Color = FromArgb(fill)
                    }
                })
                index += 1
            End While
            tb.IsHitTestVisible = False
            lineMarker.Content = tb
        End If
    End Sub
```



ヒットテスト

FlexChart は、ヒットテストをサポートし、実行時にコントロールの特定のポイントの情報を取得できます。ポイントされた座標に関する情報を再利用して、チャートデータをドリルダウンしたり、アラートを設定するなどのユーザー操作機能を実装できます。また、ユーザーは、マウス操作またはタッチ操作を使用して、指定されたチャート要素に関する関連情報を取得できます。



FlexChart は、**HitTest()** メソッドを使用したヒットテストをサポートします。このメソッドは、指示されたエンティティの位置（座標）を取得します。そして、ポインタ位置に関する次の情報を提供する **HitTestInfo** クラスのオブジェクトを返します。

- ポインタの位置にあるチャート要素
- ポインタがプロット領域内にある場合は、チャート内の最も近いデータポイントからポインタ位置までの距離。ポインタがプロット領域外にある場合は、距離として Double.Nan が返されます。
- 最も近いデータポイントに対応するデータオブジェクト
- 最も近いデータポイントのインデックス
- 最も近いデータポイントが属する系列名
- 最も近いデータポイントの X 値
- 最も近いデータポイントの Y 値

HitTest() メソッドに渡されるマウス座標はピクセル単位で、フォームの左上隅を基準とします。

この例では、FlexChart コントロールの MouseMove イベントで **HitTest()** メソッドが呼び出されます。ここで、ポインタ位置のポイント座標がパラメータとして HitTest() メソッドに渡されます。

FlexChart でヒットテストを有効にするには、次の手順に従います。

1. **データ連結 FlexChart コントロールの追加**
2. **マウスまたはタッチイベントのサブスクライブ**
3. **マウスまたはタッチイベントハンドラでのチャートの HitTest メソッドの呼び出し**
4. **HitTestInfo オブジェクトから返された情報の使用**

先頭に戻る

1. **データ連結 FlexChart コントロールの追加**

   以下のコードスニペットに示すように、アプリケーションに FlexChart コントロールのインスタンスを追加し、適切なデータソースに連結します。

   - **Xaml**

```
        <Chart:C1FlexChart x:Name="flexChart"
                           HorizontalAlignment="Left"
                           Height="220" Margin="70,25,0,0"
                           VerticalAlignment="Top" Width="455"
                           Binding="YVals" BindingX="XVals" ChartType="SplineSymbols" >
            <Chart:C1FlexChart.Series>
                <Chart:Series x:Name="series0" SeriesName="系列0"/>
                <Chart:Series x:Name="series1" SeriesName="系列1" />
            </Chart:C1FlexChart.Series>
        </Chart:C1FlexChart>
        <TextBlock x:Name="tbPosition1" Margin="155,250,255,185"/>
    </Grid>
</Page>
```

先頭に戻る

2. **マウスまたはタッチイベントのサブスクライブ**

   以下のコードスニペットに示すように、ポインタ座標を取得するためにマウスイベントをサブスクライブします。

   - **Xaml**

```
    <Chart:C1FlexChart x:Name="flexChart"
                       HorizontalAlignment="Left"
                       Height="220" Margin="70,25,0,0"
                       VerticalAlignment="Top" Width="455"
                       Binding="YVals" BindingX="XVals" ChartType="SplineSymbols"
                       PointerPressed="flexChart_PointerPressed" >
    <Chart:C1FlexChart.Series>
        <Chart:Series x:Name="series0" SeriesName="系列0"/>
        <Chart:Series x:Name="series1" SeriesName="系列1" />
    </Chart:C1FlexChart.Series>
</Chart:C1FlexChart>
```

先頭に戻る

3. **マウスまたはタッチイベントハンドラでのチャートの HitTest メソッドの呼び出し**

   以下のコードスニペットに示すように、対応するイベントハンドラで、**HitTest()** メソッドを呼び出し、取得したマウスポインタ座標を渡します。

   - **C#**

```
private void flexChart_PointerPressed(object sender, PointerRoutedEventArgs e)
{
    HitTestOnFlexChart(e.GetCurrentPoint(flexChart).Position);
}
```

先頭に戻る

4. **HitTestInfo オブジェクトから返された情報の使用**

   次に、**HitTestInfo** オブジェクトで返されたマウスポインタの位置に関する情報を再利用できます。たとえば、以下のコードスニペットでは、**HitTestInfo** オブジェクトで返された値が文字列に変換され、TextBlock に表示されます。

   - **C#**

```
void HitTestOnFlexChart(Point p)
{
    // マウスオーバーまたはタッチ時のチャート要素に関する情報を表示します
    var ht = flexChart.HitTest(p);
    var result = new StringBuilder();
    result.AppendLine(string.Format("チャート要素: {0}", ht.ChartElement));
```

```
        if (ht.Series != null)
            result.AppendLine(string.Format("系列名: {0}", ht.Series.Name));
        if (ht.PointIndex > 0)
            result.AppendLine(string.Format("ポイントインデックス= {0:0}", ht.PointIndex));
        if (ht.Distance > 0)
            result.AppendLine(string.Format("距離= {0:0}", ht.Distance));
        if (ht.X != null)
            result.AppendLine(string.Format("X= {0:0:0}", ht.X));
        if (ht.Y != null)
            result.AppendLine(string.Format("Y= {0:0:0}", ht.Y));
        tbPosition1.Text = result.ToString();
    }
```

先頭に戻る

アニメーション

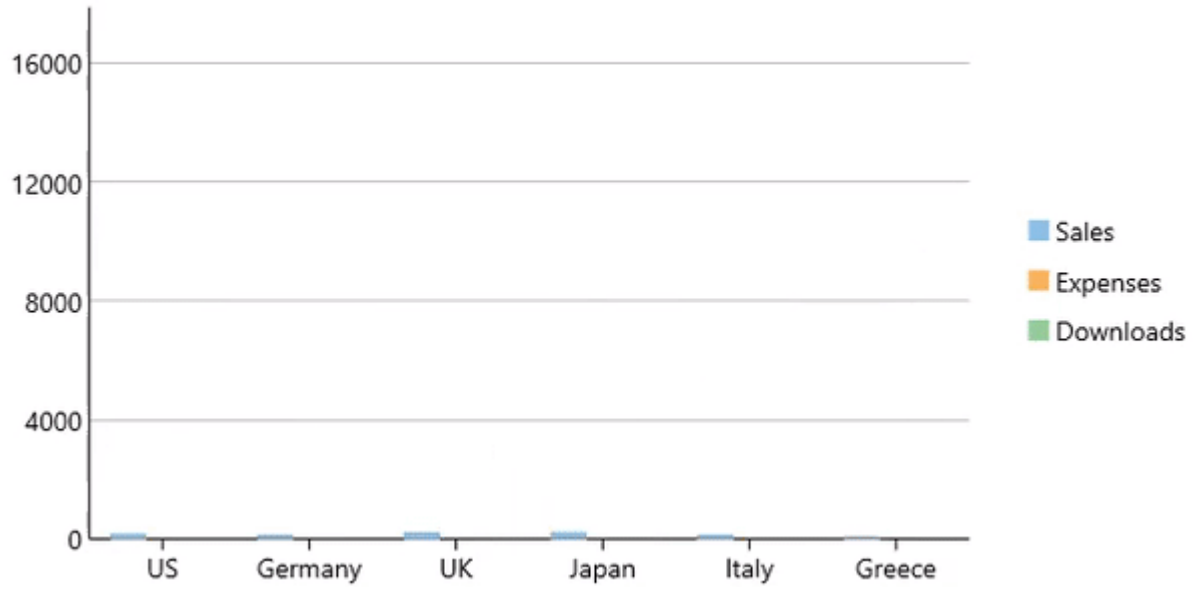
FlexChart allows you to control how the animation is applied to each series and series elements. It allows you to enable chart animation effects through a combination of different properties available in the FlexChart class. These properties allow you to apply duration, delay and an easing function for each animation. The animation effects are applied in one of the two scenarios, either while loading the chart for the first time or while the chart is redrawn after modifications.

The FlexChart control supports two basic scenarios for animation.

- **Load**
- **Update**

Load

Implement animation when the data is first loaded into the chart, for example when a new series is added to the FlexChart, you can apply animation properties. The following image shows how animation works while loading the FlexChart control.



Use the following code to implement animation while loading the FlexChart control.

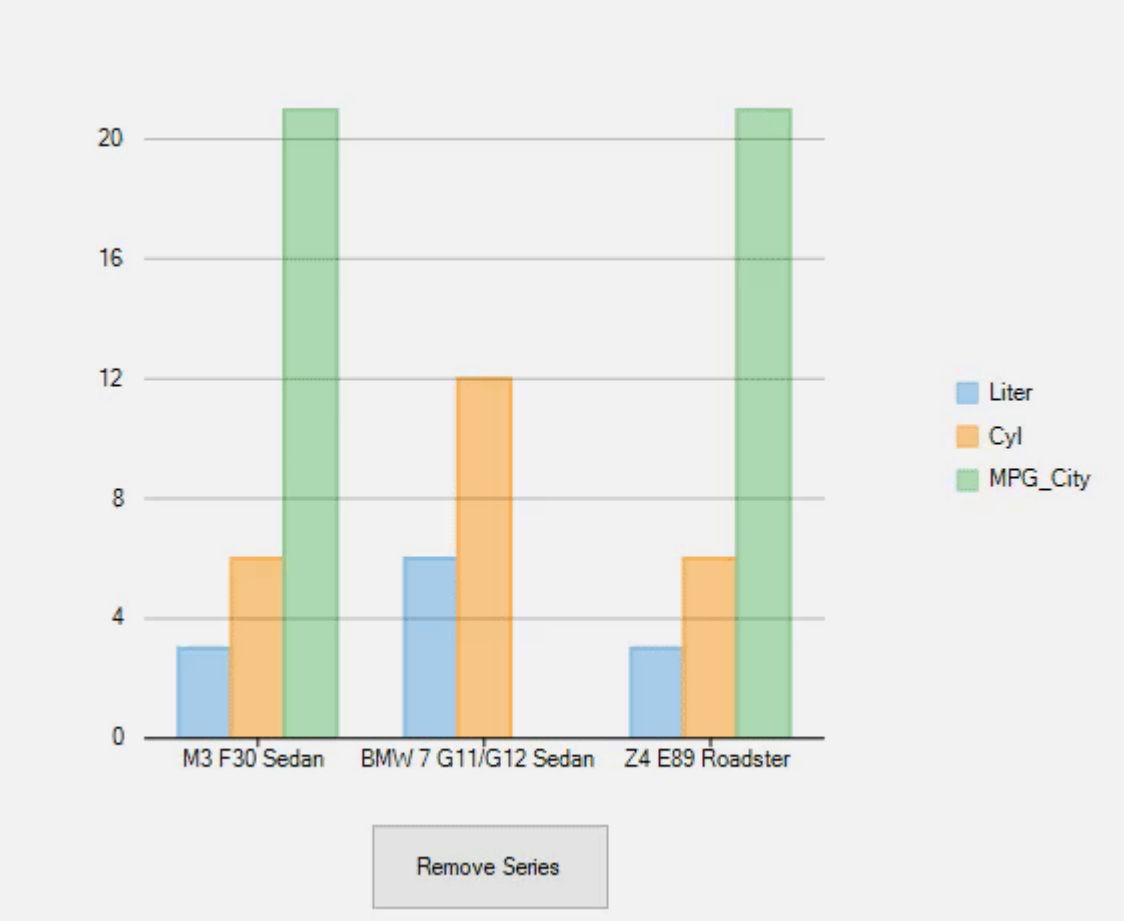
C#

```
// アニメーション
flexChart1.AnimationSettings = C1.Chart.AnimationSettings.Load;
flexChart1.AnimationUpdate.Easing = C1.Chart.Easing.Linear;
flexChart1.AnimationUpdate.Duration = 500;
```

```
flexChart1.AnimationLoad.Type = C1.Chart.AnimationType.Series;
```

Update

Implement animation when existing data is modified in the chart, for example when a new series is added or removed from the FlexChart control, you can apply animation properties. The following image shows how animation works while updating the FlexChart control.



Use the following code to implement animation while loading the FlexChart control.

C#

```
// アニメーション
flexChart1.AnimationSettings = C1.Chart.AnimationSettings.Update;
flexChart1.AnimationUpdate.Easing = C1.Chart.Easing.Linear;
flexChart1.AnimationUpdate.Duration = 500;
flexChart1.AnimationUpdate.Type = C1.Chart.AnimationType.All;

private void button1_Click(object sender, EventArgs e)
{
   //FlexChartを更新します
   flexChart1.BeginUpdate();
   flexChart1.Series.RemoveAt(0);
   flexChart1.EndUpdate();
}
```

Every animation scenario has its own set of options that can be set independently in the FlexChart control. These options include various properties describing the corresponding animation.

To implement animation in the FlexChart control, you need make use of the following properties.

1. **AnimationSettings** -This property allows the user to apply settings for the animation. It allows the user to specify that when to apply animation in the FlexChart control. This property accepts values from the **AnimationSettings** enumeration provided by the FlexChart class. The AnimationSettings enumeration has special flags to control axes animation (smooth transition) so that you can enable or disable smooth axis transition for loading or updating data.

2. AnimationOptions - The **AnimationLoad** and **AnimationUpdate** properties includes the following options.

    1. **Duration**: This property allows you to set the duration of animation in the FlexChart control. This property accepts an integer value which defines duration in milliseconds.

    2. **Easing**: This property allows the user to set different type of easing functions on the FlexChart control. This property accepts values from the **Easing** enumeration provided by the C1.Chart namespace.

    3. **Type**: This property allows you to set the animation type on the FlexChart control. This property accepts the following values from the **AnimationType** enumeration provided by the C1.Chart namespace.

        - **All**: All plot elements animate at once from the bottom of the plot area.
        - **Series**: Each series animates one at a time from the bottom of the plot area.
        - **Points**: The plot elements appear one at a time from left to right.

FlexChart の要素

チャートの要素をカスタマイズして、見栄えがよく本格的な外観のチャートを作成できます。

FlexChart は、軸、凡例、およびタイトルで構成されています。これらの要素については、既に「FlexChart の基本」で簡単に説明しました。

以下のセクションでは、FlexChart のこれらの要素のカスタマイズに焦点を当てます。

- FlexChart の軸
- FlexChart の軸ラベル
- 注釈
- FlexChart の凡例
- FlexChart の系列
- FlexChart のデータラベル
- 複数のプロット領域

FlexChart の軸

チャートには、一般に、データを測定および分類するために、垂直軸（Y 軸）と水平軸（X 軸）の 2 つの軸があります。垂直軸は値軸とも呼ばれ、水平軸はカテゴリ軸とも呼ばれます。

どのチャートの軸も同じ方法で描画されるわけではありません。たとえば、散布図やバブルチャートでは、垂直軸には数値が表示され、水平軸には離散的または連続的な数値データが表示されます。リアルタイムの例として、さまざまな年齢層別のインターネット利用率（週あたり時間数）をプロットする場合を考えます。この場合、2 つの項目はどちらも数値で、X 軸と Y 軸の数値に対応してデータポイントがプロットされます。

折れ線グラフ、縦棒グラフ、横棒グラフ、面グラフなど、他のチャートは、垂直軸に数値を表示し、水平軸にカテゴリを表示します。リアルタイムの例として、さまざまな地域別のインターネット利用率（週あたり時間数）をプロットする場合を考えます。この場合は、テキストで表されるカテゴリとして地域が水平軸にプロットされます。

ただし、FlexChart は、横棒グラフ、折れ線グラフ、面グラフの場合でも、X 軸と Y 軸の両方に数値を表示できる高い柔軟性を有しています。また、異なるタイプの値を表示する場合に、追加の設定を行う必要がありません。

FlexChart の軸は、**Axis** クラスで表されます。FlexChart の主軸には、**AxisX** プロパティと **AxisY** プロパティを使用してアクセ

スできます。

第 1 X 軸は下部に水平方向にレンダリングされ、第 1 Y 軸は左側に垂直方向にレンダリングされます。ただし、主軸をカスタマイズしたり、複数の軸を使用することで、この規則にも例外を設けることができます。

FlexChart を使用する際、目盛りマークや軸ラベルの外観を変更できます。軸の値間の単位数を指定することで、X 軸と Y 軸の軸ラベルの数を減らすことができます。さらに、ラベルの配置や方向を変更したり、表示される数値の書式を変更することができます。必要に応じて、軸のスタイルを設定したり、位置を変更することもできます。

以下のセクションでは、FlexChart の軸に対して行うことができるさまざまなカスタマイズや変更について説明します。

軸の位置

FlexChart では、**Position** プロパティを使用して、軸の位置を変更できます。

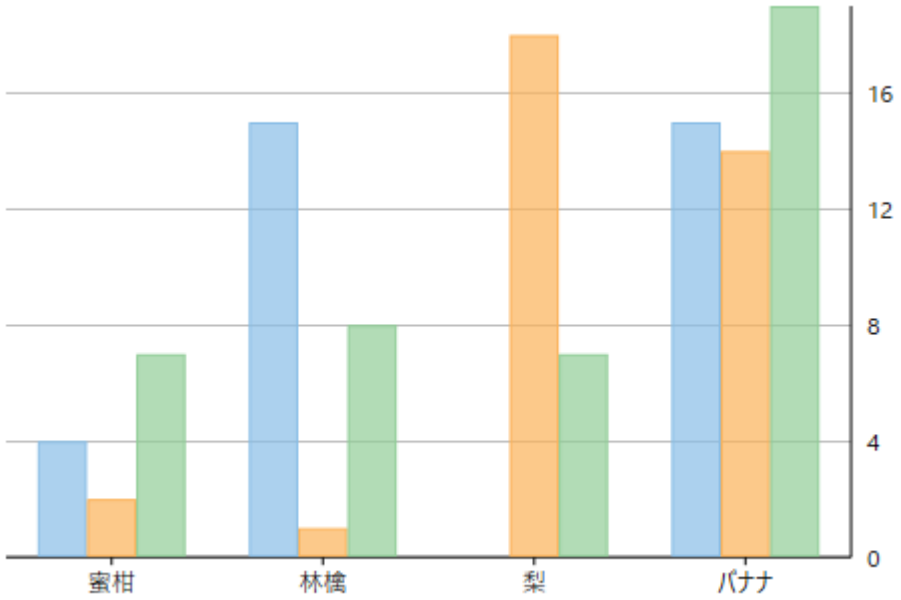
軸の **Position** プロパティは、**Position** 列挙の次の値に設定できます。

プロパティ	説明
Position.Auto	項目を自動的に配置します。
Position.Bottom	項目を下に配置します。
Position.Left	項目を左に配置します。
Position.None	項目を非表示にします。
Position.Right	項目を右に配置します。
Position.Top	項目を上に配置します。

次にサンプルコードを示します。

- **XAML**

```
<Chart:C1FlexChart.AxisX>
    <Chart:Axis Title="果物" AxisLine="True" Position="Bottom"></Chart:Axis>
</Chart:C1FlexChart.AxisX>
<Chart:C1FlexChart.AxisY>
    <Chart:Axis AxisLine="True" Position="Right"></Chart:Axis>
</Chart:C1FlexChart.AxisY>
```



軸のタイトル

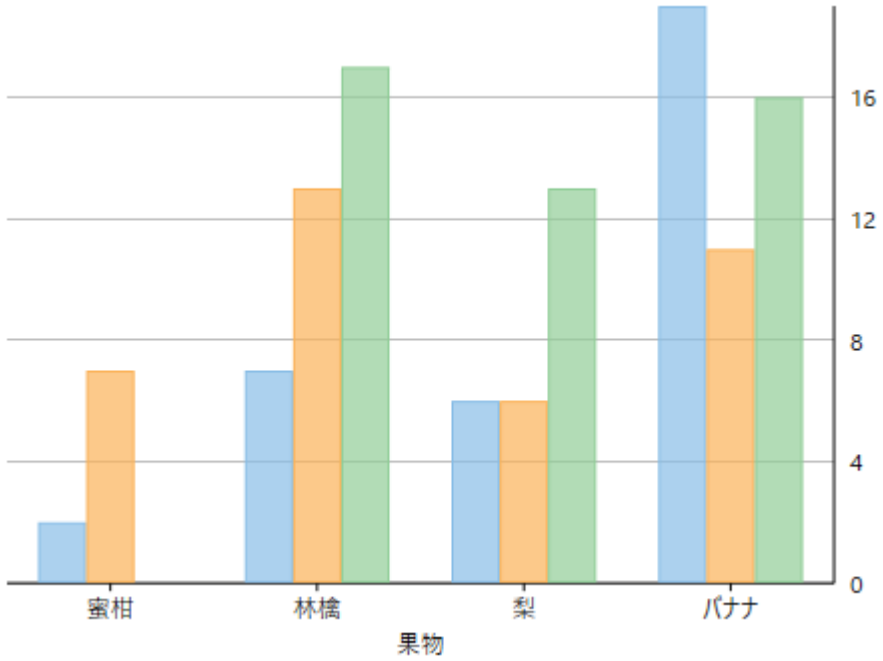
チャートを作成したら、チャート内の任意の垂直軸または水平軸にタイトルを追加できます。軸のタイトルは、その軸に何が表示されているかという情報を表示します。チャートを見ているエンドユーザーは、軸のタイトルから何のデータであるかを理解できます。ただし、円グラフなどの軸のないチャートに軸タイトルを追加することはできません。

FlexChart では、文字列を受け取る **Title** プロパティを使用して軸タイトルを設定できます。

次のコードスニペットを参照してください。

- **XAML**

```
<Chart:C1FlexChart.AxisX>
    <Chart:Axis Title="果物" AxisLine="True" Position="Bottom"></Chart:Axis>
</Chart:C1FlexChart.AxisX>
<Chart:C1FlexChart.AxisY>
    <Chart:Axis AxisLine="True" Position="Right"></Chart:Axis>
</Chart:C1FlexChart.AxisY>
```



軸の目盛りマーク

軸の目盛りマークは、ラベルが軸上にプロットされるポイントであり、軸上の項目の位置を示す小さなマークです。また、軸の特定のプロパティによって決定される値に基づいて、軸を等間隔に分割します。グリッド線の位置も、目盛りマークの位置によって制御されます。

軸の目盛りマークについては、基本的に、大目盛りマークと小目盛りマークの 2 種類がレンダリングされます。大目盛りマークは、軸が間隔グリッド線と交差する場所に自動的にレンダリングされます。小目盛りマークは、大目盛りマークの間にレンダリングされます。

デフォルトの **FlexChart** は、X 軸には大目盛りマークを設定し、Y 軸には目盛りマークなしを設定します。

ただし、**MajorTickMarks** プロパティと **MinorTickMarks** プロパティを使用すると、それぞれ大目盛りマークと小目盛りマークの位置を操作できます。

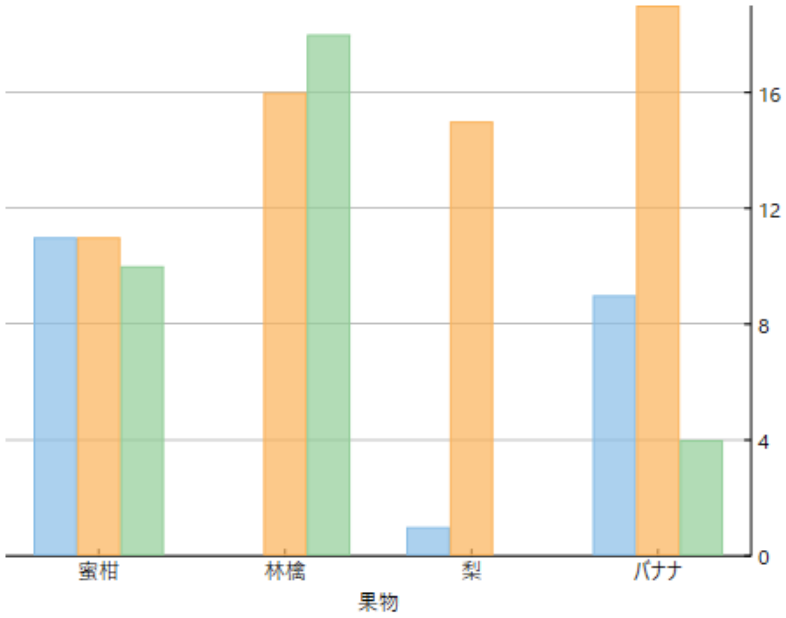
どちらのプロパティも、次に示す **TickMark** 列挙値のいずれかに設定できます。

値	説明
TickMark.Cross	軸に交差する目盛りマークを表示します。
TickMark.Inside	目盛りマークをプロットの内側に表示します。
TickMark.None	目盛りマークを表示しません。
TickMark.Outside	目盛りマークをプロットの外側に表示します。

次のコードサンプルを参照してください。

- **XAML**

```
<Chart:C1FlexChart.AxisX>
    <Chart:Axis MajorTickMarks="Inside" Title="果物" AxisLine="True" Position="Bottom"></Chart:Axis>
</Chart:C1FlexChart.AxisX>
<Chart:C1FlexChart.AxisY>
    <Chart:Axis MajorTickMarks="Inside" AxisLine="True" Position="Right"></Chart:Axis>
</Chart:C1FlexChart.AxisY>
```



軸のグリッド線

軸のグリッド線は、垂直軸または水平軸からチャートのプロット領域を横切って伸びる線です。ラベル単位およびデータ単位で表示され、軸に表示される大目盛りマークおよび小目盛りマークに揃えて配置されます。これらの補助線は格子状に表示され、特に正確な値が必要な場合に、チャートが読みやすくなります。

軸のグリッド線には、主に、主グリッド線と副グリッド線の 2 種類があります。大目盛りマークに垂直にラベル単位間隔で表示される線が主グリッド線、小目盛りマークに垂直にデータ単位間隔で表示される線が副グリッド線です。

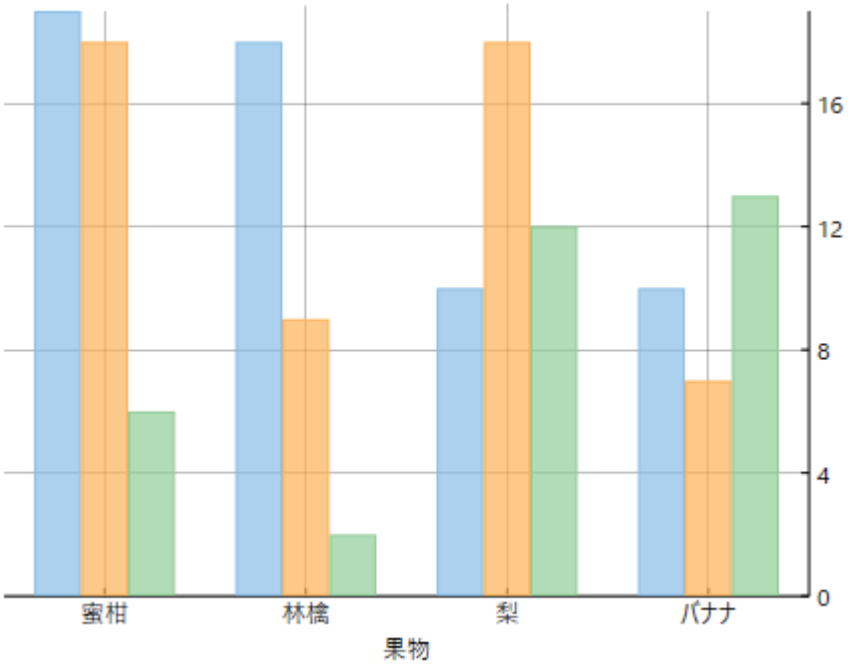
FlexChart の主グリッド線は **MajorGrid** プロパティ、副グリッド線は **MinorGrid** プロパティによって制御されます。また、主グリッド線と副グリッド線の外観は、それぞれ **MajorGridStyle** プロパティと **MinorGridStyle** プロパティによって制御されます。

これらのプロパティを使用して、水平および垂直方向のグリッド線を表示することができ、FlexChart のデータをさらに読みやすくできます。

次のコードは、これらのプロパティの設定方法を示します。

- **XAML**

```
<Chart:C1FlexChart.AxisX>
    <Chart:Axis MajorGrid="True" MajorTickMarks="Inside" Title="果物"
    AxisLine="True" Position="Bottom"></Chart:Axis>
</Chart:C1FlexChart.AxisX>
<Chart:C1FlexChart.AxisY>
    <Chart:Axis MajorGrid="True" MajorTickMarks="Inside" AxisLine="True"
    Position="Right"></Chart:Axis>
</Chart:C1FlexChart.AxisY>
```



軸の範囲

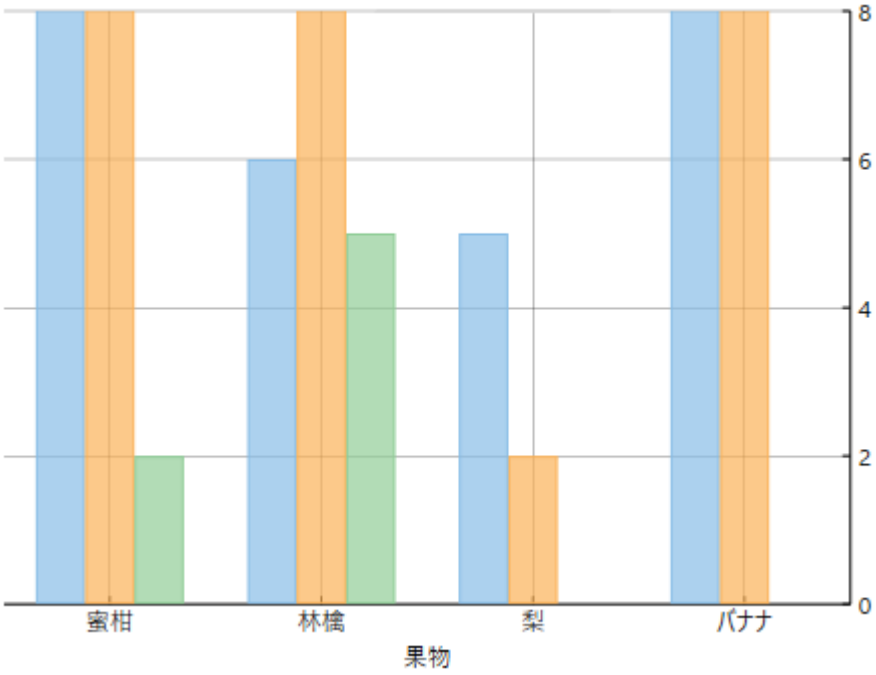
チャート内に特定のデータ部分を表示する場合は、軸の範囲を固定します。軸の範囲を固定すると、データの最小値と最大値を考慮して、各軸の範囲が決定されます。

FlexChart では、軸の **Min** プロパティと **Max** プロパティを設定して、軸の範囲を設定できます。

次のコードは、**Min** プロパティと **Max** プロパティの設定方法を示します。

- **XAML**

```
<Chart:C1FlexChart.AxisY>
    <Chart:Axis Min="0" Max="8" MajorGrid="True" MajorTickMarks="Inside"
    AxisLine="True" Position="Right"></Chart:Axis>
</Chart:C1FlexChart.AxisY>
```



軸の反転

データセットに含まれる X または Y の値の範囲が広いと、一般的なチャート設定では、情報を効率よく表示できないことがあります。軸を反転させることで、チャートデータをより効果的に提示できることもよくあります。

FlexChart の軸は、**Reversed**プロパティを使用して反転させることができます。

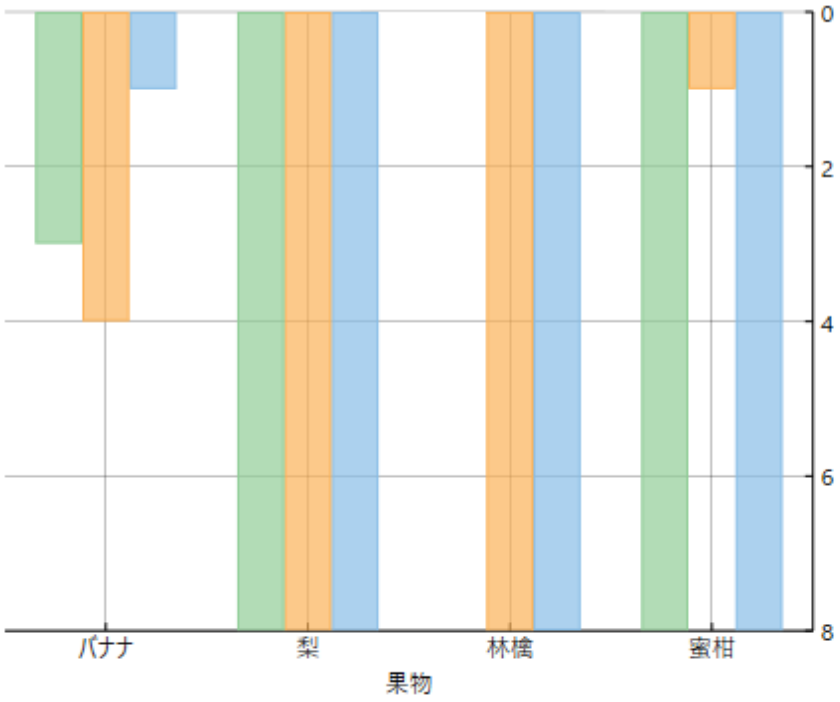
軸の **Reversed** プロパティを True に設定すると、軸が反転します。つまり、軸の最大値が最小値の位置に表示され、最小値が最大値の位置に表示されます。

初期状態では、最小値は X 軸の左側、Y 軸の下端に表示されます。しかし、軸の Reversed プロパティは最大値と最小値を入れ替えて表示します。

次にサンプルコードを示します。

- **XAML**

```
<Chart:C1FlexChart.AxisX>
    <Chart:Axis Reversed="True" MajorGrid="True"
    MajorTickMarks="Inside" Title="果物" AxisLine="True" Position="Bottom"></Chart:Axis>
</Chart:C1FlexChart.AxisX>
<Chart:C1FlexChart.AxisY>
    <Chart:Axis Reversed="True" Min="0" Max="8" MajorGrid="True"
     MajorTickMarks="Inside" AxisLine="True" Position="Right"></Chart:Axis>
</Chart:C1FlexChart.AxisY>
```

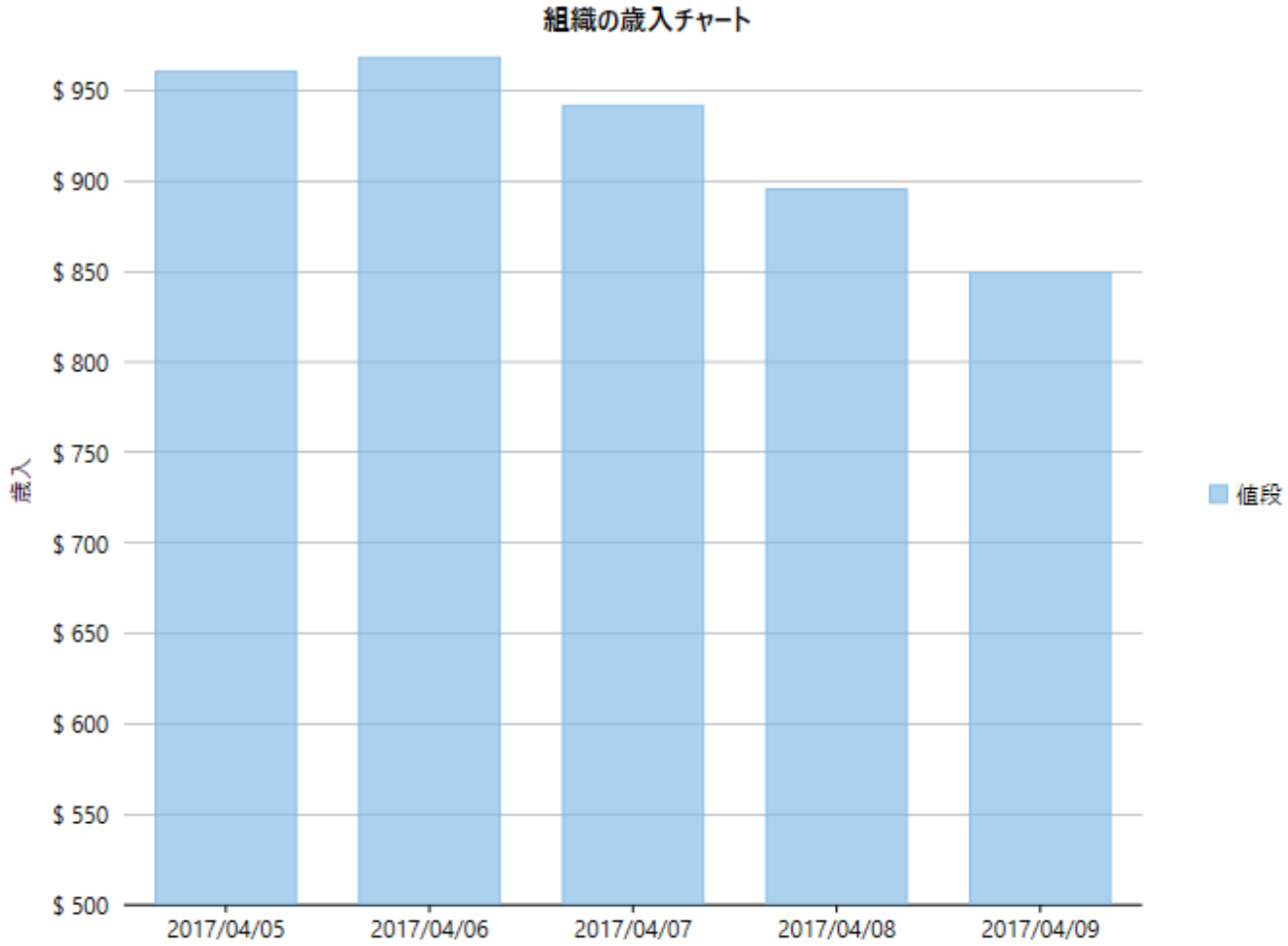


軸の連結

軸を連結すると、チャートの連結に基づいて軸に表示されるデフォルトの軸ラベルが上書きされます。つまり、軸の連結を使用すると、チャートデータソース以外のデータソースの軸ラベルを表示できます。

FlexChart では、**Axis** クラスの **ItemsSource** プロパティを使用して、軸をデータソースに連結します。Axis クラスの **Binding** プロパティを使用して、データソースの軸ラベルの値を含むフィールドを指定します。

次の図は、チャートデータソースに含まれないフィールドから Y 軸に表示されたラベルです。



次のコードでは、特定の年の組織の収益データを使用します。チャートデータソースには、ユーロ通貨の収益データが含まれます。ユーロ通貨のラベルを米ドル通貨のラベルに置換するために、コードで Y 軸を米ドル軸ラベルのデータを含むデータソースに連結します。

- **Visual Basic**

```
' Y軸をデータソースにバインドします。
flexChart.AxisY.ItemsSource = AxisData

' 軸ラベルの値を含むフィールドを指定します。
flexChart.Binding = "Value,Text"
```

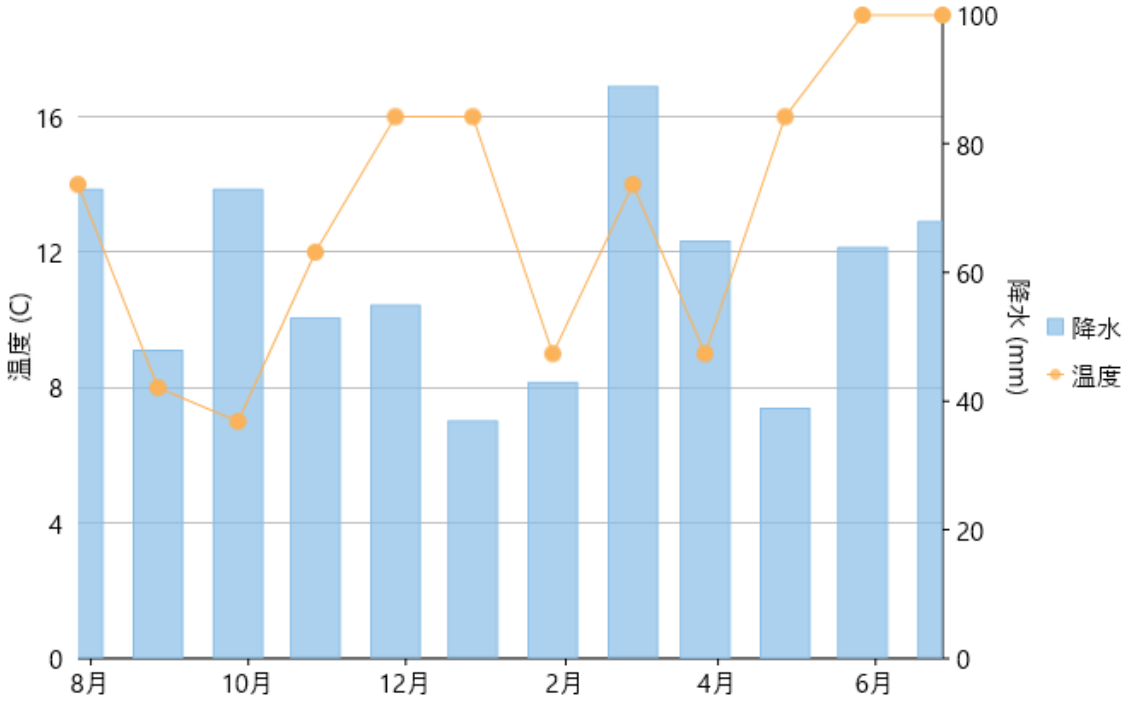
- **C#**

```
// Y軸をデータソースにバインドします。
flexChart.AxisY.ItemsSource = AxisData;

// 軸ラベルの値を含むフィールドを指定します。
flexChart.Binding = "Value,Text";
```

複数の軸

チャートに第 1 X 軸と第 1 Y 軸が含まれているが、軸を追加しないと要件を満たせないことがあります。たとえば、同じチャートに値の範囲がまったく異なる系列をプロットする場合です。また、1 つのチャートに（異なるタイプの）まったく異なる値をプロットする場合もあります。このような場合、2 つの軸だけではデータを効率よく表示できません。そのような場合は、第 2 軸を使用すると便利です。第 2 軸を使用するには、それぞれ独自の X 軸と Y 軸を使用して 1 つのチャートに複数の系列をプロットできます。

FlexChart では、複数の軸を簡単に使用できます。要件に応じて追加の軸を作成し、系列の AxisX プロパティと AxisY プロパティに同様に連結するだけです。



次のコードスニペットは、FlexChart で複数の軸を作成して使用する方法を示します。

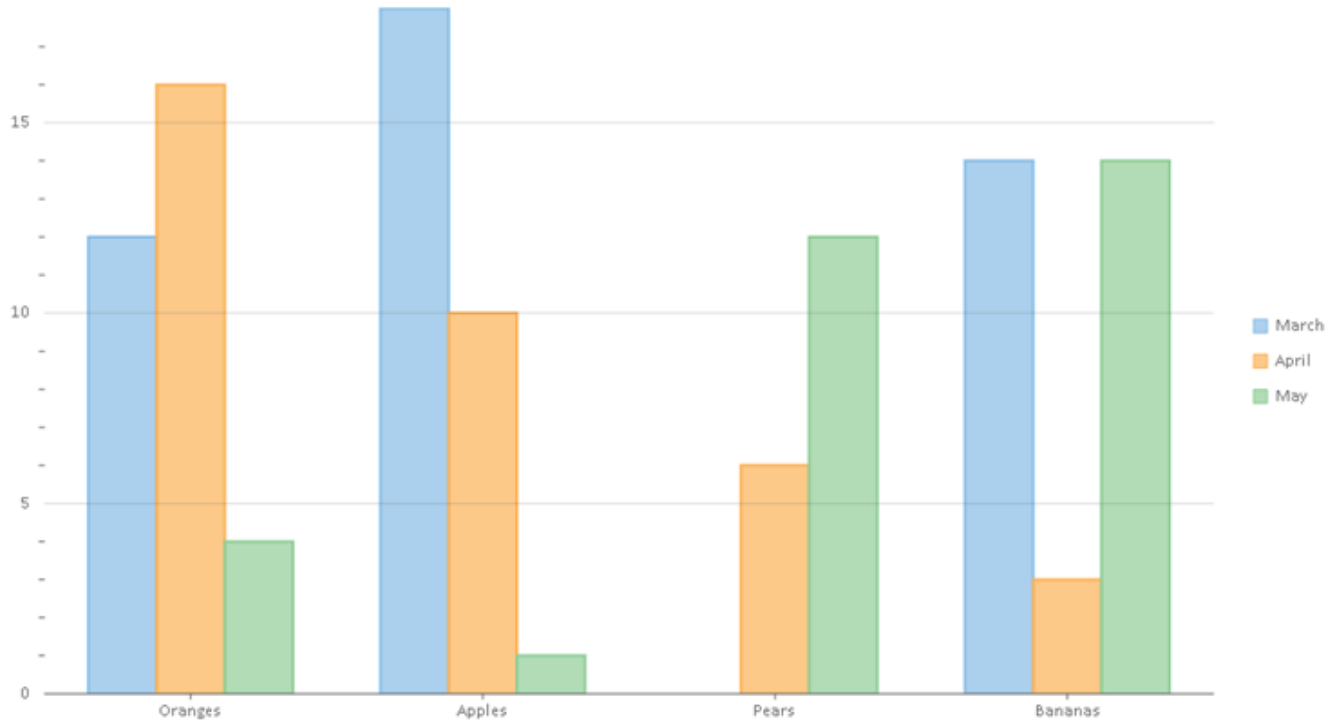
- **XAML**

```
<Chart:C1FlexChart x:Name="flexChart" ItemsSource="{Binding DataContext.Data}" BindingX="Time"
                   Grid.Row="1">
    <Chart:Series x:Name="precSeries" SeriesName="降水" Binding="Precipitation">
        <Chart:Series.AxisY>
            <Chart:Axis Position="Right" Min="0" Max="100" Title="降水 (mm)" MajorGrid="False"/>
        </Chart:Series.AxisY>
    </Chart:Series>
    <Chart:Series x:Name="avgSeries" SeriesName="温度" ChartType="LineSymbols" Binding="Temperature"/>
    <Chart:C1FlexChart.AxisY>
        <Chart:Axis Title="温度 (C)" Min="0" MajorGrid="True" AxisLine="False" Position="Left"
        MajorTickMarks="None"></Chart:Axis>
    </Chart:C1FlexChart.AxisY>
</Chart:C1FlexChart>
```

軸の単位

Axis units define at what intervals the tickmarks/gridlines should be displayed along the axis. FlexChart, by default, calculates the major and minor units automatically according to the data to be plotted on the chart. However, you can choose to change these intervals by using MajorUnit and MinorUnit property. This change impacts the values displayed on the value axis as well as the positioning of tick marks and grid lines if you choose to show them.

For example, in the FlexChart shown below, we have set the **MajorUnit** property to 5 and **MinorUnit** property to 1 on the Y-axis. You will notice that the difference between each major tick mark is 5 and minor tick mark is 1.



Use the following code to set the MajorUnit and MinorUnit properties.

In XAML

```
<Chart:C1FlexChart.AxisY>
    <Chart:Axis Format="0.00" MajorUnit="5" MinorUnit="1"></Chart:Axis>
</Chart:C1FlexChart.AxisY>
```

In Code

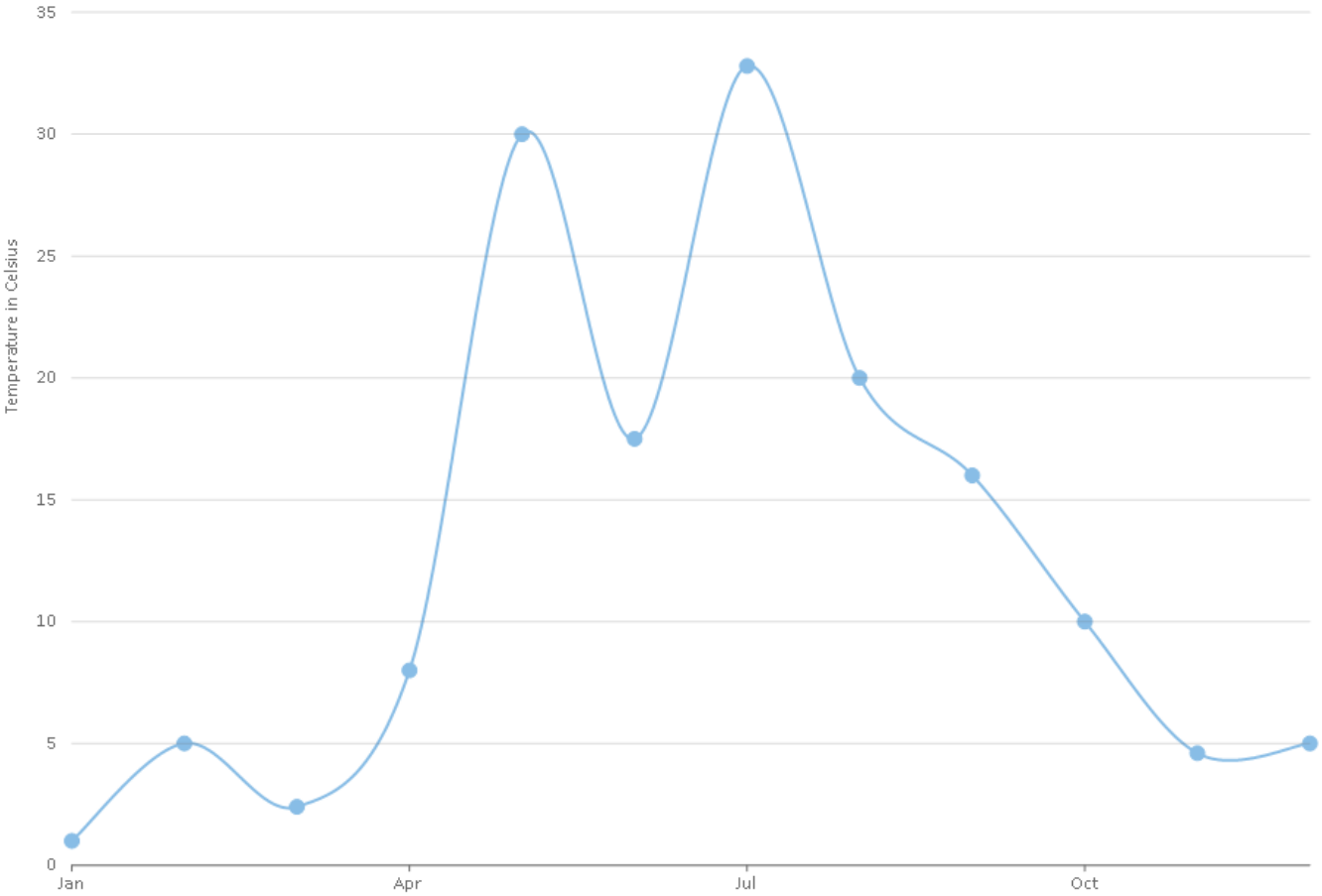
```
flexChart.AxisY.MajorUnit = 5;
flexChart.AxisY.MinorUnit = 1;
```

DateTime Axis Units

In case of DateTime axis, FlexChart provides you an option to set the time unit as well using the TimeUnit property. You can choose from day, month, quarter, week, and year options. This property accepts value from the TimeUnits enumeration. The enumeration includes the following values.

- Day
- Week
- Month
- Quarter
- Year

For example, in order to set major unit of a value axis to 3 months, you first need to set the **TimeUnit** property to Month, and then set the **MajorUnit** property to 3.



Use the following code to set the **TimeUnit** and **MajorUnit** properties.

In XAML

```
<Chart:C1FlexChart.AxisX>
   <Chart:Axis Format="0.00" TimeUnit="Month" MajorUnit="3"></Chart:Axis>
</Chart:C1FlexChart.AxisX>
```

In Code

```
flexChart1.AxisX.TimeUnit = C1.Chart.TimeUnits.Month;
flexChart1.AxisX.MajorUnit = 3;
```

FlexChart の軸ラベル

軸ラベルは、軸に沿って表示される値です。デフォルトでは、軸ラベルは、軸のデータポイントおよび生成された間隔に基づいて決定されます。

FlexChart では、以下に示すプロパティを使用して、軸ラベルの外観、書式、および配置を変更できます。

プロパティ	説明
Format	軸ラベルで使用する書式文字列を指定します。
LabelAlignment	軸ラベルの配置を設定します。
LabelAngle	ラベルの回転角度を指定します。
Labels	軸ラベルを表示するかどうかを示します。

OverlappingLabels	重なっているラベルを処理する方法を示します。

以下のセクションでは、これらのプロパティの使用方法を説明します。

- 軸ラベルの書式
- 軸ラベルの回転
- 軸ラベルの表示/非表示
- 軸ラベルの重なり
- Axis Grouping

軸ラベルの書式

デフォルトでは、軸ラベルは、軸のデータポイントと生成された間隔に基づいて自動的に決定されます。しかし、**Format** プロパティを使用して、ニーズに合わせて軸ラベルを書式設定することもできます。

Format プロパティは、標準の .Net 書式文字列の値を受け取ります。

軸ラベルの回転

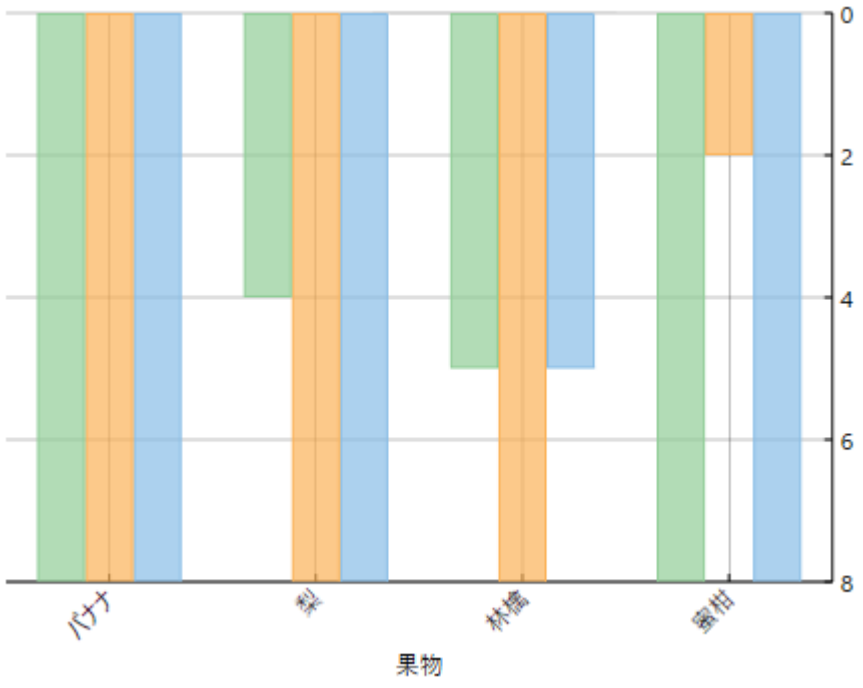
水平軸に軸ラベルが密集している場合は、ラベルを回転して、外観が雑然とならないようにする必要があります。ラベルを回転すると、多数のラベルを軸上の限られたスペースに収めることができます。

FlexChart では、**LabelAngle** プロパティを使用して、軸ラベルを反時計回りに回転させることができます。

次のコードを参照してください。

- **XAML**

```
<Chart:C1FlexChart.AxisX>
    <Chart:Axis LabelAngle="45" Reversed="True" MajorGrid="True"
    MajorTickMarks="Inside" Title="果物"
    AxisLine="True" Position="Bottom"></Chart:Axis>
</Chart:C1FlexChart.AxisX>
```



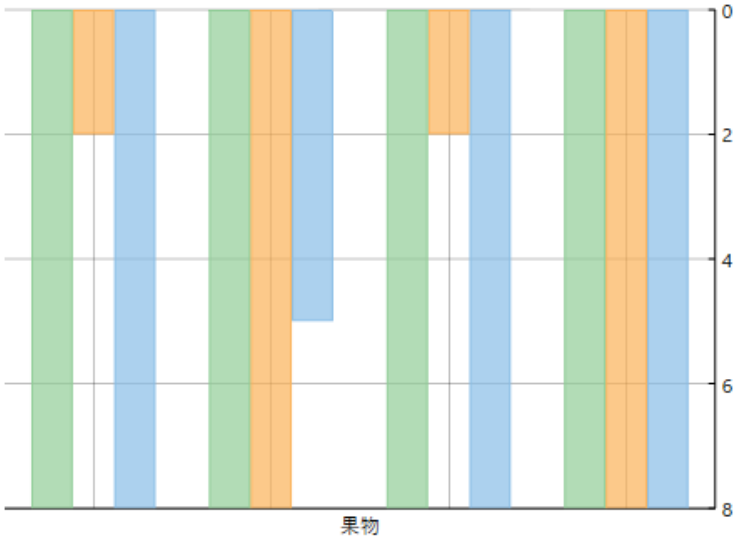
軸ラベルの表示/非表示

FlexChart では、**Labels** プロパティを使用して、軸ラベルを表示または非表示にすることができます。特定の軸の軸ラベルを非表示にする場合は、その軸で、このプロパティを False に設定します。**Labels** プロパティのデフォルト値は True です。

次のコードスニペットを参照してください。

- **XAML**

```
<Chart:C1FlexChart.AxisX>
    <Chart:Axis Labels="False" LabelAngle="45" Reversed="True" MajorGrid="True" MajorTickMarks="Inside"
    Title="果物" AxisLine="True" Position="Bottom"></Chart:Axis>
</Chart:C1FlexChart.AxisX>
```



軸ラベルの重なり

In case there are less number of data points and shorter label text, axis labels are rendered without any overlapping. However, axis labels may overlap due to its long text or large numbers of data points in chart.

To manage overlapped axis labels in FlexChart, use the following options.

- **Trim or Wrap Axis Labels**
- **Staggered Axis Labels**

Trim or Wrap Axis Labels

In case there are overlapping labels in the chart for any reason, you can manage the same using the **OverlappingLabels** property.

The OverlappingLabels property accepts the following values in the **OverlappingLabels** enumeration:

Property	Description
Auto	Hides overlapping labels.
Show	Shows all labels including the overlapping ones.
Trim	Trim label, if it's larger than the available width.
WordWrap	Wrap label, if it's larger than the available width.

Here is the code snippet:

- **XAML**

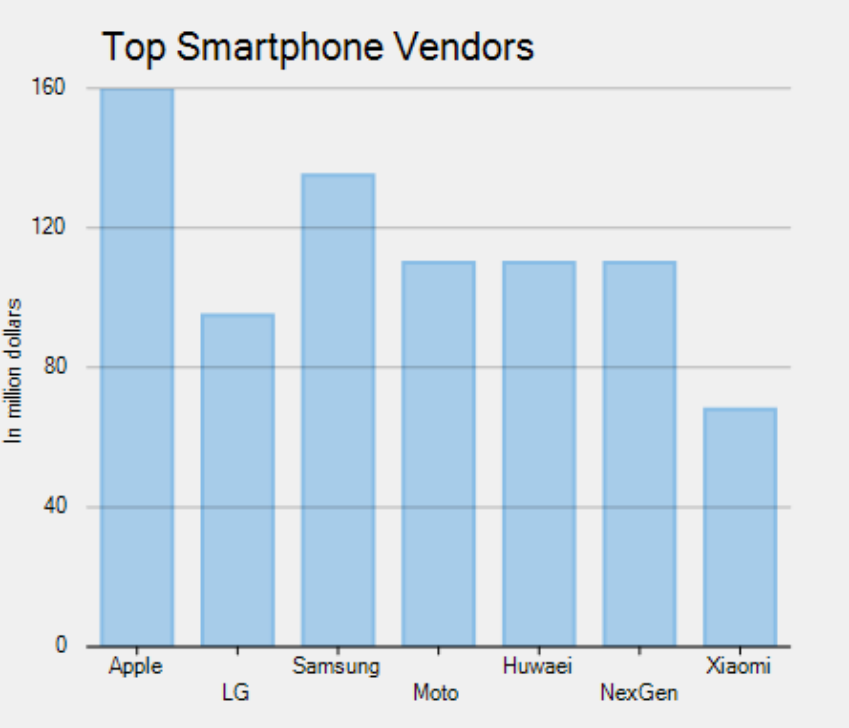
```
<Chart:C1FlexChart.AxisX>
    <Chart:Axis OverlappingLabels="Auto" Labels="False" LabelAngle="45"
    Reversed="True" MajorGrid="True" MajorTickMarks="Inside" Title="Fruits"
    AxisLine="True" Position="Bottom"></Chart:Axis>
</Chart:C1FlexChart.AxisX>
<Chart:C1FlexChart.AxisY>
    <Chart:Axis OverlappingLabels="Show" Reversed="True" Min="0" Max="8"
    MajorGrid="True" MajorTickMarks="Inside" AxisLine="True" Position="Right"></Chart:Axis>
</Chart:C1FlexChart.AxisY>
```

Staggered Axis Labels

Another way to handle overlapping of axis labels is to stagger them for better visibility. Staggered axis labels can generated by using **StaggeredLines** property. This property accepts an integer value and the default value is set to 1.

C#

```
// StaggeredLines プロパティを設定します
flexChart1.AxisX.StaggeredLines = 2;
```



軸のグループ化

FlexChart provides the flexibility to group axis labels as per the requirement. Axis grouping helps in improving the readability of the chart and makes it easy for analyzing data from different levels. Implementation of axis grouping in FlexChart depends on the data you are using, it can be either categorical data, numerical data, or DateTime data.

FlexChart supports the following axis grouping depending upon the data.

Categorical Axis Grouping
: Learn how to perform axis grouping while working with categorical data.

Numerical Axis Grouping

Learn how to perform axis grouping while working with numerical data.

DateTime Axis Grouping

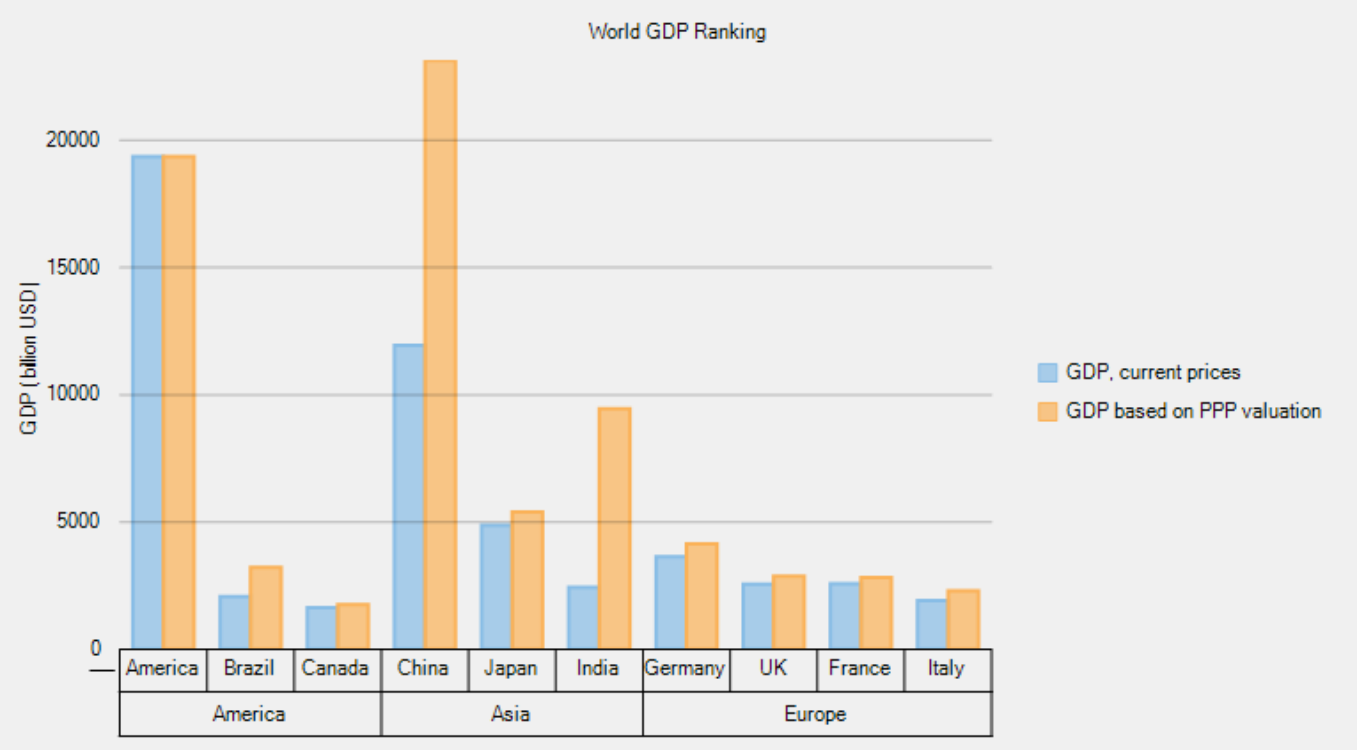
Learn how to perform axis grouping while working with date time format data.

カテゴリ別の軸グループ化

Categorical axis grouping is applicable in scenarios where the data displayed on the axis is categorical in nature. Categorical data can either be flat or hierarchical. In case you are using flat data, use the **GroupNames** property to apply axis grouping. And, in case you are using hierarchical data, use the GroupNames and **GroupItemsPath** property to apply axis grouping.

Moreover, FlexChart allows you to set the group separator using the **GroupSeparator** property. Also, it allows you to expand or collapse group levels by setting the **GroupVisibilityLevel** property.

The following image shows how FlexChart appears after setting the categorical axis grouping using flat data.



Add the following code in Index.xaml.

XAML

```
<Chart:C1FlexChart x:Name="flexChart" ChartType="Column" BindingX="Country"
     ItemsSource="{Binding Data}" ToolTipContent="{}{x}{seriesName}
{y:n0}" Header="World GDP Ranking" Grid.Row="1" >
            <Chart:C1FlexChart.HeaderStyle>
                <Chart:ChartStyle FontSize="20" FontFamily="GenericSansSerif"/>
            </Chart:C1FlexChart.HeaderStyle>
            <Chart:Series SeriesName="GDP, current prices"
Binding="CurrentPrices"/>
            <Chart:Series SeriesName="GDP based on PPP valuation"
Binding="PPPValuation"/>
            <Chart:C1FlexChart.AxisX>
```

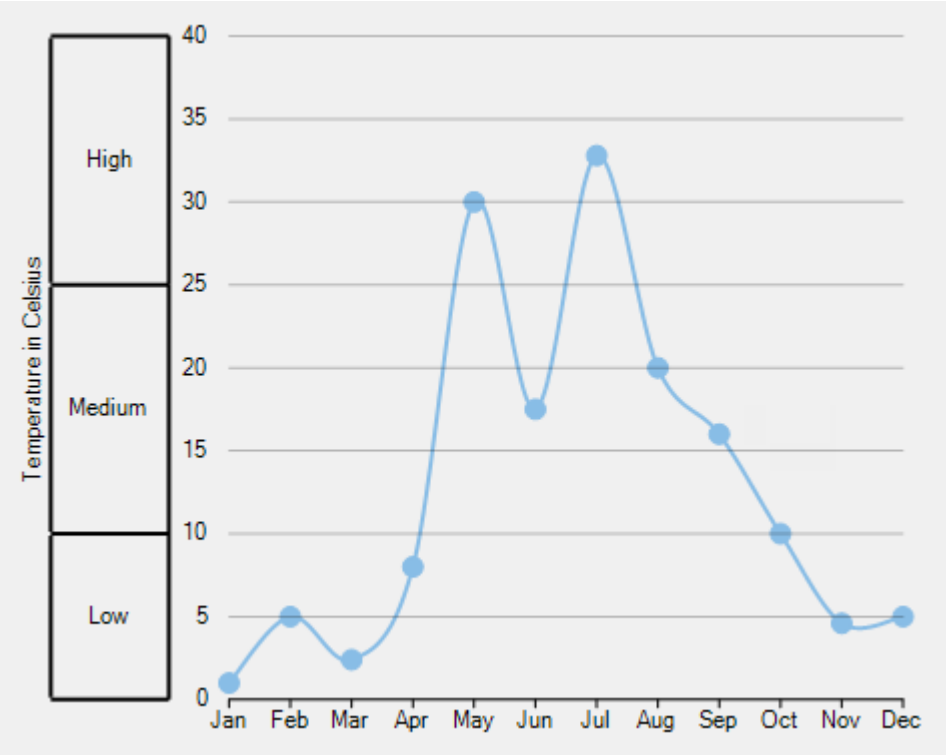
```
                <Chart:Axis GroupSeparator="Vertical" GroupVisibilityLevel="-2"
GroupNames="Continent" />
            </Chart:C1FlexChart.AxisX>
            <Chart:C1FlexChart.AxisY>
                <Chart:Axis Title="GDP (billion USD)" MajorGrid="True"/>
            </Chart:C1FlexChart.AxisY>
        </Chart:C1FlexChart>
```

数値の軸グループ化

Numerical axis grouping is applicable in scenarios where the data displayed on the axis represents numeric values. To implement numerical axis grouping in FlexChart, set the GroupProvider property to an object of the **IAxisGroupProvider** implementation.

In the example code below, we have created a class **NumericAxisGroupProvider** that implements the IAxisGroupProvider interface. The interface provides **GetLevels** method that returns the group levels and **GetRanges** method that returns the group ranges for a given level. Moreover, FlexChart allows you to set the group separator using the **GroupSeparator** property.

The following image shows how FlexChart appears after setting the numerical axis grouping.



Add the following code in Index.xaml.

XAML

```
<Chart:C1FlexChart x:Name="flexChart" ChartType="SplineSymbols" BindingX="Month"
            ItemsSource="{Binding Data}" Grid.Row="1" >
            <Chart:Series Binding="Temperature"/>
            <Chart:C1FlexChart.AxisY>
                <Chart:Axis Title="Temperature in Celsius" MajorGrid="True"
GroupSeparator="Horizontal" Min="0" Max="40" Position="Right"/>
```

```
            </Chart:C1FlexChart.AxisY>
</Chart:C1FlexChart>
```

Code

```
public NumericAxisGrouping()
        {
            InitializeComponent();
            flexChart.AxisY.GroupProvider = new NumericAxisGroupProvider();
        }
        class NumericAxisGroupProvider : IAxisGroupProvider
        {
            public int GetLevels(IRange range)
            {
                return 1;
            }

            public IList<IRange> GetRanges(IRange range, int level)
            {
                var ranges = new List<IRange>();
                if (level == 1)
                {
                    ranges.Add(new DoubleRange("Low", 0, 10));
                    ranges.Add(new DoubleRange("Medium", 10, 25));
                    ranges.Add(new DoubleRange("High", 25, 40));
                }
                return ranges;
            }
        }
```

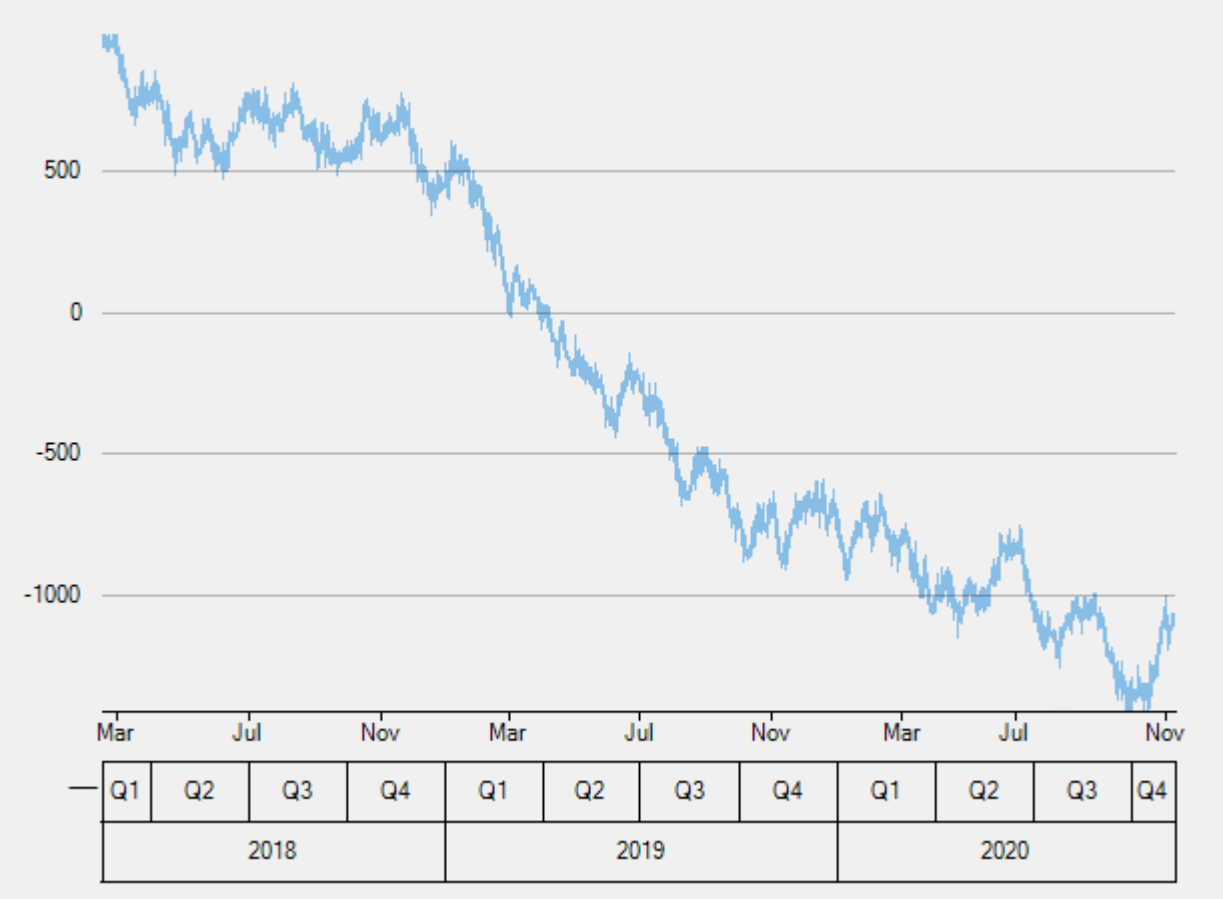
DateTimeの軸グループ化

DateTime axis grouping is applicable in scenarios where the data displayed on the axis represents date time values. To implement date axis grouping in FlexChart, set the **GroupProvider** property to an object of the **IAxisGroupProvider** implementation.

In the example code below, we have created a class **DateTimeGroupProvider** that implements the IAxisGroupProvider interface. The interface provides **GetLevels** method that returns the group levels and **GetRanges** method that returns the group ranges for a given level. Moreover, FlexChart allows you to set the group separator using the **GroupSeparator** property.

Moreover, FlexChart allows you to set the group separator using the **GroupSeparator** property. Also, it allows you to expand or collapse group levels by setting the **GroupVisibilityLevel** property.

The following image shows how FlexChart appears after setting the date axis grouping.



Add the following code in Index.xaml.

XAML

```
<Chart:C1FlexChart x:Name="flexChart" ChartType="Line" BindingX="Time"
            ItemsSource="{Binding Data}" Grid.Row="1" >
            <Chart:Series Binding="Price"/>
            <Chart:C1FlexChart.AxisX>
                <Chart:Axis GroupSeparator="Horizontal" GroupVisibilityLevel = "1"
Format="MMM" />
            </Chart:C1FlexChart.AxisX>
        </Chart:C1FlexChart>
```

Code

```
public DateTimeAxisGrouping()
        {
            InitializeComponent();
            flexChart.AxisX.GroupProvider = new DateTimeGroupProvider();
        }
        public class DateTimeGroupProvider : IAxisGroupProvider
        {
            public int GetLevels(IRange range)
            {
                return 2;
```

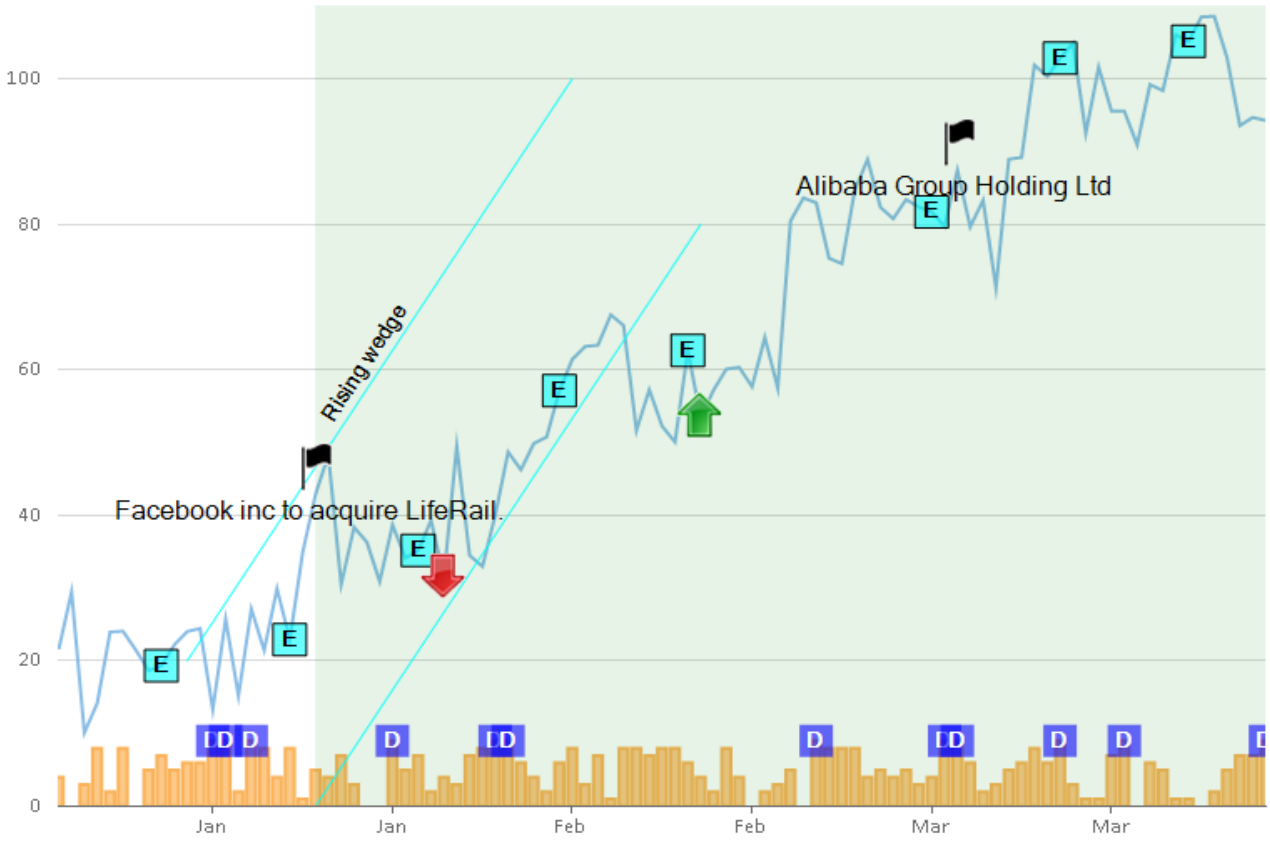
```
            }

            public IList<IRange> GetRanges(IRange range, int level)
            {
                var timeRange = range as TimeRange;
                if (timeRange == null)
                    return null;
                var min = timeRange.TimeMin;
                var max = timeRange.TimeMax;
                var span = max - min;

                List<IRange> ranges = new List<IRange>();
                DateTime start;
                if (level == 1)
                {
                    start = new DateTime(min.Year,
((int)Math.Ceiling((double)min.Month / 3) - 1) * 3 + 1, 1);
                    ranges = Enumerable.Range(0, ((max.Month - start.Month) / 3 +
1) + 4 * (max.Year - start.Year)).Select(a => start.AddMonths(a * 3))
                   .TakeWhile(a => a <= max)
                   .Select(a => (IRange)(new TimeRange("Q" +
(int)Math.Ceiling((double)a.Month / 3), a, a.AddMonths(3)))).ToList();
                }
                else
                {
                    start = new DateTime(min.Year, 1, 1);
                    ranges = Enumerable.Range(0, max.Year - start.Year +
1).Select(a => start.AddYears(a))
                   .TakeWhile(a => a <= max)
                   .Select(a => (IRange)(new TimeRange(a.ToString("yyyy"), a,
a.AddYears(1)))).ToList();
                }

                return ranges;
            }
        }
```

注釈

注釈は、チャート内の特定の領域をマークしたり強調表示するために使用するビジュアル要素です。たとえば、テキスト、画像、図形を使用して、特定のデータポイントに関する重要な情報を表示したり強調表示することができます。チャートを使用する主な目的は、チャートデータをわかりやすく伝えることです。

FlexChart には、図形、テキスト、画像の 3 つの注釈カテゴリと、8 つのタイプの注釈があります。注釈タイプごとに、さまざまな方法でチャートデータを有益な情報にすることができます。円、四角形、多角形などの図形で情報を表示したり、わかりやすいメモや画像でデータを強調表示することができます。

さらに、絶対、相対、データインデックス、データ座標などの添付モードを使用して、FlexChart に注釈を配置できます。フォント、色、ストロークのスタイル設定プロパティから、注釈とそのコンテンツの両方をカスタマイズできます。ツールチップ、特に画像注釈を追加して、注釈を対話式にすることができます。



注釈については、次のセクションを参照してください。

- 注釈の追加
- 注釈の配置
- 注釈のカスタマイズ
- 注釈のタイプ
- 吹き出しの作成

注釈の追加

FlexChart では、注釈レイヤに注釈を追加できます。注釈レイヤには、チャート内のすべての注釈のコレクションが含まれます。

FlexChart に注釈を追加するには、次の手順に従います。

1. FlexChart で注釈レイヤを作成します。
2. 注釈レイヤに注釈インスタンスを追加します。

FlexChart で注釈レイヤを作成するには、**AnnotationLayer**クラスのインスタンスを作成し、それをFlexChartの **Layers**コレクションに追加します。注釈レイヤに注釈を追加するには、注釈のタイプに対応する注釈クラスのインスタンスを作成します。その注釈インスタンスを注釈レイヤの **Annotations** コレクションに追加します。

次のコードスニペットは、FlexChart で四角形注釈を作成し、それを注釈レイヤに追加する方法を示します。

XAML

```
<Chart:C1FlexChart.Layers>
    <Annotation:AnnotationLayer>
        <Annotation:AnnotationLayer.Annotations>
            <Annotation:Rectangle Content="最大税収&#13;2013&#13;45000">
            </Annotation:Rectangle>
```

```
        </Annotation:AnnotationLayer.Annotations>
    </Annotation:AnnotationLayer>
</Chart:C1FlexChart.Layers>
```

注釈の配置

FlexChart の注釈の配置には、次の 2 つのメカニズムがあります（順序は不同）。

- チャートに相対的に注釈を配置する。
- データポイントに相対的に注釈を配置する。

チャートに相対的に注釈を配置する

チャートに相対的に注釈を配置する場合は、チャート内の注釈の添付と場所を指定します。

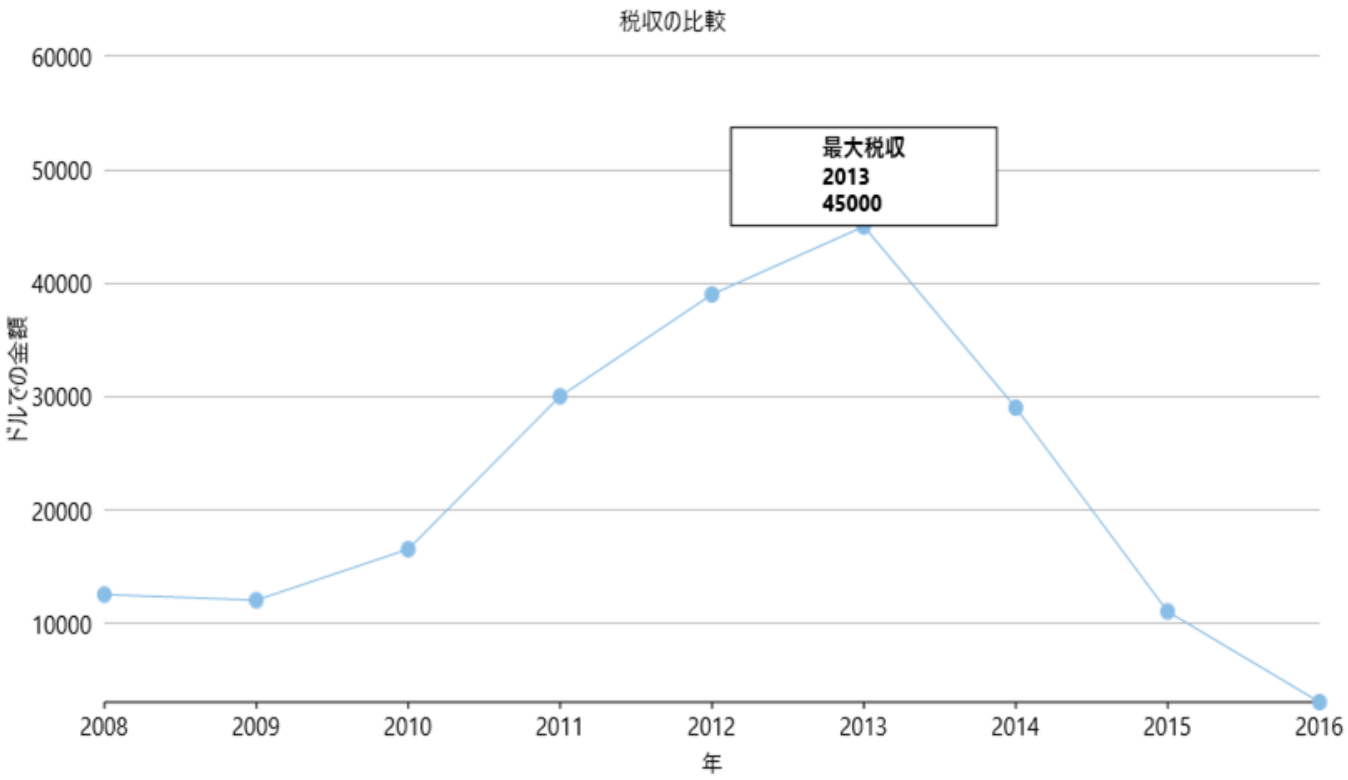
FlexChart で注釈を添付する方法は、次の 4 つがあります。

- **Absolute**：この添付は、アプリケーションのサイズ変更に関係なく、注釈が固定されて移動しないことを示します。絶対添付を設定するには、**Attachment** プロパティを **AnnotationAttachment** 列挙の **Absolute** に設定します。絶対添付モードで注釈の場所を設定するには、注釈の座標をピクセル単位で設定します。
- **DataCoordinate**：この添付は、注釈が特定のデータポイントに添付されることを示します。この添付を設定するには、Attachment プロパティを AnnotationAttachment 列挙の **DataCoordinate** に設定します。注釈の位置を設定するには、**Location** プロパティを設定して、注釈のデータ座標を指定します。
- **DataIndex**：この添付は、系列のインデックスに基づいて系列に、ポイントインデックスに基づいてポイントに、注釈が添付されることを示します。この添付を設定するには、Attachment プロパティを AnnotationAttachment 列挙の **DataIndex** に設定します。注釈の場所を指定するには、**SeriesIndex** および **PointIndex** プロパティを設定します。
- **Relative**：この添付は、注釈がチャートに相対的な場所とサイズを持つことを示します。この添付を設定するには、Attachment プロパティを AnnotationAttachment 列挙の **Relative** に設定します。Location プロパティを使用して、チャート内の相対的な注釈の場所を指定します。（0, 0）は左上、（1, 1）は右下です。

データポイントに相対的に注釈を配置する

AnnotationPosition 列挙の **Position** プロパティを設定して、データポイントに対する注釈の場所を指定します。

次の図に、2013 年の最大の税収入を強調表示する四角形注釈を示します。



次のコードは、9 年間の税収入データを比較し、最大の収入を表示します。このコードは、FlexChart の注釈レイヤに四角形注釈の添付、場所、位置を指定する方法を示します。

XAML

```
<Annotation:Rectangle Content="最大税収&#13;2013&#13;45000"
                      Attachment="DataIndex"
                      SeriesIndex="0"
                      PointIndex="5"
                      Position="Top"
                      Width="140"
                      Height="50">
</Annotation:Rectangle>
```

注釈のカスタマイズ

FlexChart の注釈は、サイズ（図形）、スケーリング（画像）、コンテンツスタイル（画像以外）をカスタマイズできます。

- **サイズ**：対応するクラスのサイズプロパティを使用して、すべての図形のサイズを変更します。たとえば、四角形注釈のサイズを変更するには、**Rectangle** クラスの **Height** および **Width** プロパティを設定します。
- **スタイル**：**AnnotationBase** クラスの **Style** プロパティを使用して、図形およびテキスト注釈の外観の色、フォント、ストロークをカスタマイズします。
- **コンテンツスタイル**：**Shape**クラスの **ContentStyle** プロパティを使用して、図形注釈内のコンテンツの外観をカスタマイズします。

次の図に、2013 年の最大の税収入をさらに強調表示するようにカスタマイズされた四角形注釈を示します。

次のコードは、9 年間の税収入データを比較し、最大の収入を表示します。このコードは、四角形注釈のサイズを設定し、外観とコンテンツをカスタマイズする方法を示します。

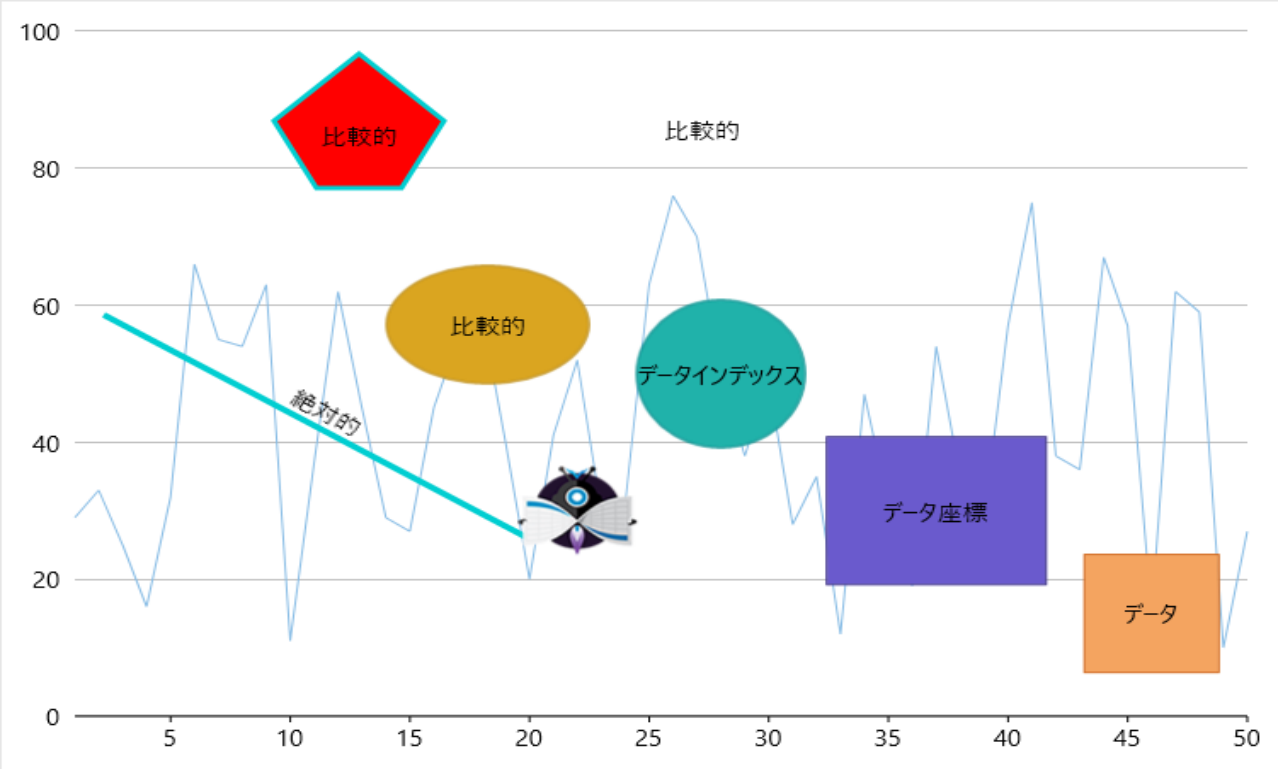
XAML

```
<Annotation:Rectangle.Style>
    <Chart:ChartStyle Fill="DarkBlue"
                      Stroke="OrangeRed"
                      StrokeThickness="2"
                      StrokeDashArray="1,2"
                      FontFamily="GenericSansSerif"
                      FontWeight="Normal" />
</Annotation:Rectangle.Style>
<Annotation:Rectangle.ContentStyle>
    <Chart:ChartStyle Stroke="Yellow"
                      FontFamily="GenericSansSerif"
                      FontSize="8.5"
                      FontWeight="Bold">
    </Chart:ChartStyle>
</Annotation:Rectangle.ContentStyle>
```

注釈のタイプ

FlexChart には、次の 3 つのカテゴリと 8 つのタイプの注釈があります。

- **図形**:円、楕円、四角形、正方形、線、多角形などの図形を使用して、チャートデータ内の特定の領域に有益な情報を表示したり、それらの領域を強調表示します。
- **テキスト**:テキスト注釈を使用して、説明用のメモや情報用のコメントをチャート内の特定のポイントに追加します。
- **画像**:画像注釈を使用して、一目でわかる画像によってチャートデータの意味が伝わりやすくします。



FlexChart が提供するさまざまな注釈タイプについては、次のセクションを参照してください。

- 図形注釈
- テキスト注釈
- 画像注釈

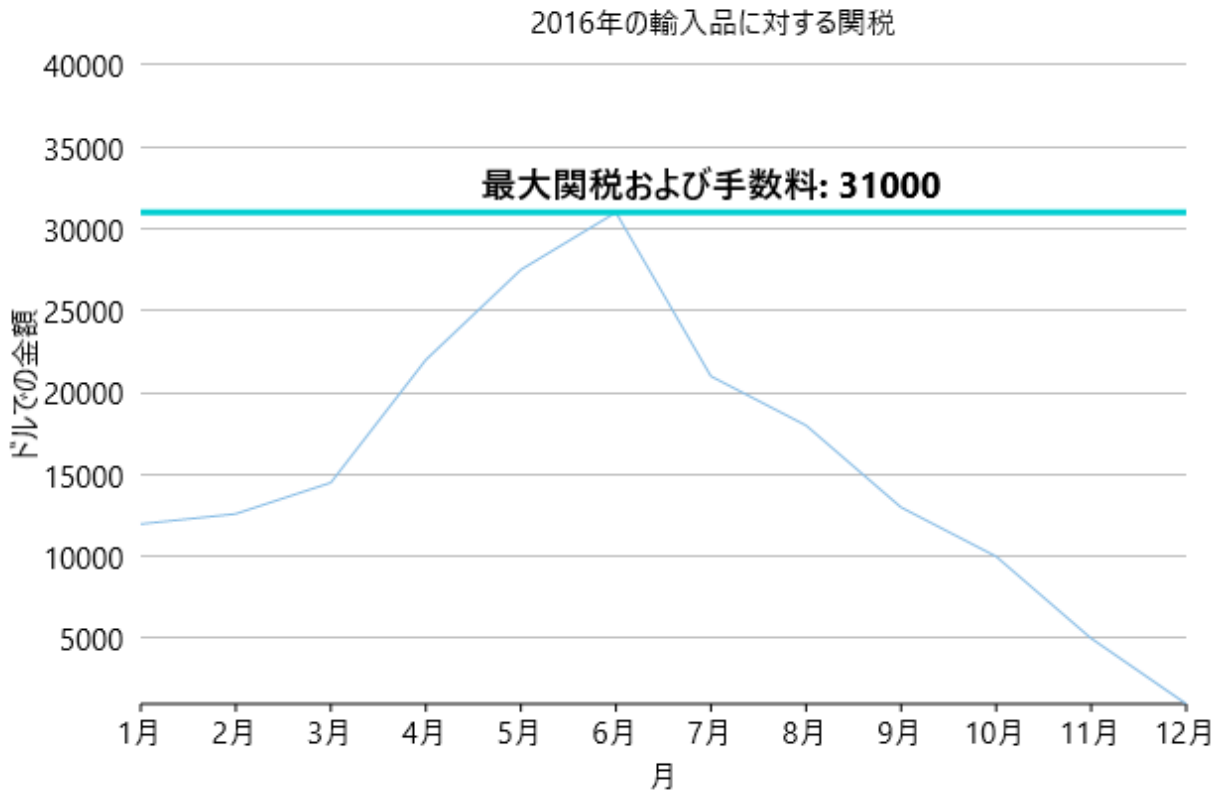
図形注釈

図形には、重要なデータを含む領域を強調表示してユーザーの注意を引きつける効果があります。

FlexChart は、次の 6 つの図形注釈を提供します。

- 円
- 楕円
- 線
- 多角形
- 四角形
- 正方形

次の図に、2016 年の最大の関税額および通関手数料を強調表示した線注釈を示します。



これらの図形を作成するには、図形注釈クラスのインスタンスを作成します。対応するクラスのサイズプロパティを使用して、図形のサイズを設定します。たとえば、線注釈を作成するには、**Line** クラスのインスタンスを作成します。Line クラスの **Start** および **End** プロパティを設定して、線注釈の長さを指定したり、線注釈を回転させます。

すべての図形注釈の基本クラスである **Shape** クラスの **Content** プロパティを設定して、テキストを指定します。さらに、FlexChart の多角形注釈を使用すると、三角形や矢印などの図形も作成できます。

次のコードは、商品の輸入関税データを使用して、2016 年のデータの増減を示します。このコードは、FlexChart の線注釈を追加、配置、カスタマイズする方法を示します。

XAML

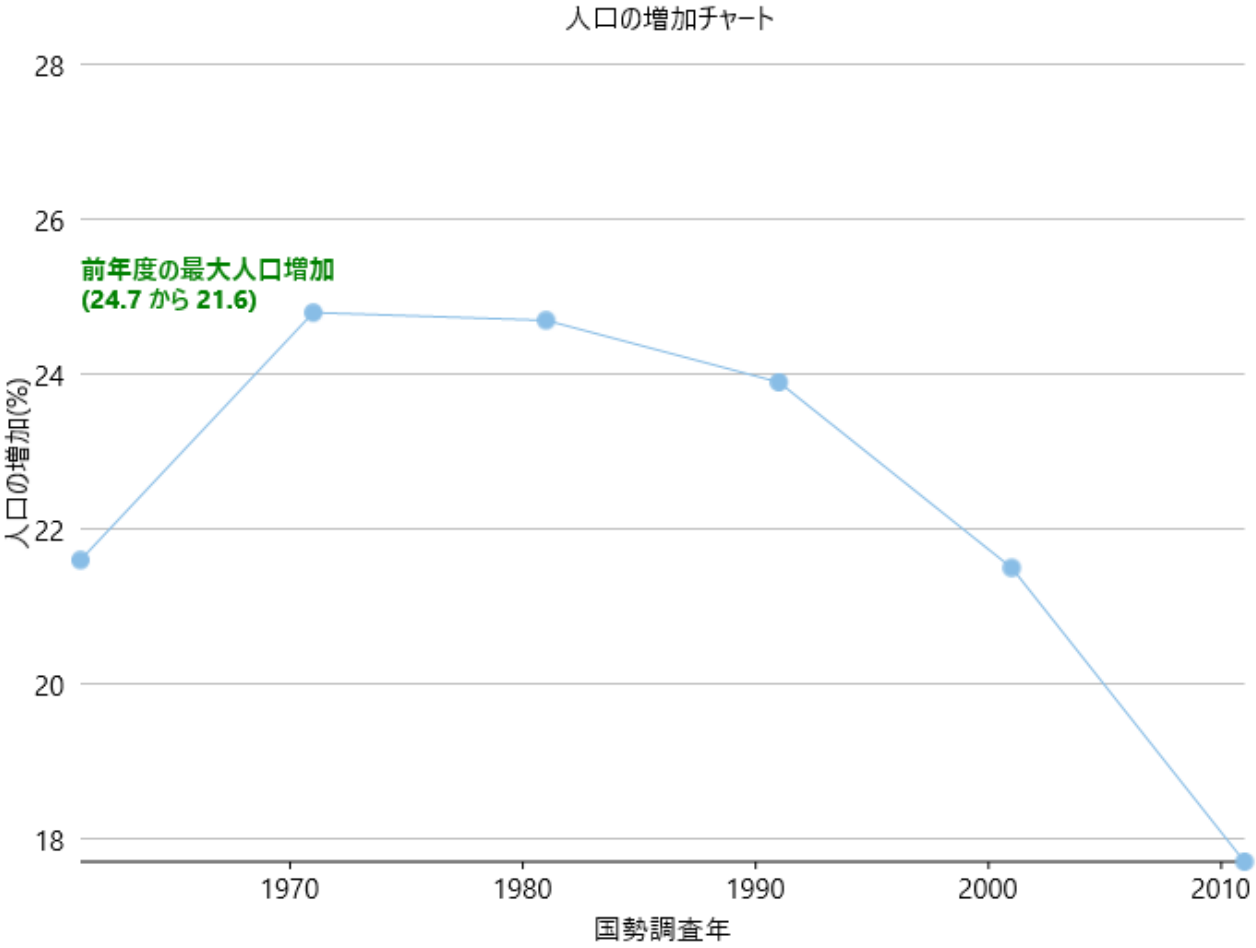
```
<Chart:C1FlexChart.Layers>
    <Annotation:AnnotationLayer>
        <Annotation:AnnotationLayer.Annotations>
            <Annotation:Line Content="最大関税および手数料: 31000"
                             Attachment="DataCoordinate"
                             Start="0,31000"
                             End="12,31000"
                             Position="Top">
                <Annotation:Line.Style>
                    <Chart:ChartStyle  Stroke="DarkTurquoise"
                                       StrokeThickness="3" />
                </Annotation:Line.Style>
            <Annotation:Line.ContentStyle>
                    <Chart:ChartStyle  Stroke="Black"
                                       FontFamily="GenericSansSerif"
                                       FontSize="9"
                                       FontWeight="Bold"/>
                </Annotation:Line.ContentStyle>
            </Annotation:Line>
        </Annotation:AnnotationLayer.Annotations>
    </Annotation:AnnotationLayer>
```

```
</Chart:C1FlexChart.Layers>
```

テキスト注釈

テキスト注釈を使用すると、特定のデータポイントに情報を追加して、データをわかりやすくすることができます。FlexChart では、1 行だけでなく、複数行のテキストをテキスト注釈に追加できます。

次の図に、1961 ～ 2011 年の最大の人口増加率を表示するテキスト注釈を示します。



FlexChart でテキスト注釈を使用するには、**Text** クラスのインスタンスを作成し、そのインスタンスの **Content** プロパティを設定します。

次のコードは、50 年間の人口の増加率を比較します。このコードは、FlexChart のテキスト注釈を追加、配置、カスタマイズする方法を示します。

XAML

```
<Chart:C1FlexChart.Layers>
    <Annotation:AnnotationLayer>
            <Annotation:AnnotationLayer.Annotations>
            <Annotation:Text Content="前年度の最大人口増加&#x0a;(24.7 から 21.6)"
                        Attachment="DataCoordinate"
                        Location="1961,25.15"
                        Position="Top">
                <Annotation:Text.Style>
                    <Chart:ChartStyle Stroke="Green"
                                FontFamily="GenericSansSerif"
```

```
                                    FontSize="14"
                                    FontWeight="Bold" />
                </Annotation:Text.Style>
            </Annotation:Text>
        </Annotation:AnnotationLayer.Annotations>
        </Annotation:AnnotationLayer>
</Chart:C1FlexChart.Layers>
```

画像注釈

画像注釈には視覚的なインパクトがあり、ユーザーがチャートデータをすばやく解釈できるようにします。画像注釈によって有益な情報を伝えるには、ツールチップを追加すると便利です。

次の図に、ツールチップと画像注釈を使用して、ファーストフードチェーンの中で売上高が最大のチェーンを示します。

FlexChart で画像注釈を使用するには、**Image** クラスのインスタンスを作成し、**Source** プロパティで画像のパスを指定してインスタンスの画像を設定します。**Height** および **Width** プロパティを設定して、画像をスケーリングしたり、サイズを調整します。画像注釈にツールチップを追加するには、画像注釈インスタンスで AnnotationBase クラスの **TooltipText** プロパティを設定します。

次のコードは、米国におけるファーストフードチェーンの売上高を比較します。このコードは、FlexChart の画像注釈を追加、配置、カスタマイズする方法を示します。

XAML

```
<Chart:C1FlexChart.Layers>
    <Annotation:AnnotationLayer>
        <Annotation:AnnotationLayer.Annotations>
```

```
            <Annotation:Image Source="C:\\Resources\\image1.png"
                Attachment="DataCoordinate"
                      Location="1,35"
                      Position="Center"
                      Width="68"
                      Height="62"
                      TooltipText="トップファストフードチェーンの中で最も高い収益\n35ビリオン($)">
            </Annotation:Image>
            <Annotation:Image Source="C:\\Resources\\image2.png"
                Attachment="DataCoordinate"
                      Location="2,15"
                      Position="Center"
                      Width="60"
                      Height="61">
            </Annotation:Image>
            <Annotation:Image Source="C:\\Resources\\image3.png"
                Attachment="DataCoordinate"
                      Location="3,11"
                      Position="Center">
            </Annotation:Image>
            <Annotation:Image Source="C:\\Resources\\image4.png"
                Attachment="DataCoordinate"
                      Location="4,8"
                      Position="Center">
            </Annotation:Image>
            <Annotation:Image Source="C:\\Resources\\image5.png"
                Attachment="DataCoordinate"
                      Location="5,7"
                      Position="Center">
            </Annotation:Image>
        </Annotation:AnnotationLayer.Annotations>
    </Annotation:AnnotationLayer>
</Chart:C1FlexChart.Layers>
```

吹き出しの作成

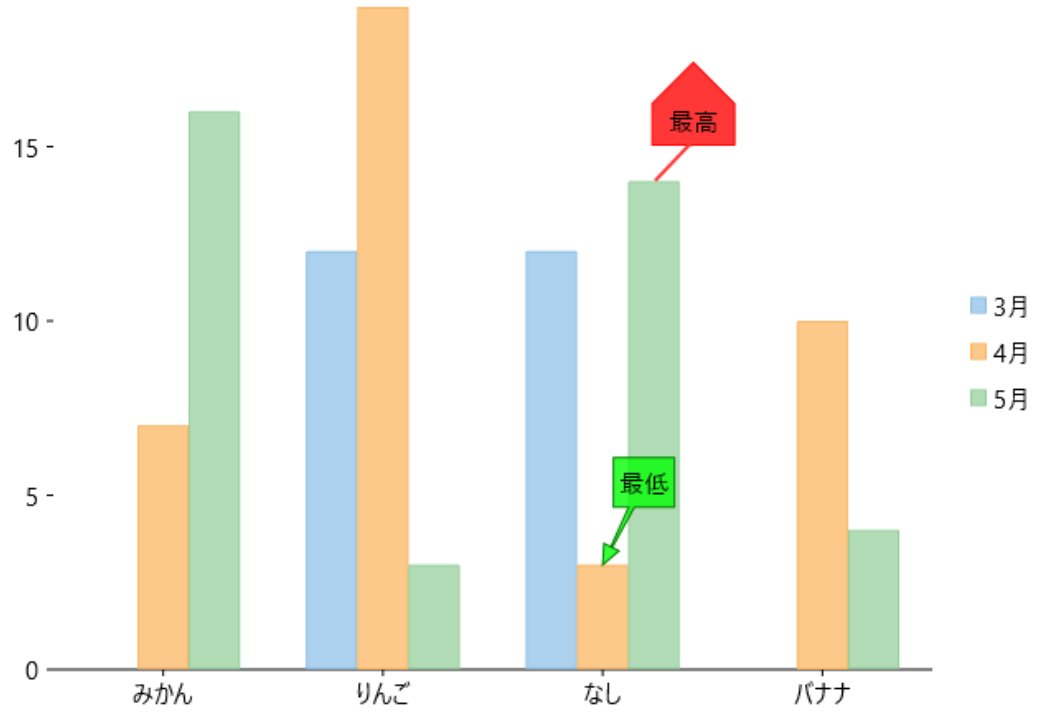
チャート内で吹き出しを使用して、データ系列や個別のデータポイントの詳細を読みやすい形式で表示できます。データポイントに接続された吹き出しは、チャート領域内の見た目の乱雑さを最小限に抑え、チャートデータを見やすく、かつわかりやすくします。FlexChart では、**Polygon** 型の注釈をカスタマイズして、直線または矢印の接続線が付いた吹き出しをチャートに作成できます。

この例では、クイックスタートで作成したサンプルを使用して、矢印付き吹き出しと接続線付きの多角形注釈を作成します。それには、**Points** プロパティと **ContentCenter** プロパティを使用して、それぞれ多角形の頂点の座標と注釈のコンテンツの中心を定義します。

それぞれのデータポイントに接続される吹き出しを作成するには、次の手順に従います。

- **手順 1：接続線付きの注釈の作成**
- **手順 2：矢印付き注釈吹き出しの作成**
- **手順 3：チャートへの注釈のレンダリング**

次の図に、矢印と接続線でデータポイントに接続される多角形注釈を示します。



手順 1:接続線付きの注釈の作成

直線付きの吹き出しを作成するには、次のコードを使用します。

- **Visual Basic**

```
    ...
    lineCallouts.SeriesIndex = 2
    lineCallouts.PointIndex = 2
    lineCallouts.ContentCenter = New Point(-50, 75)
    Dim lineCalloutsPoints As PointCollection = New PointCollection()
    lineCalloutsPoints.Add(New Point(0, 0))
    lineCalloutsPoints.Add(New Point(25, -25))
    lineCalloutsPoints.Add(New Point(50, -25))
    lineCalloutsPoints.Add(New Point(50, -50))
    lineCalloutsPoints.Add(New Point(25, -75))
    lineCalloutsPoints.Add(New Point(0, -50))
    lineCalloutsPoints.Add(New Point(0, -25))
    lineCalloutsPoints.Add(New Point(25, -25))
    lineCalloutsPoints.Add(New Point(0, 0))
    lineCallouts.Points = lineCalloutsPoints
    flexChart.Invalidate()
End Sub
```

- **C#**

```
    // 多角形の線吹き出しの注釈を作成する
    lineCallout.SeriesIndex = 2;
    lineCallout.PointIndex = 2;
    lineCallout.ContentCenter = new Point(25, -40);
    // 線吹き出しの注釈のポイントのリストを作成する
    var lineConnectorPoints = new PointCollection();
    lineConnectorPoints.Add(new Point(0, 0));
    lineConnectorPoints.Add(new Point(25, -25));
    lineConnectorPoints.Add(new Point(50, -25));
    lineConnectorPoints.Add(new Point(50, -50));
    lineConnectorPoints.Add(new Point(25, -75));
    lineConnectorPoints.Add(new Point(0, -50));
    lineConnectorPoints.Add(new Point(0, -25));
    lineConnectorPoints.Add(new Point(25, -25));
    lineConnectorPoints.Add(new Point(0, 0));
    lineCallout.Points = lineConnectorPoints;
    flexChart.Invalidate();
}
```

先頭に戻る

手順 2:矢印付き注釈吹き出しの作成

1. 矢印付きの吹き出しを作成するには、次のコードを使用します。
   - **Visual Basic**

```
Private Sub SetUpAttotations()
    ' 多角形の矢印吹き出しの注釈を作成する
    Dim arrowCalloutContentCenter As Point = New Point(25, -50)
    arrowCallouts.ContentCenter = arrowCalloutContentCenter

    ' GetPointsForArrowCallout()を呼び出して矢印吹き出しのポイントのリストを作成する
    arrowCallouts.Points = GetPointsForArrowCallout(arrowCalloutContentCenter.X,
    arrowCalloutContentCenter.Y, "Low")
    arrowCallouts.SeriesIndex = 1
    arrowCallouts.PointIndex = 2
    ...
```

   - **C#**

```
private void SetUpAnnotations()
{

    // 多角形の矢印吹き出しの注釈を作成する
    var arrowCalloutContentCenter = new Point(25, -50);
    arrowCallout.ContentCenter = arrowCalloutContentCenter;

    // GetPointsForArrowCallout()を呼び出して矢印吹き出しのポイントのリストを作成する
    arrowCallout.Points = GetPointsForArrowCallout(arrowCalloutContentCenter.X,
        arrowCalloutContentCenter.Y, "Low");
    arrowCallout.SeriesIndex = 1;
    arrowCallout.PointIndex = 2;
```

2. GetPointsForArrowCallout() メソッドを定義して、矢印付きの吹き出しのポイントを指定します。
   1. 矢印付きの吹き出し内のコンテンツ文字列のサイズを測定し、これを再利用して、矢印付きの注釈のサイズを計算して設定するには、次のコードを使用します。
      - **Visual Basic**

```
Private Function GetPointsForArrowCallout(centerX As Double, centerY As Double,
content As String) As PointCollection
    Dim size As _Size = _engine.MeasureString(content)
    Return GetPointsForArrowCallout(centerX, centerY, size.Width + 10, size.Height + 10)
End Function
```

      - **C#**

```
PointCollection GetPointsForArrowCallout(double centerX, double centerY, string content)
{
    _Size size = _engine.MeasureString(content);
    return GetPointsForArrowCallout(centerX, centerY, (float)size.Width + 10,
    (float)size.Height + 10);
}
```

   2. 矢印付きの注釈のサイズとポイントを計算するには、次のようにオーバーロードメソッド GetPointsForArrowCallout() を定義します。
      - **Visual Basic**

```
Private Function GetPointsForArrowCallout(centerX As Double, centerY As Double,
rectWidth As Double, rectHeight As Double) As PointCollection
    Dim points As PointCollection = New PointCollection()

    Dim rectLeft As Double = centerX - rectWidth / 2
    Dim rectRight As Double = centerX + rectWidth / 2
    Dim rectTop As Double = centerY - rectHeight / 2
    Dim rectBottom As Double = centerY + rectHeight / 2

    Dim angle As Double = Math.Atan2(-centerY, centerX)
    Dim angleOffset1 As Double = 0.4
    Dim angleOffset2 As Double = 0.04
    Dim arrowHeight As Double = 0.4 * rectHeight
    Dim hypotenuse As Double = arrowHeight / Math.Cos(angleOffset1)
    Dim subHypotenuse As Double = arrowHeight / Math.Cos(angleOffset2)
    Dim isNearBottom As Boolean = Math.Abs(rectTop) > Math.Abs(rectBottom)

    Dim nearHorizontalEdge As Double
    If (isNearBottom) Then
        nearHorizontalEdge = rectBottom
    Else
        nearHorizontalEdge = rectTop
    End If

    Dim isNearRight As Boolean = Math.Abs(rectLeft) > Math.Abs(rectRight)

    Dim nearVerticalEdge As Double
```

```
    If (isNearRight) Then
        nearVerticalEdge = rectRight
    Else
        nearVerticalEdge = rectLeft
    End If

    Dim isHorizontalCrossed As Boolean = Math.Abs(nearHorizontalEdge) >
    Math.Abs(nearVerticalEdge)
    Dim nearEdge As Double
    If (isHorizontalCrossed) Then
        nearEdge = nearHorizontalEdge
    Else
        nearEdge = nearVerticalEdge
    End If

    Dim factor As Int16
    If (nearEdge > 0) Then
        factor = -1
    Else
        factor = 1
    End If

    Dim crossedPointOffsetToCenter As Double
    If (isHorizontalCrossed) Then
        crossedPointOffsetToCenter = rectHeight / (2 * Math.Tan(angle)) * factor
    Else
        crossedPointOffsetToCenter = rectWidth * Math.Tan(angle) * factor / 2
    End If

    '矢印のポイント
    points.Add(New Point(0, 0))
    points.Add(New Point(Math.Cos(angle + angleOffset1) * hypotenuse,
    -Math.Sin(angle + angleOffset1) * hypotenuse))
    points.Add(New Point(Math.Cos(angle + angleOffset2) * subHypotenuse,
    -Math.Sin(angle + angleOffset2) * subHypotenuse))

    '四角形のポイント
    If (isHorizontalCrossed) Then
        points.Add(New Point(-nearEdge / Math.Tan(angle + angleOffset2), nearEdge))
        If (isNearBottom) Then
            points.Add(New Point(rectLeft, rectBottom))
            points.Add(New Point(rectLeft, rectTop))
            points.Add(New Point(rectRight, rectTop))
            points.Add(New Point(rectRight, rectBottom))
        Else
            points.Add(New Point(rectRight, rectTop))
            points.Add(New Point(rectRight, rectBottom))
            points.Add(New Point(rectLeft, rectBottom))
            points.Add(New Point(rectLeft, rectTop))
        End If

        points.Add(New Point(-nearEdge / Math.Tan(angle - angleOffset2), nearEdge))
    Else
        points.Add(New Point(nearEdge, -nearEdge * Math.Tan(angle + angleOffset2)))
        If (isNearRight) Then
            points.Add(New Point(rectRight, rectBottom))
            points.Add(New Point(rectLeft, rectBottom))
            points.Add(New Point(rectLeft, rectTop))
            points.Add(New Point(rectRight, rectTop))
        Else
            points.Add(New Point(rectLeft, rectTop))
            points.Add(New Point(rectRight, rectTop))
            points.Add(New Point(rectRight, rectBottom))
            points.Add(New Point(rectLeft, rectBottom))
        End If

        points.Add(New Point(nearEdge, -nearEdge * Math.Tan(angle - angleOffset2)))
    End If

    '矢印のポイント
    points.Add(New Point(Math.Cos(angle - angleOffset2) * subHypotenuse, -Math.Sin(angle -
    angleOffset2) * subHypotenuse))
    points.Add(New Point(Math.Cos(angle - angleOffset1) * hypotenuse, -Math.Sin(angle -
    angleOffset1) * hypotenuse))
    Return points
End Function
```

■ **C#**

```
PointCollection GetPointsForArrowCallout(double centerX, double centerY,
    double rectWidth, double rectHeight)
{
    var points = new PointCollection();

    double rectLeft = centerX - rectWidth / 2;
    double rectRight = centerX + rectWidth / 2;
    double rectTop = centerY - rectHeight / 2;
    double rectBottom = centerY + rectHeight / 2;

    double angle = Math.Atan2(-centerY, centerX);
    double angleOffset1 = 0.4;
    double angleOffset2 = 0.04;
    double arrowHeight = 0.4 * rectHeight;
    double hypotenuse = arrowHeight / Math.Cos(angleOffset1);
    double subHypotenuse = arrowHeight / Math.Cos(angleOffset2);

    bool isNearBottom = Math.Abs(rectTop) > Math.Abs(rectBottom);
    double nearHorizontalEdge = isNearBottom ? rectBottom : rectTop;
    bool isNearRight = Math.Abs(rectLeft) > Math.Abs(rectRight);
    double nearVerticalEdge = isNearRight ? rectRight : rectLeft;
    bool isHorizontalCrossed = Math.Abs(nearHorizontalEdge) > Math.Abs(nearVerticalEdge);
    double nearEdge = isHorizontalCrossed ? nearHorizontalEdge : nearVerticalEdge;

    int factor = nearEdge > 0 ? -1 : 1;
    double crossedPointOffsetToCenter = isHorizontalCrossed ?
        rectHeight / (2 * Math.Tan(angle)) * factor : rectWidth * Math.Tan(angle) * factor / 2;

    // 矢印のポイント
    points.Add(new Point(0, 0));
    points.Add(new Point(Math.Cos(angle + angleOffset1) * hypotenuse, -Math.Sin(angle +
    angleOffset1) * hypotenuse));
    points.Add(new Point(Math.Cos(angle + angleOffset2) * subHypotenuse, -Math.Sin(angle +
    angleOffset2) * subHypotenuse));

    // 四角形のポイント
    if (isHorizontalCrossed)
    {
        points.Add(new Point(-nearEdge / Math.Tan(angle + angleOffset2), nearEdge));
        if (isNearBottom)
        {
            points.Add(new Point(rectLeft, rectBottom));
            points.Add(new Point(rectLeft, rectTop));
            points.Add(new Point(rectRight, rectTop));
            points.Add(new Point(rectRight, rectBottom));
        }
        else
        {
            points.Add(new Point(rectRight, rectTop));
            points.Add(new Point(rectRight, rectBottom));
            points.Add(new Point(rectLeft, rectBottom));
            points.Add(new Point(rectLeft, rectTop));
        }
        points.Add(new Point(-nearEdge / Math.Tan(angle - angleOffset2), nearEdge));
    }
    else
    {
        points.Add(new Point(nearEdge, -nearEdge * Math.Tan(angle + angleOffset2)));
        if (isNearRight)
        {
            points.Add(new Point(rectRight, rectBottom));
            points.Add(new Point(rectLeft, rectBottom));
            points.Add(new Point(rectLeft, rectTop));
            points.Add(new Point(rectRight, rectTop));
        }
        else
        {
            points.Add(new Point(rectLeft, rectTop));
            points.Add(new Point(rectRight, rectTop));
            points.Add(new Point(rectRight, rectBottom));
            points.Add(new Point(rectLeft, rectBottom));
        }
        points.Add(new Point(nearEdge, -nearEdge * Math.Tan(angle - angleOffset2)));
    }
```

```
            // 矢印のポイント
            points.Add(new Point(Math.Cos(angle - angleOffset2) * subHypotenuse, -Math.Sin(angle -
            angleOffset2) * subHypotenuse));
            points.Add(new Point(Math.Cos(angle - angleOffset1) * hypotenuse, -Math.Sin(angle -
            angleOffset1) * hypotenuse));
            return points;
        }
```

先頭に戻る

手順 3:チャートへの注釈のレンダリング

チャートの注釈をレンダリングするには、次の手順に従います。

1. ンダリングエンジンのグローバルフィールドを定義します。
   - **Visual Basic**

```
Dim _engine As IRenderEngine
```

   - **C#**

```
IRenderEngine _engine;
```

2. 次のコードを使用して、**AnnotationLayer** クラスのインスタンスを作成し、吹き出し注釈を annotationLayer に追加します。
   - **XAML**

```
 <Chart:C1FlexChart.Layers>
 <Annotation:AnnotationLayer>
     <Annotation:AnnotationLayer.Annotations>
         <Annotation:Polygon x:Name="arrowCallout" Content="最低"
                             SeriesIndex="0" PointIndex="1" Attachment="DataIndex">
             <Annotation:Polygon.Style>
                 <Chart:ChartStyle Fill="#C800FF00" Stroke="Green"/>
             </Annotation:Polygon.Style>
         </Annotation:Polygon>

         <Annotation:Polygon x:Name="lineCallout" Content="最高"
                             SeriesIndex="0" PointIndex="4" Attachment="DataIndex">
             <Annotation:Polygon.Style>
                 <Chart:ChartStyle Fill="#C8FF0000" Stroke="Red" />
             </Annotation:Polygon.Style>
         </Annotation:Polygon>
     </Annotation:AnnotationLayer.Annotations>
 </Annotation:AnnotationLayer>
</Chart:C1FlexChart.Layers>
```

3. 吹き出しをレンダリングするには、FlexChart の **Rendered** イベントで次のコードを使用します。
   - **Visual Basic**

```
Private Sub flexChart_Rendered(sender As Object, e As RenderEventArgs)
    If (_engine Is Nothing) Then
        _engine = e.Engine
        SetUpAttotations()
    End If
End Sub
```

   - **C#**

```
private void flexChart_Rendered(object sender, C1.Xaml.Chart.RenderEventArgs e)
{
    if (_engine == null)
    {
        _engine = e.Engine;
        SetUpAnnotations();
    }
}
```

先頭に戻る

FlexChart の凡例

関連する名前によって系列が指定されるまで、FlexChart に凡例は表示されません。関連する名前で系列を設定すると、FlexChart によって凡例が表示されます。

次に、凡例の外観全体のカスタマイズに使用できるプロパティを示します。

プロパティ	説明
LegendPosition	凡例の位置を決定します。

LegendOrientation	凡例の外観を変更します。
LegendTitle	凡例のタイトルを変更します。

凡例テキストの折り返し

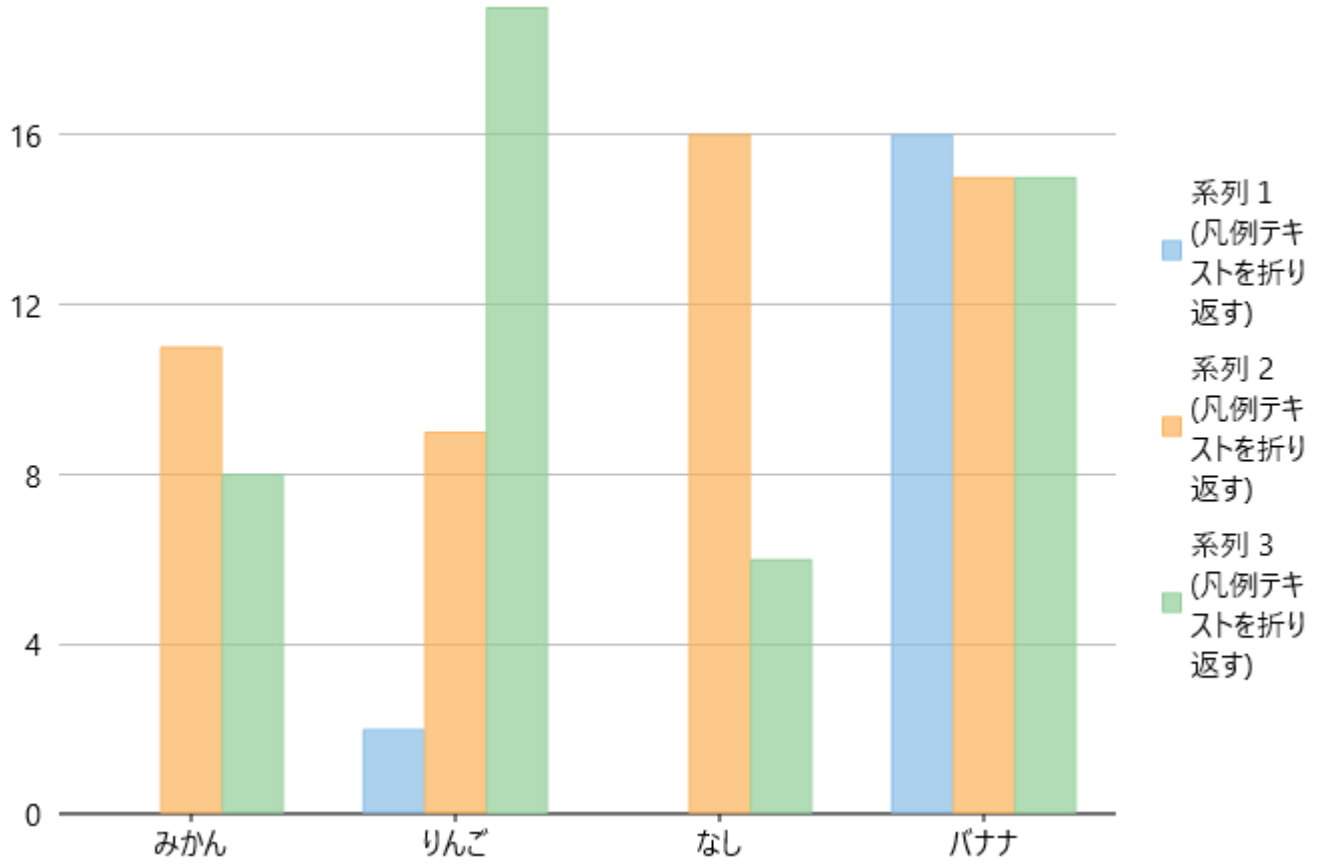
凡例テキストの折り返しは、凡例エントリを切り詰めるか複数の行に折り返して短くします。この機能を使用すると、凡例が占有するスペースを調整して、チャートの表示領域を柔軟に効率よく活用できます。

FlexChart は、各凡例エントリの最大幅を設定する **LegendMaxWidth** プロパティの値を超えた凡例テキストを折り返します。次の 2 つの方法で凡例エントリを管理できます。

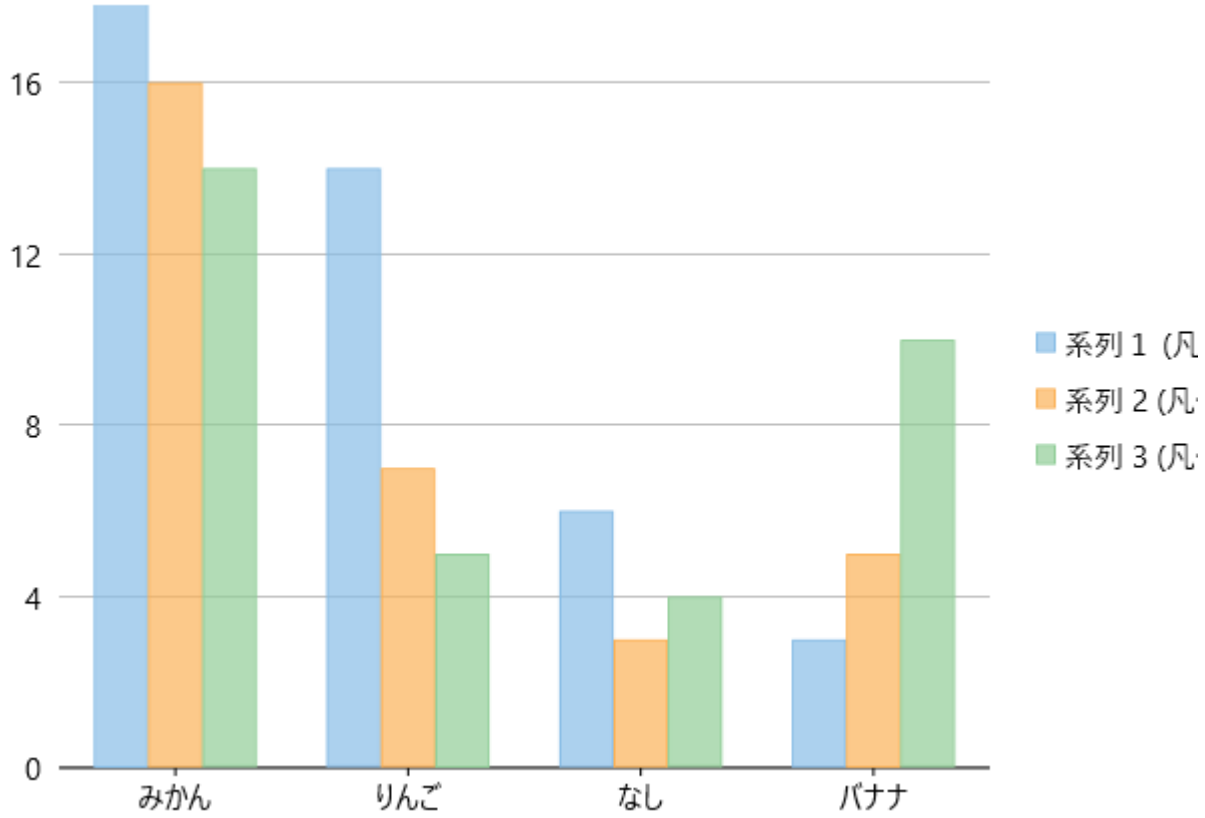
- **折り返し**:このモードでは、凡例エントリを改行して複数行に分けます。テキストの折り返しは、チャート領域に凡例テキスト全体を表示する必要がある場合に便利です。FlexChart で凡例テキストを折り返すには、**LegendTextWrapping** プロパティを**Wrap** に設定します。
- **切り詰め**:このモードでは、テキストの末尾を切り捨てて、凡例エントリを短くします。FlexChart で凡例テキストを折り返すには、**LegendTextWrapping** プロパティを **Truncate** に設定します。

FlexChart の凡例エントリに設定する最大幅は、テキストの折り返しとテキストの切り詰めの両方に影響を与えます。チャートの凡例の最大幅を設定するには、**LegendMaxWidth**プロパティを設定します。凡例エントリの最大幅を大きな値に設定すると、折り返されたり切り詰められる凡例テキストが少なくなります。

次の図に、複数の行に折り返された凡例テキストを示します。



次の図に、切り詰められた凡例テキストを示します。



次のコードは、クイックスタートで作成したサンプルを使用し、FlexChart で凡例テキストの折り返しを実装する方法を示します。

XAML

- **Tab Caption**

```
<Chart:C1FlexChart x:Name="flexChart"
                   ItemsSource="{Binding DataContext.Data}"
                   BindingX="Fruit"
                   LegendTextWrapping="Wrap"
                   LegendPosition="Right"
                   LegendMaxWidth="80">
    <Chart:Series SeriesName="系列1 （凡例テキストを折り返す）"
                  Binding="March"/>
    <Chart:Series SeriesName="系列 2 （凡例テキストを折り返す）"
                  Binding="April"/>
    <Chart:Series SeriesName="系列 3 （凡例テキストを折り返す）"
                  Binding="May"/>
</Chart:C1FlexChart>
```

凡例のグループ化

凡例グループは、名前が示すように、チャート系列の凡例エントリを、それらが表すデータに基づいてカテゴリ化します。そのため、似たデータを持つ複数のチャート系列を凡例内でグループにまとめて、わかりやすく示すことができます。これにより、凡例が整理され、複数の系列を使用してチャートを視覚化および分析する際に役立ちます。

FlexChart では、チャート内の系列の対応する凡例項目をグループ化するために、**LegendGroup** プロパティを使用します。LegendGroup プロパティに文字列値を設定することで、特定の系列または凡例項目が属するグループ名を指定できます。LegendGroup プロパティの値が同じ系列は、凡例内でグループ化されます。ただし、LegendGroup プロパティが定義されていない系列は、0 番目のグ

ループに入ります。

LegendGroup プロパティの値は、対応する凡例項目の上にグループタイトルとして表示されます。ただし、0 番目のグループに属する凡例項目は、グループタイトルなしで表示されます。

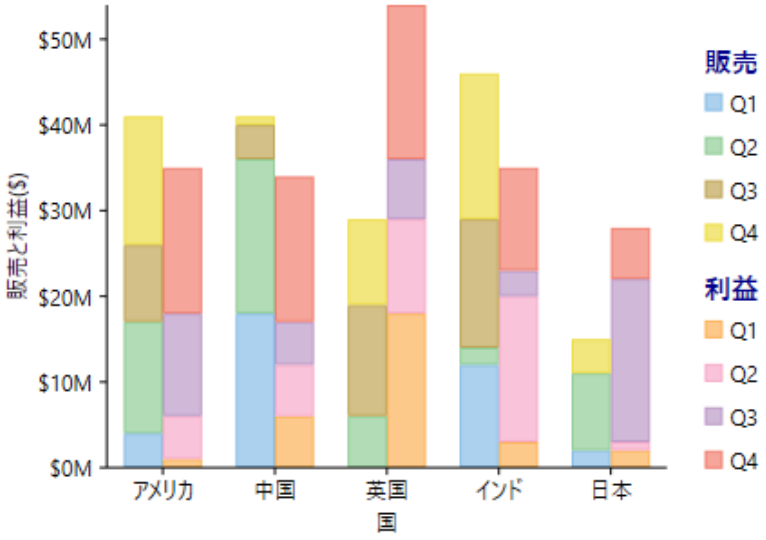
凡例グループの配置

凡例グループは、凡例の位置に応じて自動的に相互の配置が決定されます。たとえば、凡例がチャートの上または下に配置される場合、凡例グループは横に並べられます。逆に、凡例がチャートの左または右に配置される場合、凡例グループは縦に並べられます。

凡例グループのスタイル設定

FlexChart は、凡例グループのスタイル設定と書式設定もサポートします。凡例グループのタイトルの外観は、**LegendGroupHeaderStyle** プロパティを指定してカスタマイズできます。

次の図に、ある企業の国ごと、四半期ごとの売上および利益をプロットする積層グラフを示します。ここでは、すばやく簡単に分析できるように、積み重ねられる系列に基づいて凡例項目をグループ化しました。この図は、凡例と凡例グループが縦方向に配置される様子と、グループタイトルの外観をカスタマイズできることも示しています。



次のコードスニペットは、これらの系列の **LegendGroup** プロパティを目的のグループ名に設定して、それぞれの系列の凡例をグループ化する方法を示します。また、コードスニペットは、**GroupHeaderStyle** プロパティを使用して、凡例グループのヘッダーのスタイルを設定する方法も示します。

- **Xaml**

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
<Chart:C1FlexChart Name="flexChart2"
                   Stacking="Stacked"
                   ItemsSource="{Binding DataContext.Data}"
                   BindingX="Country">

    <!--凡例項目をグループ化します-->
        <Chart:Series SeriesName="Q1" Binding="SalesQ1" LegendGroup="Sales" StackingGroup="0" />
    <Chart:Series SeriesName="Q1" Binding="ProfitQ1" LegendGroup="Profit" StackingGroup="1" />
    <Chart:Series SeriesName="Q2" Binding="SalesQ2" LegendGroup="Sales" StackingGroup="0" />
    <Chart:Series SeriesName="Q2" Binding="ProfitQ2" LegendGroup="Profit" StackingGroup="1" />
    <Chart:Series SeriesName="Q3" Binding="SalesQ3" LegendGroup="Sales" StackingGroup="0" />
    <Chart:Series SeriesName="Q3" Binding="ProfitQ3" LegendGroup="Profit" StackingGroup="1" />
    <Chart:Series SeriesName="Q4" Binding="SalesQ4" LegendGroup="Sales" StackingGroup="0" />
    <Chart:Series SeriesName="Q4" Binding="ProfitQ4" LegendGroup="Profit" StackingGroup="1" />
    <Chart:C1FlexChart.AxisY>
        <Chart:Axis Format="$0M"
                    Labels="True"
                    Title="Million $"
```

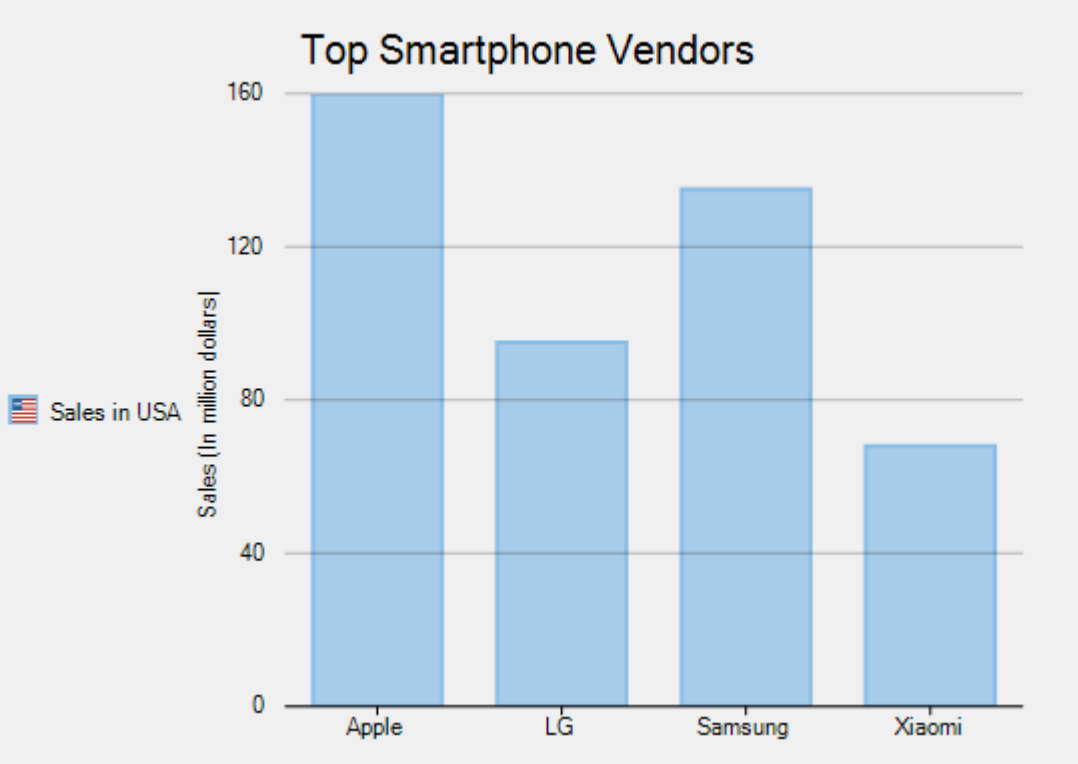
```
                    Min="0" MajorGrid="True"
                    AxisLine="False"
                    Position="Left"
                    MajorTickMarks="None" />
    </Chart:C1FlexChart.AxisY>

    <!--凡例グループのヘッダーをスタイル設定します-->
    <Chart:C1FlexChart.LegendGroupHeaderStyle>
        <Chart:ChartStyle Stroke="DarkBlue"
                          FontFamily="Cambria"
                          FontStyle="Normal"
                          FontSize="15" FontWeight="Bold"/>
    </Chart:C1FlexChart.LegendGroupHeaderStyle>
</Chart:C1FlexChart>
```

カスタムの凡例アイコン

FlexChart allows you to apply custom image or icon for customizing the legend items. To enable FlexChart to display custom legend icon, implement the **GetLegendItemImageSource** method provided by ISeries interface. This method primarily accepts two parameters; index and _size. The index parameter refers to the legend item position and _size parameter refers to the default legend icon size.

In the example code below, we have implemented the GetLegendItemImageSource method to customize the image size and draw a border around it. This method then returns the image object. To apply the custom legend icon add object of the class SeriesWithPointLegendItems to the chart Series collection.

The image shows how FlexChart appears after using custom legend icon.



Use the following code snippet to implement custom legend icon.

XAML

```
<Chart:C1FlexChart x:Name="flexChart" ItemsSource="{Binding SmartPhoneVendors}"
Binding="Sales" BindingX="Name" Header="Top Smartphone Vendors" Grid.Row="1">
```

```
            <Chart:C1FlexChart.HeaderStyle>
                <Chart:ChartStyle FontSize="15" FontFamily="GenericSansSerif"/>
            </Chart:C1FlexChart.HeaderStyle>
</Chart:C1FlexChart>
```

Code

HTML

```
public partial class LegendItems
    {
        static List<SmartPhoneVendor> vendors = new List<SmartPhoneVendor>();
        static Image LegendIconImage = Properties.Resources.usa;
        Series customSeries;
        public LegendItems()
        {
            InitializeComponent();
            vendors = SmartPhoneVendors();

            // カスタム系列を追加します
            customSeries = new SeriesWithPointLegendItems();
            customSeries.Name = "Sales in USA";
            flexChart1.Series.Add(customSeries);
            flexChart1.Legend.Position = Position.Left;
            flexChart1.ToolTip.Content = "{seriesName}\r\n{value}";

        }
        public class SeriesWithPointLegendItems : Series, ISeries
        {
            object ISeries.GetLegendItemImageSource(int index, ref C1.Chart._Size
imageSize)
            {
                {
                    // 元の画像/ロゴはすべて50x50ピクセルです
                    // ここでは、ロゴの周りに5ピクセルのボーダーが追加された新しいイメージに置き換え
られます
                    imageSize.Height = 60;
                    imageSize.Width = 60;

                    SmartPhoneVendor vendor = vendors.ElementAt(index);
                    if (LegendIconImage != null && LegendIconImage.Width != 60)
                    {
                        Bitmap bmp = new Bitmap(60, 60);
                        using (SolidBrush sb = new SolidBrush(vendor.Color))
                        {
                            using (Graphics g = Graphics.FromImage(bmp))
                            {
                                Rectangle r = new Rectangle(0, 0,
(int)imageSize.Width, (int)imageSize.Height);
                                using (Pen p = new Pen(sb))
                                {
                                    g.DrawRectangle(p, r);
```

```
                                }
                                g.FillRectangle(sb, r);

                                Point ci = new Point((int)(0.5 * (imageSize.Width
- LegendIconImage.Width)),
                                    (int)(0.5 * (imageSize.Height -
LegendIconImage.Height)));
                                g.DrawImage(LegendIconImage, ci);
                            }
                        }
                        LegendIconImage = bmp;
                    }
                    // ロゴビットマップの元のサイズを維持しますが、グラフウィンドウがビットマップを適切
に表示するには
                // 小さすぎる場合は、サイズを縮小します
                    Size bounds = this.Chart.ClientSize;
                    double divadj = (bounds.Height > 800) ? 12 : 25;
                    double fracHeight = bounds.Height / divadj;
                    if (fracHeight < imageSize.Height)
                        imageSize.Width = imageSize.Height = fracHeight;
                    return LegendIconImage;
                }
            }
        }
        private static List<SmartPhoneVendor> SmartPhoneVendors()
        {

            vendors.Add(new SmartPhoneVendor()
            {
                Name = "Apple",
                Color = Color.FromArgb(136, 189, 230),
                Sales = 350,
            });
            vendors.Add(new SmartPhoneVendor()
            {
                Name = "LG",
                Color = Color.FromArgb(251, 178, 88),
                Sales = 120,
            });

            vendors.Add(new SmartPhoneVendor()
            {
                Name = "Samsung",
                Color = Color.FromArgb(188, 153, 199),
                Sales = 280,
            });

            vendors.Add(new SmartPhoneVendor()
            {
                Name = "Xiaomi",
                Color = Color.FromArgb(240, 126, 110),
                Sales = 68,
            });
```

```
            return vendors;
        }
        public class SmartPhoneVendor
        {
            public string Name { get; set; }
            public double Sales { get; set; }
            public Color Color { get; set; }
        }
    }
```

FlexChart の系列

系列は 1 組のデータです。より具体的に言えば、互いに関連するデータポイントとしてチャートにプロットされるデータです。

FlexChart 内の 1 つの系列は **Series** オブジェクトで表され、これがチャートにプロットされるデータ全体を提供します。さらに、**Flexchart.Series** コレクションは、コントロール内のすべてのデータ系列（**Series** オブジェクト）で構成されます。

FlexChart 内の系列には、次のプロパティを割り当てることができます。

- X 軸（**Series.AxisX**）
- Y 軸（**Series.AxisY**）
- 系列の Y 値を含むプロパティ（**Series.Binding**）
- 系列の X 値を含むプロパティ（**Series.BindingX**）
- チャートタイプ（**Series.ChartType**）
- 系列データを含むオブジェクトのコレクション（**Series.ItemsSource**）
- 名前（**Series.SeriesName**）

系列を構成するデータポイントのコレクションは、次のプロパティを使用してカスタマイズできます。

- 系列の各データポイントで使用されるマーカーの形を設定する：**Series.SymbolMarker**
- 系列のレンダリングに使用されるシンボルのサイズを設定する：**Series.SymbolSize**
- 系列のデータポイントで使用されるシンボルスタイルを設定する：**Series.SymbolStyle**

これらのプロパティを系列に設定すると、すべてのデータポイントで同じ設定が継承されます。

FlexChart の Series オブジェクトに関する主要な情報については、以下のリンクを参照してください。

- 系列の作成と追加
- 系列へのデータの追加
- 各種データの強調
- 系列のカスタマイズ
- ウォータフォール
- 箱ひげ図
- 誤差範囲
- 積層グループ

系列の作成と追加

デフォルトでは、**FlexChart for UWP** は、設計時、実行時共に、ダミーデータを含む 3 つの系列を表示します。しかし、独自のデータを提供し、そのデータを使用して系列を表示することができます。FlexChart にデータを提供する方法については、「データの提供」を参照してください。

まず、**Series** オブジェクトを使用して系列を作成する必要があります。次に、**FlexChart.Series** コレクションプロパティの **Add** メソッドを使用して、FlexChart Series コレクションに系列を追加します。

次のコードは、実行時に FlexChart に系列を作成して追加する方法を示します。

XAML

- **タブキャプション**

```
<Chart:Series Binding="Y" BindingX="X" SeriesName="Series 4">
    <Chart:Series.ItemsSource>
        <PointCollection>
            <Foundation:Point>1,8</Foundation:Point>
            <Foundation:Point>2,12</Foundation:Point>
            <Foundation:Point>3,10</Foundation:Point>
            <Foundation:Point>4,12</Foundation:Point>
            <Foundation:Point>5,15</Foundation:Point>
        </PointCollection>
    </Chart:Series.ItemsSource>
</Chart:Series>
```

コード

C# copyCode

```
C1.Xaml.Chart.Series series4 = new C1.Xaml.Chart.Series();
flexChart.Series.Add(series4);
```

系列へのデータの追加

系列へのデータの追加については、強力な連結による方法が提供されています。FlexChart 内の系列を複数のデータソースと連結することができるため、複数のデータソースのデータを組み合わせることができます。複数のデータソースのデータをプロットするには、**Series.ItemsSource** プロパティを使用する必要があります。

次のコードを参照してください。次のコードでは、**DataCreator.cs** クラスを使用してデータを生成しています。

XAML

- **タブキャプション**

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Chart:C1FlexChart x:Name="flexChart" ChartType="Scatter">
        <Chart:C1FlexChart.Series>
            <Chart:Series x:Name="Function1"
            SeriesName="Function 1" BindingX="XVals" Binding="YVals"></Chart:Series>
            <Chart:Series x:Name="Function2"
            SeriesName="Function 2" BindingX="XVals" Binding="YVals"></Chart:Series>
        </Chart:C1FlexChart.Series>
    </Chart:C1FlexChart>
    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup>
            <VisualState x:Name="WideLayout">
                <VisualState.StateTriggers>
                    <AdaptiveTrigger MinWindowWidth="540"></AdaptiveTrigger>
                </VisualState.StateTriggers>
            </VisualState>
            <VisualState x:Name="NarrowLayout">
```

```
                <VisualState.StateTriggers>
                    <AdaptiveTrigger MinWindowWidth="0"></AdaptiveTrigger>
                </VisualState.StateTriggers>
                <VisualState.Setters>
                    <Setter Target="flexChart.LegendPosition" Value="Top"></Setter>
                </VisualState.Setters>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
</Grid>
```

コード

DataCreator.cs copyCode

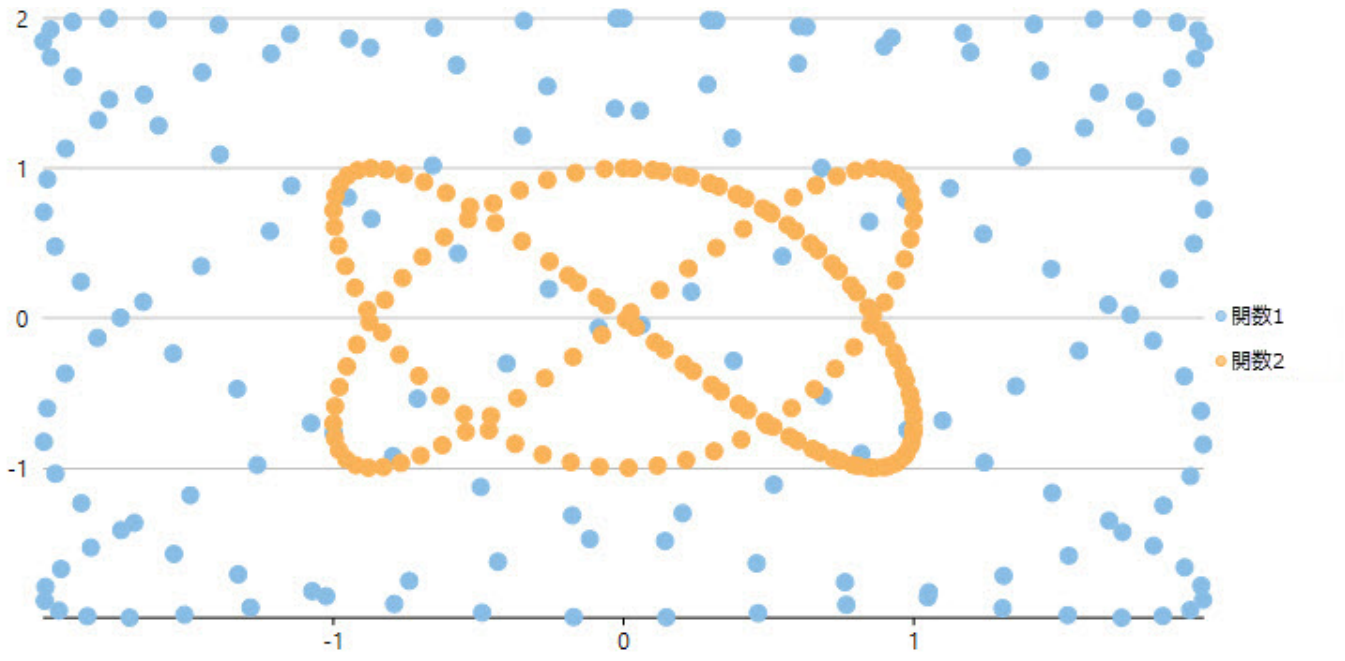
```
class DataCreator
{
    public delegate double MathActionDouble(double num);
    public delegate double MathActionInt(int num);

   public static List<DataPoint> Create(MathActionDouble function,
          double from,double to,double step)
    {
        var result = new List<DataPoint>();
        var count = (to - from) / step;

        for (double r = from; r < to; r += step)
        {
            result.Add(new DataPoint()
            {
                XVals = r,
                YVals = function(r)
            });
        }
        return result;
    }

  public static List<DataPoint> Create(MathActionInt function, int from,
         int to, int step)
    {
        var result = new List<DataPoint>();
        var count = (to - from) / step;

        for (int r = from; r < to; r += step)
        {
            result.Add(new DataPoint()
            {
                XVals = r,
                YVals = function(r)
            });
        }
        return result;
    }
```

```
    public static List<DataPoint> Create(MathActionDouble functionX,
         MathActionDouble functionY, int ptsCount)
    {
        var result = new List<DataPoint>();

        for (double i = 0; i < ptsCount; i++)
        {
            result.Add(new DataPoint()
            {
                XVals = functionX(i),
                YVals = functionY(i)
            });
        }
        return result;
    }
}

public class DataPoint
{
    public double XVals { get; set; }
    public double YVals { get; set; }
}
```

MainWindow.xaml.cs copyCode

```
public sealed partial class MainPage : Page
{
    List<DataPoint> _function1Source;
    List<DataPoint> _function2Source;

    public MainPage()
    {
        this.InitializeComponent();
        this.Loaded += SeriesBinding_Loaded;
    }

    private void SeriesBinding_Loaded(object sender, RoutedEventArgs e)
    {
        SetupChart();
    }

    void SetupChart()
    {
        flexChart.BeginUpdate();
        this.Function1.ItemsSource = Function1Source;
        this.Function2.ItemsSource = Function2Source;
        flexChart.EndUpdate();
    }

    public List<DataPoint> Function1Source
    {
        get
        {
            if (_function1Source == null)
```

```
            {
                _function1Source = DataCreator.Create(x => 2 * Math.Sin(0.16 * x),
                 y => 2 * Math.Cos(0.12 * y), 160);
            }

            return _function1Source;
        }
    }

    public List<DataPoint> Function2Source
    {
        get
        {
            if (_function2Source == null)
            {
                _function2Source = DataCreator.Create(x => Math.Sin(0.1 * x),
                 y => Math.Cos(0.15 * y), 160);
            }

            return _function2Source;
        }
    }
}
```



各種データの強調

チャートでさまざまなタイプのデータを強調したいというニーズはよくあり、しかもそれは重要です。識別しやすいチャートデータほど解釈や理解が容易なので、さまざまなタイプのデータを強調して識別できるようにすることは必須です。

このニーズに応えるために、FlexChart では、2 つ以上のチャートタイプを 1 つのチャートに組み合わせることができます。たとえば、折れ線シンボルグラフを縦棒グラフと組み合わせて、チャートのデータを解釈しやすくすることができます。**ChartType**

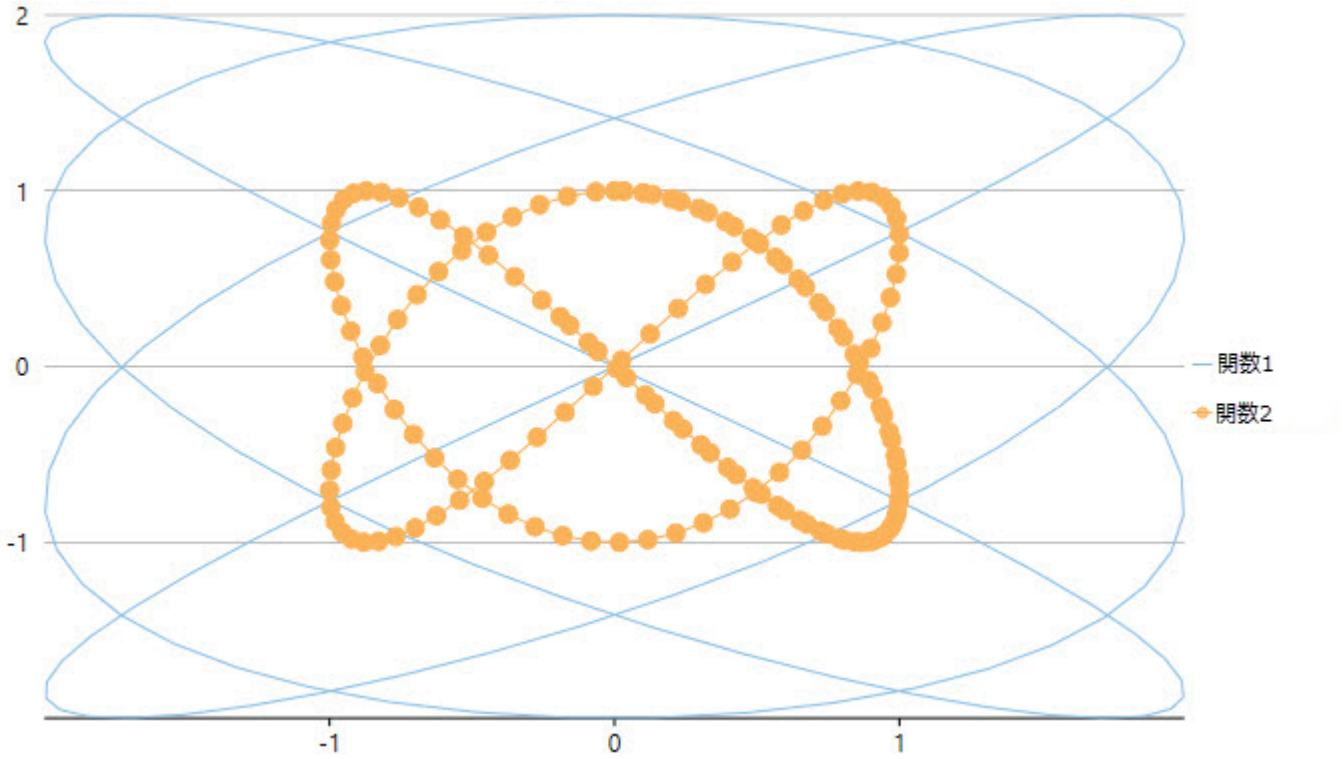
プロパティを使用すると、系列ごとに系列レベルでチャートタイプを指定でき、その結果、複数のチャートタイプが含まれるチャートを作成することができます。

次のコードは、2 つのチャートタイプを 1 つのチャートに組み合わせています。

XAML

- **タブキャプション**

```
<Chart:C1FlexChart.Series>
    <Chart:Series ChartType="Line" x:Name="Function1" SeriesName="Function 1"
    BindingX="XVals" Binding="YVals"></Chart:Series>
    <Chart:Series ChartType="LineSymbols" x:Name="Function2" SeriesName="Function 2"
     BindingX="XVals" Binding="YVals"></Chart:Series>
</Chart:C1FlexChart.Series>
```



系列のカスタマイズ

系列がチャートに表示されたら、表示された系列をカスタマイズして、より効率的な管理を行うことができます。

FlexChart では、プロット領域、凡例、またはその両方で系列を表示または非表示にして系列をカスタマイズできます。チャートのスペースには限りがあるため、チャートに表示される系列が大量にある場合は、系列の管理が確実に必要になります。

FlexChart では、系列の **Visibility** プロパティを使用して系列を管理できます。**Visibility** プロパティは、**SeriesVisibility** 列挙型の値を受け取ります。

このプロパティを以下に示す値に設定して、系列を表示または非表示にすることができます。

値	説明
SeriesVisibility.Visible	系列はプロットと凡例の両方に表示されます。

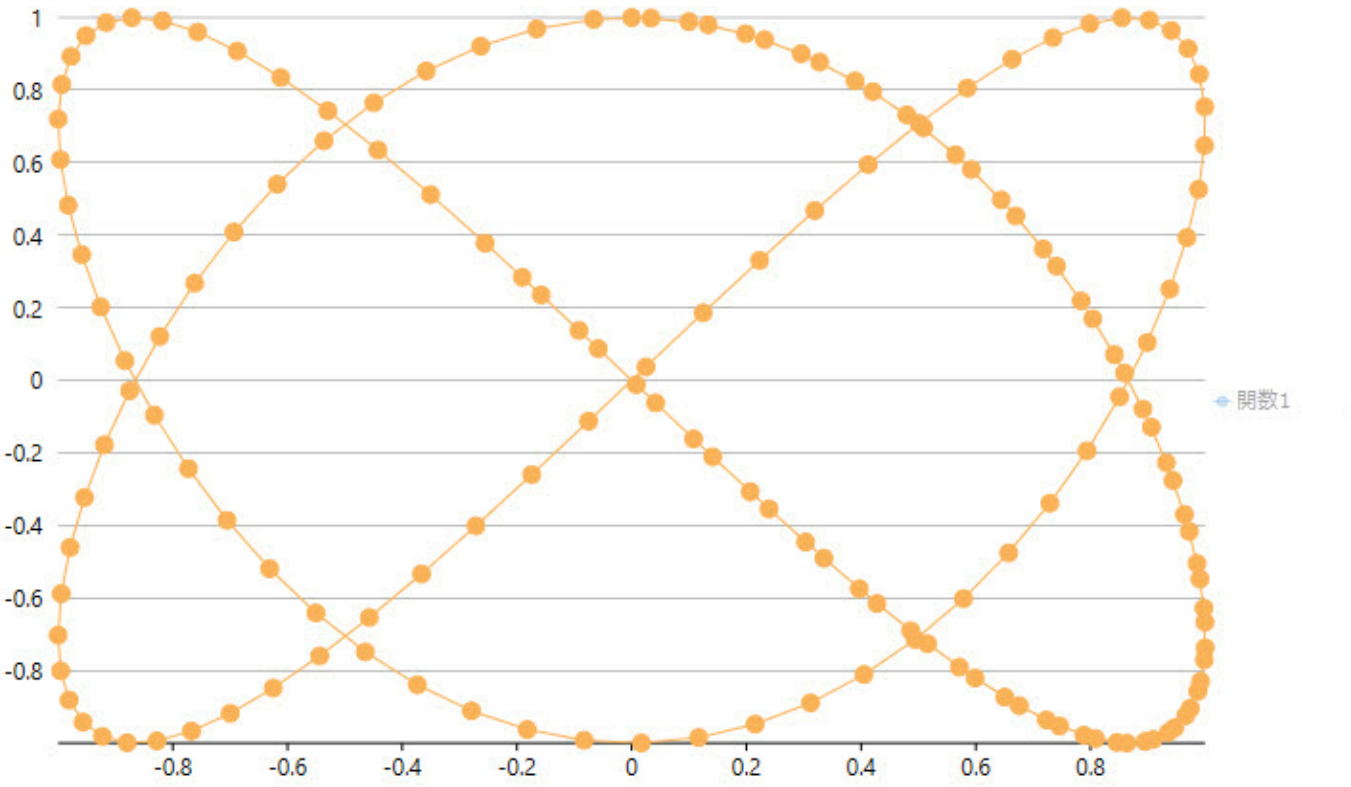
SeriesVisibility.Plot	系列はプロットには表示されますが、凡例には表示されません。
SeriesVisibility.Legend	系列は凡例には表示されますが、プロットには表示されません。
SeriesVisibility.Hidden	系列はプロットと凡例のどちらにも表示されません。

次のコードスニペットを参照してください。

XAML

- **タブキャプション**

```
<Chart:C1FlexChart.Series>
    <Chart:Series Visibility="Hidden" x:Name="Function1" SeriesName="Function 1"
    BindingX="XVals" Binding="YVals"></Chart:Series>
    <Chart:Series Visibility="Plot" x:Name="Function2" SeriesName="Function 2"
    BindingX="XVals" Binding="YVals"></Chart:Series>
</Chart:C1FlexChart.Series>
```



このほかに、FlexChart のさまざまなパレットを設定して、系列の視覚効果を高めることができます。詳細については、「FlexChart: パレットの設定」を参照してください。

さまざまなシンボルスタイルを使用して、チャートに視覚効果の高い系列をレンダリングすることもできます。詳細については、「系列のシンボルスタイル」を参照してください。

ウォータフォール

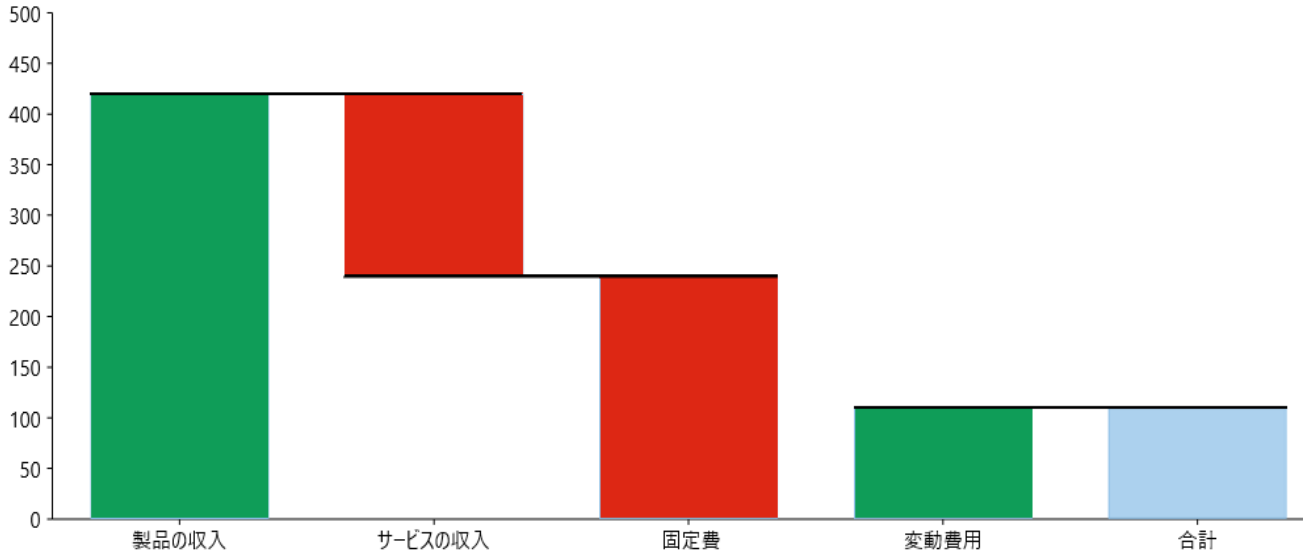
ウォータフォール系列を使用すると、一連の正の値と負の値の累積的な影響を把握することができます。一連の正の値と負の値が初期値にどのように影響していくかを理解できると有益です。ウォータフォール系列は、正の値と負の値を容易に区別できるよう

に縦棒を色分けして描画します。一般に、最初の値と最後の値は合計を表す縦棒で表され、中間の値は浮いた状態の縦棒で表されます。ウォータフォール系列は、カテゴリテキストの列があり、正の値と負の値が混在している場合に使用することをお勧めします。主に、在庫分析や売上分析などの定量分析がこれに該当します。これらのエンティティの定量値が増減する場合に、チャートにはその段階的な変化が示されます。

FlexChart の次の機能を実装およびカスタマイズして、ウォータフォール系列によるデータ視覚化をさらに強化できます。

- **接続線**: 接続線は、色分けされた縦棒を接続して、チャート内のデータの流れを表示します。接続線を表示するには、**WaterFall** クラスの **ConnectorLines** プロパティを true に設定します。
- **接続線のカスタマイズ**: 表示した接続線は、**ConnectorLineStyle** プロパティを使用してカスタマイズできます。このプロパティから、**ChartStyle** クラスのスタイル設定プロパティにアクセスできます。
- **縦棒のカスタマイズ**: 正の値、負の値、合計値をわかりやすく区別できるように、これらの値を示す縦棒にさまざまなスタイルを適用することができます。そのために、Waterfall クラスで提供されている **RisingStyle**、**FallingStyle**、**TotalStyle**、**StartStyle** など、さまざまなプロパティを使用できます。

次の図に、一連の正の値と負の値の累積的な影響を表したウォータフォール系列を示します。



FlexChart でウォータフォール系列を使用するには、まず **Waterfall** クラスのインスタンスを作成します。このクラスは、**Series** クラスを継承します。次に、**C1FlexChart** クラスで提供されている **Series** プロパティを使用して、作成したインスタンスを FlexChart Series コレクションに追加します。

次のコードスニペットは、FlexChart でウォータフォール系列を使用する際に、さまざまなプロパティを設定する方法を示します。このコードスニペットでは、まず **DataCreator** クラスを作成してチャートのデータを生成し、次に系列をデータソースに連結しています。

- **Visual Basic**

```
Class DataCreator
    Public Shared Function CreateData() As List(Of DataItem)
        Dim data = New List(Of DataItem)()
        data.Add(New DataItem("製品の収入", 420))
        data.Add(New DataItem("サービスの収入", -180))
        data.Add(New DataItem("固定費", 130))
        data.Add(New DataItem("変動費用", -20))
        Return data
    End Function
End Class

Public Class DataItem
    Public Sub New(costs__1 As String, amount__2 As Integer)
        Costs = costs__1
        Amount = amount__2
```

```
    End Sub

    Public Property Costs() As String
        Get
            Return m_Costs
        End Get
        Set
            m_Costs = Value
        End Set
    End Property
    Private m_Costs As String
    Public Property Amount() As Integer
        Get
            Return m_Amount
        End Get
        Set
            m_Amount = Value
        End Set
    End Property
    Private m_Amount As Integer
End Class
```

- **C#**

```
using System.Collections.Generic;

namespace Waterfall
{
    class DataCreator
    {
        public static List<DataItem> CreateData()
        {
            var data = new List<DataItem>();
            data.Add(new DataItem("製品の収入", 420));
            data.Add(new DataItem("サービスの収入", -180));
            data.Add(new DataItem("固定費", 130));
            data.Add(new DataItem("変動費用", -20));
            return data;
        }
    }

    public class DataItem
    {
        public DataItem(string costs, int amount)
        {
            Costs = costs;
            Amount = amount;
        }

        public string Costs { get; set; }
        public int Amount { get; set; }
    }
}
```

次に、FlexChart をデータソースに連結するためのコードスニペットを示します。

XAML

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:Waterfall"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:Chart="using:C1.Xaml.Chart"
    xmlns:Foundation="using:Windows.Foundation"
    x:Class="Waterfall.MainPage"
    DataContext="{Binding RelativeSource={RelativeSource Mode=Self}}"
    mc:Ignorable="d" Height="528.558" Width="712.292">
    <Grid Margin="-140,0,-223.667,-0.333">
        <Chart:C1FlexChart x:Name="flexChart"
                        BindingX="Costs"
                        ItemsSource="{Binding DataContext.Data}"  Margin="10,127,10,10">
            <Chart:C1FlexChart.Series>
                <Chart:Waterfall Binding="Amount" ConnectorLines="True"
                 ShowTotal="True" ShowIntermediateTotal="True">
                    <Chart:Waterfall.StartStyle>
                        <Chart:ChartStyle Fill="#7dc7ed" />
                    </Chart:Waterfall.StartStyle>
                    <Chart:Waterfall.FallingStyle>
                        <Chart:ChartStyle Fill="#dd2714" />
                    </Chart:Waterfall.FallingStyle>
                    <Chart:Waterfall.RisingStyle>
                        <Chart:ChartStyle Fill="#0f9d58" Stroke="#0f9d58" />
                    </Chart:Waterfall.RisingStyle>
                    <Chart:Waterfall.IntermediateTotalStyle>
                        <Chart:ChartStyle Fill="#7dc7ed" />
                    </Chart:Waterfall.IntermediateTotalStyle>
                    <Chart:Waterfall.TotalStyle>
                        <Chart:ChartStyle Fill="#7dc7ed" />
                    </Chart:Waterfall.TotalStyle>
                    <Chart:Waterfall.ConnectorLineStyle>
                        <Chart:ChartStyle Stroke="#333" StrokeDashArray="5,5"/>
                    </Chart:Waterfall.ConnectorLineStyle>
                </Chart:Waterfall>
            </Chart:C1FlexChart.Series>
            <Chart:C1FlexChart.AxisY>
                <Chart:Axis Min="0" Max="500"></Chart:Axis>
            </Chart:C1FlexChart.AxisY>
        </Chart:C1FlexChart>
    </Grid>
</Page>
```

コード

C# copyCode

```
// FlexChartの系列コレクションをクリアします。
flexChart.Series.Clear();

// ウォータフォール系列のインスタンスを作成します。
C1.Xaml.Chart.Waterfall waterFall = new C1.Xaml.Chart.Waterfall();

// Seriesコレクションにインスタンスを追加します。
flexChart.Series.Add(waterFall);

// FlexChartのY値を含むフィールドをバインドします。
waterFall.Binding = "Amount";

// FlexChartのX値を含むフィールドをバインドします。
flexChart.BindingX = "Costs";

// ConnectorLinesプロパティを設定します。
waterFall.ConnectorLines = true;
```

```
// ShowTotalプロパティを設定します。
waterFall.ShowTotal = true;
```

VB copyCode

```
' FlexChartの系列コレクションをクリアします。
flexChart.Series.Clear()

' ウォータフォール系列のインスタンスを作成します。
Dim waterFall As New C1.Xaml.Chart.Waterfall()

' Seriesコレクションにインスタンスを追加します。
flexChart.Series.Add(waterFall)

' FlexChartのY値を含むフィールドをバインドします。
waterFall.Binding = "Amount"

' FlexChartのX値を含むフィールドをバインドします。
flexChart.BindingX = "Costs"

' ConnectorLinesプロパティを設定します。
waterFall.ConnectorLines = True

' ShowTotalプロパティを設定します。
waterFall.ShowTotal = True
```

箱ひげ図

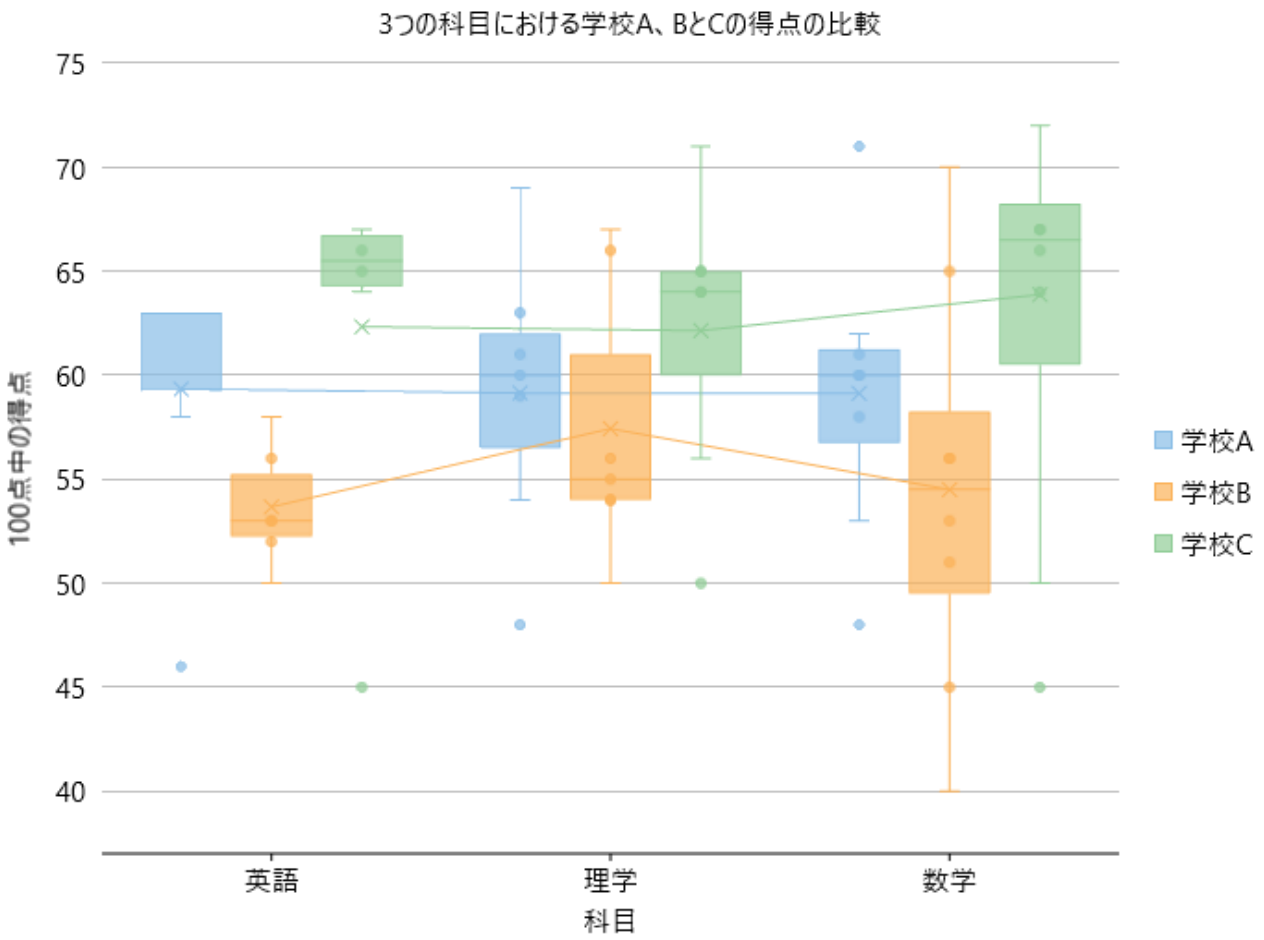
箱ひげ図系列を使用すると、データのグループを範囲、四分位数、中央値で表示できます。その名前が示すとおり、系列データが箱とひげで描写されます。

箱は、四分位数（上下）と中央値を示す範囲です。一方、ひげは、箱から垂直に伸びる線です。これらの線は、上下の四分位数の外側にあるデータのばらつきを示します。さらに、これらの線の外側にあるポイントは、異常値と呼ばれます。

箱ひげ系列は、統計分布を視覚化したり、複数のデータセットをグラフィカルに検証する場合に理想的です。FlexChart では、この系列で次の機能を使用できます。

- **四分位数**：四分位数を計算するために中央値を含めるか除外するかを指定します。四分位数の計算を指定するには、**QuartileCalculation** 列挙を使用して **QuartileCalculation** プロパティを設定します。
- **内側ポイント**：**ShowInnerPoints** プロパティを設定して、内側ポイントを表示するかどうかを示します。
- **異常値**：**ShowOutliers** プロパティを設定して、異常値を表示するかどうかを示します。
- **平均線**：**ShowMeanLine** プロパティを設定して、平均線を表示します。
- **平均マーク**：**ShowMeanMarks** プロパティを設定して、平均マークを表示します。

次の図に、各学校の学生の 3 教科の点数を比較するために、これらのデータの四分位数、中央値、ひげを示します。



次のコードは、学校 A、B、C の学生が取った 3 教科の点数に関するデータを使用します。このコードは、FlexChart で箱ひげ系列を実装する方法を示します。

- **DataCreator.vb**

```
Class DataCreator
    Public Shared Function CreateSchoolScoreData() As List(Of ClassScore)
        Dim result = New List(Of ClassScore)()
        result.Add(New ClassScore() With {
            .ClassName = "英語",
            .SchoolA = 46,
            .SchoolB = 53,
            .SchoolC = 66
        })
        result.Add(New ClassScore() With {
            .ClassName = "理学",
            .SchoolA = 61,
            .SchoolB = 55,
            .SchoolC = 65
        })
        result.Add(New ClassScore() With {
            .ClassName = "英語",
            .SchoolA = 58,
            .SchoolB = 56,
            .SchoolC = 67
        })
        result.Add(New ClassScore() With {
            .ClassName = "数学",
            .SchoolA = 58,
            .SchoolB = 51,
```

```
        .SchoolC = 64
    })
    result.Add(New ClassScore() With {
        .ClassName = "英語",
        .SchoolA = 63,
        .SchoolB = 53,
        .SchoolC = 45
    })
    result.Add(New ClassScore() With {
        .ClassName = "英語",
        .SchoolA = 63,
        .SchoolB = 50,
        .SchoolC = 65
    })
    result.Add(New ClassScore() With {
        .ClassName = "数学",
        .SchoolA = 60,
        .SchoolB = 45,
        .SchoolC = 67
    })
    result.Add(New ClassScore() With {
        .ClassName = "数学",
        .SchoolA = 62,
        .SchoolB = 53,
        .SchoolC = 66
    })
    result.Add(New ClassScore() With {
        .ClassName = "理学",
        .SchoolA = 63,
        .SchoolB = 54,
        .SchoolC = 64
    })
    result.Add(New ClassScore() With {
        .ClassName = "英語",
        .SchoolA = 63,
        .SchoolB = 52,
        .SchoolC = 67
    })
    result.Add(New ClassScore() With {
        .ClassName = "理学",
        .SchoolA = 69,
        .SchoolB = 66,
        .SchoolC = 71
    })
    result.Add(New ClassScore() With {
        .ClassName = "理学",
        .SchoolA = 48,
        .SchoolB = 67,
        .SchoolC = 50
    })
    result.Add(New ClassScore() With {
        .ClassName = "理学",
        .SchoolA = 54,
        .SchoolB = 50,
        .SchoolC = 56
    })
    result.Add(New ClassScore() With {
        .ClassName = "理学",
        .SchoolA = 60,
        .SchoolB = 56,
        .SchoolC = 64
    })
    result.Add(New ClassScore() With {
        .ClassName = "数学",
        .SchoolA = 71,
        .SchoolB = 65,
```

```
            .SchoolC = 50
        })
        result.Add(New ClassScore() With {
            .ClassName = "数学",
            .SchoolA = 48,
            .SchoolB = 70,
            .SchoolC = 72
        })
        result.Add(New ClassScore() With {
            .ClassName = "数学",
            .SchoolA = 53,
            .SchoolB = 40,
            .SchoolC = 80
        })
        result.Add(New ClassScore() With {
            .ClassName = "数学",
            .SchoolA = 60,
            .SchoolB = 56,
            .SchoolC = 67
        })
        result.Add(New ClassScore() With {
            .ClassName = "数学",
            .SchoolA = 61,
            .SchoolB = 56,
            .SchoolC = 45
        })
        result.Add(New ClassScore() With {
            .ClassName = "英語",
            .SchoolA = 63,
            .SchoolB = 58,
            .SchoolC = 64
        })
        result.Add(New ClassScore() With {
            .ClassName = "理学",
            .SchoolA = 59,
            .SchoolB = 54,
            .SchoolC = 65
        })

        Return result
    End Function
End Class

Public Class ClassScore
    Public Property ClassName() As String
        Get
            Return m_ClassName
        End Get
        Set
            m_ClassName = Value
        End Set
    End Property
    Private m_ClassName As String
    Public Property SchoolA() As Double
        Get
            Return m_SchoolA
        End Get
        Set
            m_SchoolA = Value
        End Set
    End Property
    Private m_SchoolA As Double
    Public Property SchoolB() As Double
        Get
            Return m_SchoolB
        End Get
```

```
        Set
            m_SchoolB = Value
        End Set
    End Property
    Private m_SchoolB As Double
    Public Property SchoolC() As Double
        Get
            Return m_SchoolC
        End Get
        Set
            m_SchoolC = Value
        End Set
    End Property
    Private m_SchoolC As Double
End Class
```

- **DataCreator.cs**

```
class DataCreator
{
    public static List<ClassScore> CreateSchoolScoreData()
    {
        var result = new List<ClassScore>();
        result.Add(new ClassScore() {ClassName="英語",SchoolA= 46,SchoolB= 53,SchoolC =66});
        result.Add(new ClassScore() {ClassName="理学",SchoolA= 61,SchoolB= 55,SchoolC =65});
        result.Add(new ClassScore() {ClassName="英語",SchoolA= 58,SchoolB= 56,SchoolC =67});
        result.Add(new ClassScore() {ClassName="数学",SchoolA= 58,SchoolB= 51,SchoolC =64});
        result.Add(new ClassScore() {ClassName="英語",SchoolA= 63,SchoolB= 53,SchoolC =45});
        result.Add(new ClassScore() {ClassName="英語",SchoolA= 63,SchoolB= 50,SchoolC =65});
        result.Add(new ClassScore() {ClassName="数学",SchoolA= 60,SchoolB= 45,SchoolC =67});
        result.Add(new ClassScore() {ClassName="数学",SchoolA= 62,SchoolB= 53,SchoolC =66});
        result.Add(new ClassScore() {ClassName="理学",SchoolA= 63,SchoolB= 54,SchoolC =64});
        result.Add(new ClassScore() {ClassName="英語",SchoolA= 63,SchoolB= 52,SchoolC =67});
        result.Add(new ClassScore() {ClassName="理学",SchoolA= 69,SchoolB= 66,SchoolC =71});
        result.Add(new ClassScore() {ClassName="理学",SchoolA= 48,SchoolB= 67,SchoolC =50});
        result.Add(new ClassScore() {ClassName="理学",SchoolA= 54,SchoolB= 50,SchoolC =56});
        result.Add(new ClassScore() {ClassName="理学",SchoolA= 60,SchoolB= 56,SchoolC =64});
        result.Add(new ClassScore() {ClassName="数学",SchoolA= 71,SchoolB= 65,SchoolC =50});
        result.Add(new ClassScore() {ClassName="数学",SchoolA= 48,SchoolB= 70,SchoolC =72});
        result.Add(new ClassScore() {ClassName="数学",SchoolA= 53,SchoolB= 40,SchoolC =80});
        result.Add(new ClassScore() {ClassName="数学",SchoolA= 60,SchoolB= 56,SchoolC =67});
        result.Add(new ClassScore() {ClassName="数学",SchoolA= 61,SchoolB= 56,SchoolC =45});
        result.Add(new ClassScore() {ClassName="英語",SchoolA= 63,SchoolB= 58,SchoolC =64});
        result.Add(new ClassScore() {ClassName="理学",SchoolA= 59,SchoolB= 54,SchoolC =65});

        return result;
    }

}
public class ClassScore
{
    public string ClassName { get; set; }
    public double SchoolA { get; set; }
    public double SchoolB { get; set; }
    public double SchoolC { get; set; }
}
```

- **Visual Basic**

```
Partial Public Class MainPage
    Private _data As List(Of ClassScore) = Nothing
    Public Sub New()
        InitializeComponent()

        ' 平均線を表示します。
        boxWhiskerA.ShowMeanLine = True
```

```
        boxWhiskerB.ShowMeanLine = True
        boxWhiskerC.ShowMeanLine = True

        ' 内側ポイントを表示します。
        boxWhiskerA.ShowInnerPoints = True
        boxWhiskerB.ShowInnerPoints = True
        boxWhiskerC.ShowInnerPoints = True

        ' 異常値を表示します。
        boxWhiskerA.ShowOutliers = True
        boxWhiskerB.ShowOutliers = True
        boxWhiskerC.ShowOutliers = True

        ' 平均マークを表示します。
        boxWhiskerA.ShowMeanMarks = True
        boxWhiskerB.ShowMeanMarks = True
        boxWhiskerC.ShowMeanMarks = True

        ' 四分位計算を指定します。
        boxWhiskerA.QuartileCalculation = QuartileCalculation.InclusiveMedian
        boxWhiskerB.QuartileCalculation = QuartileCalculation.InclusiveMedian
        boxWhiskerC.QuartileCalculation = QuartileCalculation.InclusiveMedian
    End Sub

End Class

Public ReadOnly Property Data() As List(Of ClassScore)
    Get
        If _data Is Nothing Then
            _data = DataCreator.CreateSchoolScoreData()
        End If

        Return _data
    End Get
End Property
```

- **C#**

```
public partial class BoxWhisker : Page
{
    private List<ClassScore> _data = null;
    public BoxWhisker()
    {
        InitializeComponent();

        // 平均線を表示します。
        boxWhiskerA.ShowMeanLine = true;
        boxWhiskerB.ShowMeanLine = true;
        boxWhiskerC.ShowMeanLine = true;

        // 内側ポイントを表示します。
        boxWhiskerA.ShowInnerPoints = true;
        boxWhiskerB.ShowInnerPoints = true;
        boxWhiskerC.ShowInnerPoints = true;

        // 異常値を表示します。
        boxWhiskerA.ShowOutliers = true;
        boxWhiskerB.ShowOutliers = true;
        boxWhiskerC.ShowOutliers = true;

        // 平均マークを表示します。
        boxWhiskerA.ShowMeanMarks = true;
        boxWhiskerB.ShowMeanMarks = true;
        boxWhiskerC.ShowMeanMarks = true;

        // 四分位計算を指定します。
        boxWhiskerA.QuartileCalculation = QuartileCalculation.InclusiveMedian;
```

```
            boxWhiskerB.QuartileCalculation = QuartileCalculation.InclusiveMedian;
            boxWhiskerC.QuartileCalculation = QuartileCalculation.InclusiveMedian;
        }
        public List<ClassScore> Data
        {
            get
            {
                if (_data == null)
                {
                    _data = DataCreator.CreateSchoolScoreData();
                }

                return _data;
            }
        }
}
```

誤差範囲

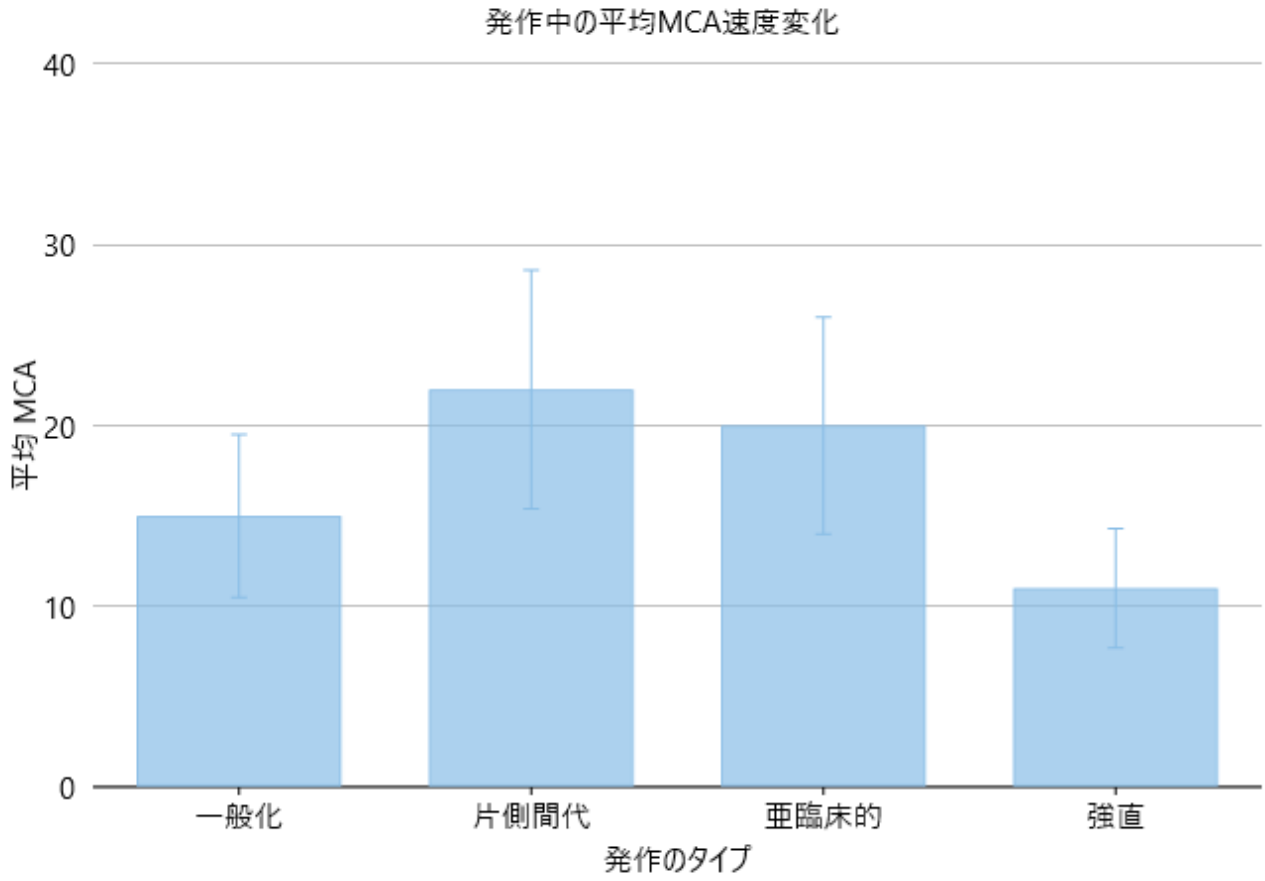
誤差範囲系列を使用すると、データのばらつきや値の不確実性を示すことができます。誤差範囲を使用して、変数データの標準偏差と誤差の範囲を表示できます。一般に、科学研究や科学実験の結果では、誤差範囲チャートを使用して、元の値からのデータのばらつきを表現します。

FlexChart では、面、棒、折れ線、折れ線シンボル、散布図、スプライン、スプライン面、スプラインシンボルなどのさまざまなチャートタイプで誤差範囲系列を使用できます。

FlexChart の誤差範囲系列には、次の機能があります。

- **誤差量**：固定値、パーセンテージ、標準誤差、標準偏差などのさまざまな方法を使用して、すべてのデータポイントに誤差範囲を設定します。さらに、必要に応じて、正確な誤差量を示すカスタム値を設定できます。これらの方法で誤差範囲を表示するには、**ErrorAmount** 列挙の **ErrorAmount** プロパティを設定します。
- **方向**：**ErrorBarDirection** 列挙の **Direction** プロパティを設定して、誤差範囲をプラス、マイナス、さらには両方向で表示します。
- **終点スタイル**：**ErrorBarEndStyle** 列挙の **EndStyle** プロパティを設定して、誤差範囲をキャップ付きまたはキャップなしで表示します。
- **範囲スタイル**：**ErrorBarStyle** プロパティを使用して、誤差範囲の外観をカスタマイズします。

次の図に、子供に観察されるさまざまなタイプの発作に対して、平均 MCA（Middle Cerebral Artery：中大脳動脈）速度データのプラスおよびマイナスの誤差量を示します。



次のコードは、子供のさまざまなタイプの発作における MCA 速度の平均パーセンテージ値を使用します。このコードは、FlexChart で誤差範囲系列を実装する方法を示します。

- **DataCreator.vb**

```
Class DataCreator
    Public Shared Function CreateData() As List(Of DataItem)
        Dim data = New List(Of DataItem)()
        data.Add(New DataItem("一般化", 15))
        data.Add(New DataItem("片側間代", 22))
        data.Add(New DataItem("亜臨床的", 20))
        data.Add(New DataItem("強直", 11))
        Return data
    End Function
End Class
Public Class DataItem
    Public Sub New(seizuretype__1 As String, meanmca__2 As Integer)
        SeizureType = seizuretype__1
        MeanMCA = meanmca__2
    End Sub

    Public Property SeizureType() As String
        Get
            Return m_SeizureType
        End Get
        Set
            m_SeizureType = Value
        End Set
    End Property
    Private m_SeizureType As String
```

```
    Public Property MeanMCA() As Integer
        Get
            Return m_MeanMCA
        End Get
        Set
            m_MeanMCA = Value
        End Set
    End Property
    Private m_MeanMCA As Integer
End Class
```

- **DataCreator.cs**

```
class DataCreator
{
    public static List<DataItem> CreateData()
    {
        var data = new List<DataItem>();
        data.Add(new DataItem("一般化", 15));
        data.Add(new DataItem("片側間代", 22));
        data.Add(new DataItem("亜臨床的", 20));
        data.Add(new DataItem("強直", 11));
        return data;
    }
}
public class DataItem
{
    public DataItem(string seizuretype, int meanmca)
    {
        SeizureType = seizuretype;
        MeanMCA = meanmca;
    }

    public string SeizureType { get; set; }
    public int MeanMCA { get; set; }
}
```

- **Visual Basic**

```
Partial Public Class MainPage
    Inherits Page
    Private _data As List(Of DataItem)
    Public Sub New()
        InitializeComponent()

        ' データ系列コレクションをクリアします。
        flexChart.Series.Clear()

        ' 誤差範囲の系列を作成します。
        Dim errorBar As New C1.Xaml.Chart.ErrorBar()

        ' データ系列コレクションに系列を追加します。
        flexChart.Series.Add(errorBar)

        ' X軸とY軸をバインドします。
        flexChart.BindingX = "SeizureType"
        errorBar.Binding = "MeanMCA"

        ' 系列の誤差量を指定します。
        errorBar.ErrorAmount = C1.Chart.ErrorAmount.Percentage

        ' 誤差の方向を指定します。
```

```
        errorBar.Direction = C1.Chart.ErrorBarDirection.Both

        ' 系列の誤差値を指定します。
        errorBar.ErrorValue = 0.3

        ' 誤差範囲の系列のスタイルします。
        errorBar.EndStyle = C1.Chart.ErrorBarEndStyle.Cap
    End Sub
    Public ReadOnly Property Data() As List(Of DataItem)
        Get
            If _data Is Nothing Then
                _data = DataCreator.CreateData()
            End If

            Return _data
        End Get
    End Property
End Class
```

- **C#**

```
public sealed partial class MainPage : Page
{
    private List<DataItem> _data;
    public MainPage()
    {
        InitializeComponent();

        // データ系列コレクションをクリアします。
        flexChart.Series.Clear();

        // 誤差範囲の系列を作成します。
        C1.Xaml.Chart.ErrorBar errorBar = new C1.Xaml.Chart.ErrorBar();

        // データ系列コレクションに系列を追加します。
        flexChart.Series.Add(errorBar);

        // X軸とY軸をバインドします。
        flexChart.BindingX = "SeizureType";
        errorBar.Binding = "MeanMCA";

        // 系列の誤差量を指定します。
        errorBar.ErrorAmount = C1.Chart.ErrorAmount.Percentage;

        //  誤差の方向を指定します。
        errorBar.Direction = C1.Chart.ErrorBarDirection.Both;

        // 系列の誤差値を指定します。
        errorBar.ErrorValue = .3;

        // 誤差範囲の系列のスタイルします。
        errorBar.EndStyle = C1.Chart.ErrorBarEndStyle.Cap;

    }
    public List<DataItem> Data
    {
        get
        {
            if (_data == null)
            {
                _data = DataCreator.CreateData();
            }
```

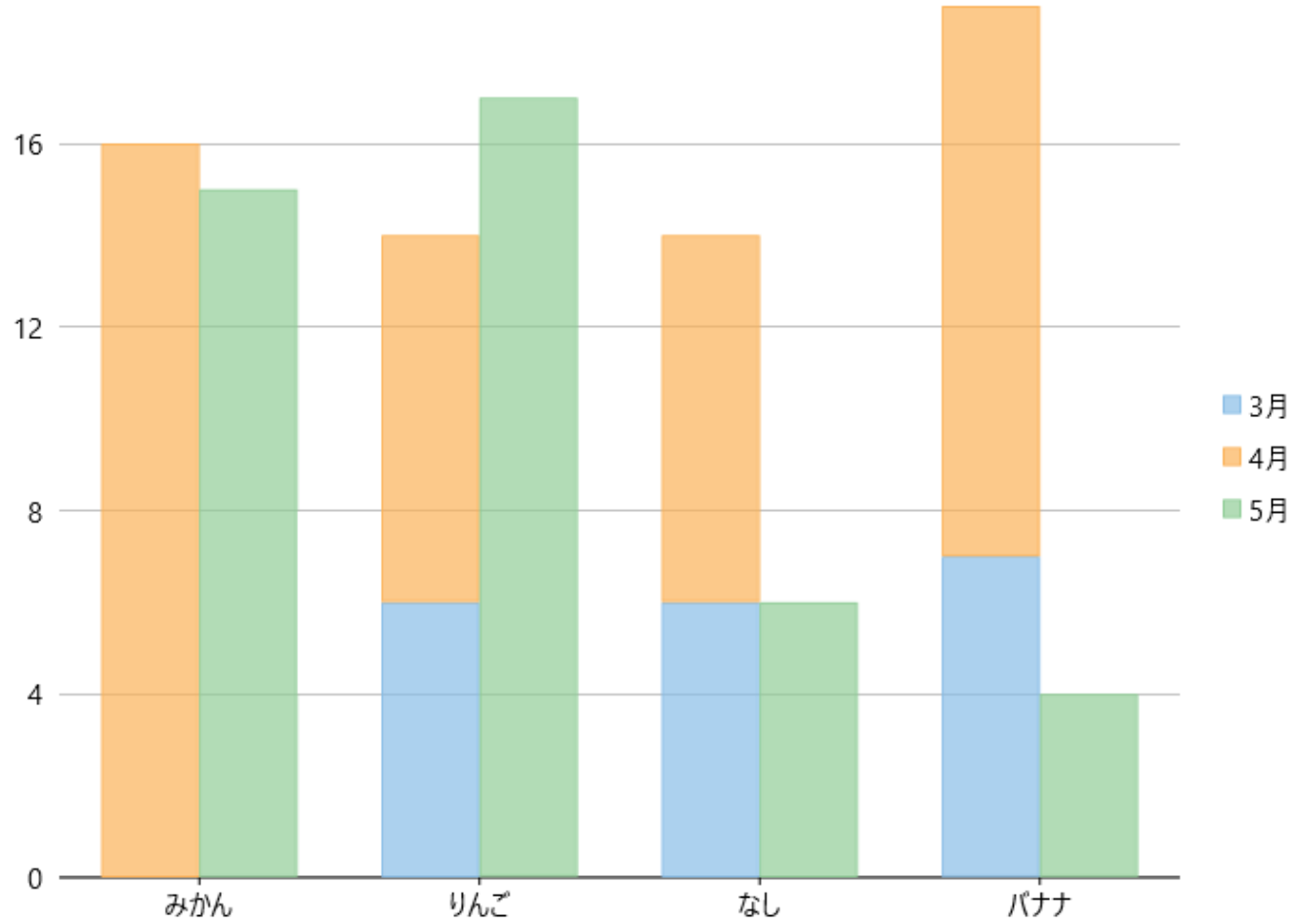
```
            return _data;
        }
    }

}
```

積層グループ

FlexChart は、縦棒グラフと横棒グラフで、データ項目の積層とグループ化をサポートします。積層は、データを上に(縦棒グラフ)または横に(横棒グラフ)積み重ねます。一方、グループ化は、縦棒グラフまたは横棒グラフで積み重ねられたデータを集合化します。

積層グループを使用すると、グループ内のカテゴリ間で項目を比較できます。さらに、各グループの項目間の相対的な差を視覚化できます。

次の図に、FlexChart の積層グループを示します。



特定の積層グループで特定の系列を積み重ねるには、系列の **StackingGroup** プロパティで積層グループのインデックス値を設定します。FlexChart の積層グループは、FlexChart の **Stacking** プロパティが **Stacked** または **Stacked100pc** に設定されている場合に実装できます。これにより、チャートのデータ値を積み重ねる方法を指定します。

次のコードは、クイックスタートで作成したサンプルを使用し、FlexChart で積層グループで実装する方法を示します。

XAML

- **Tab Caption**

```
<Chart:C1FlexChart x:Name="flexChart" Stacking="Stacked"
  ItemsSource="{Binding DataContext.Data}" BindingX="Fruit">
    <Chart:Series SeriesName="3月" Binding="March" StackingGroup="0"/>
    <Chart:Series SeriesName="4月" Binding="April" StackingGroup="0"/>
    <Chart:Series SeriesName="5月" Binding="May" StackingGroup="1"/>
</Chart:C1FlexChart>
```

FlexChart のデータラベル

データラベルは、データポイントに関連付けられたラベルで、データポイントに関する追加情報を提供します。言い換えると、データラベルは、系列のデータポイント上に表示される説明的なテキストまたは値と定義することができます。データラベルは、主に、重要なデータポイントを強調表示することで、チャートを読みやすくし、データをすばやく識別できるようにするために使用されます。

FlexChart でサポートされているデータラベルは、高度にカスタマイズ可能で、チャートデータを簡単に強調表示できます。そのため、エンドユーザーがチャートデータを効率よく識別および解釈するために役立ちます。FlexChartでデータラベルを使用するには、**DataLabel**プロパティを使用します。デフォルトでは、FlexChartにデータラベルは表示されません。しかし、**DataLabel**クラスと **DataLabelBase** クラスのさまざまなプロパティを使用して、データラベルを表示できるだけでなく、必要に応じてカスタマイズすることもできます。

以下のセクションでは、データポイントにデータラベルを追加する方法、およびデータラベルの外観やデータラベルに表示されるデータなど、データラベルの詳細を制御する方法について説明します。

- データラベルの追加と配置
- データラベルの書式設定
- Manage Overlapped Data Labels

データラベルの追加と配置

チャートのデータポイントにデータラベルを追加すると、データラベルに個々のデータポイントの詳細が表示されるため、チャートデータを理解しやすくなります。データラベルは、関連のあるデータや重要なデータをすばやく強調表示します。

FlexChart の使用時に、簡単な方法でデータポイントにデータラベルを追加できます。データラベルに表示するエントリの種類に応じて、**Content** プロパティを設定するだけです。また、チャートにデータラベルを表示するには、**Position** プロパティを使用してデータラベルの位置を設定する必要があります。

次の表に、データラベルのコンテンツのカスタマイズ化に適用される定義済みパラメータを示します。

パラメータ	説明
x	データポイントのX値を表示します。
y	データポイントのY値を表示します。
value	データポイントのY値を表示します。
name	データポイントのX値を表示します。
seriesName	系列の名前を表示します。
pointIndex	データポイントのインデックスを表示します。
P	Sunburstの親セグメントに対する割合を示します。
p	Sunburstのチャート全体に対する割合を示します。

次のコードスニペットを参照してください。

XAML

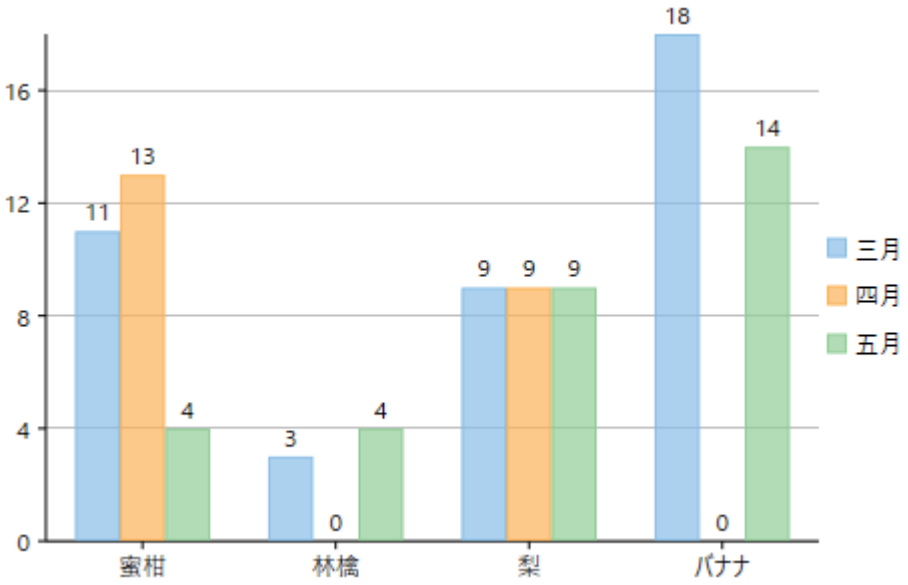
- **タブキャプション**

```
<Chart:C1FlexChart.DataLabel>
    <Chart:DataLabel Content="{}{y}" Position="Top"></Chart:DataLabel>
</Chart:C1FlexChart.DataLabel>
```

コード

C# copyCode

```
flexChart.DataLabel.Content = "{y}";
flexChart.DataLabel.Position = C1.Chart.LabelPosition.Top;
```



チャートタイプに応じて、さまざまな配置オプションを選択して、データラベルをチャートに完璧に配置できます。**Position** プロパティは、**LabelPosition** 列挙に含まれる次の値を受け取ります。

プロパティ	説明
Top	ラベルをデータポイントの上設定します。
Bottom	ラベルをデータポイントの下設定します。
Left	ラベルをデータポイントの左に設定します。
Right	ラベルをデータポイントの右に設定します。
Center	ラベルをデータポイントの中央に設定します。
None	ラベルを非表示にします。

Contentプロパティを使用することにより、データラベルの内容をカスタマイズして、系列名、インデックス値、またはデータポイントの名前をさらに含めることができます。

次のコードは、データラベルに系列名およびデータポイント値を含めるようにContentプロパティを設定する方法を示しています。

XAML

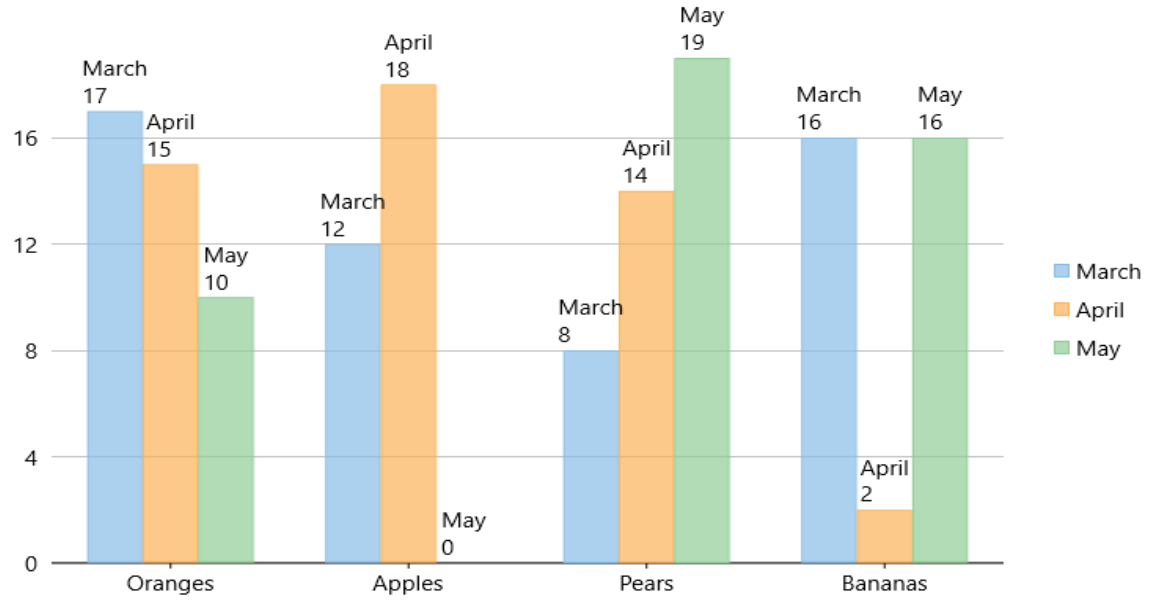
- **Tab Caption**

```
<Chart:C1FlexChart.DataLabel>
    <Chart:DataLabel Content="{}{seriesName}&#x0a;{y}"></Chart:DataLabel>
</Chart:C1FlexChart.DataLabel>
```

Code

C# copyCode

```
// Content プロパティを設定します
flexChart.DataLabel.Content = "{seriesName}\n{value}";
```



データラベルの書式設定

FlexChart では、データラベルを好みの方法で書式設定できるさまざまなオプションが提供されています。データラベルの境界線の設定およびスタイル指定、対応するデータポイントとの接続、データラベルの表示方法のカスタマイズが可能です。

データラベルの境界線の設定とスタイル指定

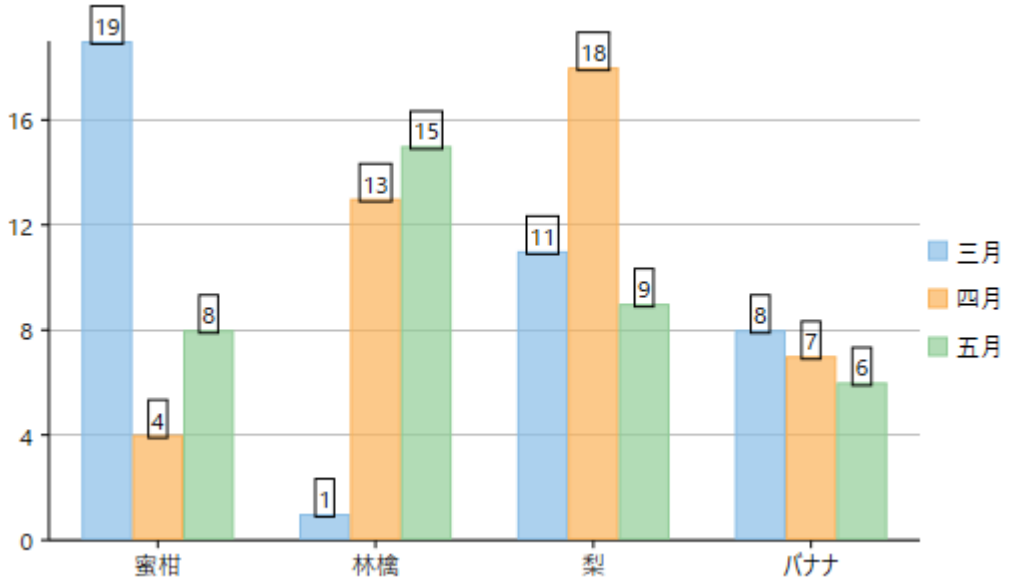
境界線はデータラベルの見栄えをさらによくし、データラベルをさらに強調します。これは、本当に重要なデータをシームレスに強調するために便利です。エンドユーザーは、重要な内容に集中することができます。

FlexChart では、境界線は、**Border**、**BorderStyle** などのさまざまなプロパティを使用して有効化およびカスタマイズできます。

次のコードスニペットは、境界線の設定およびカスタマイズ方法を示しています。

- **C#**

```
flexChart.DataLabel.Border = true;
```



データポイントへのデータラベルの接続

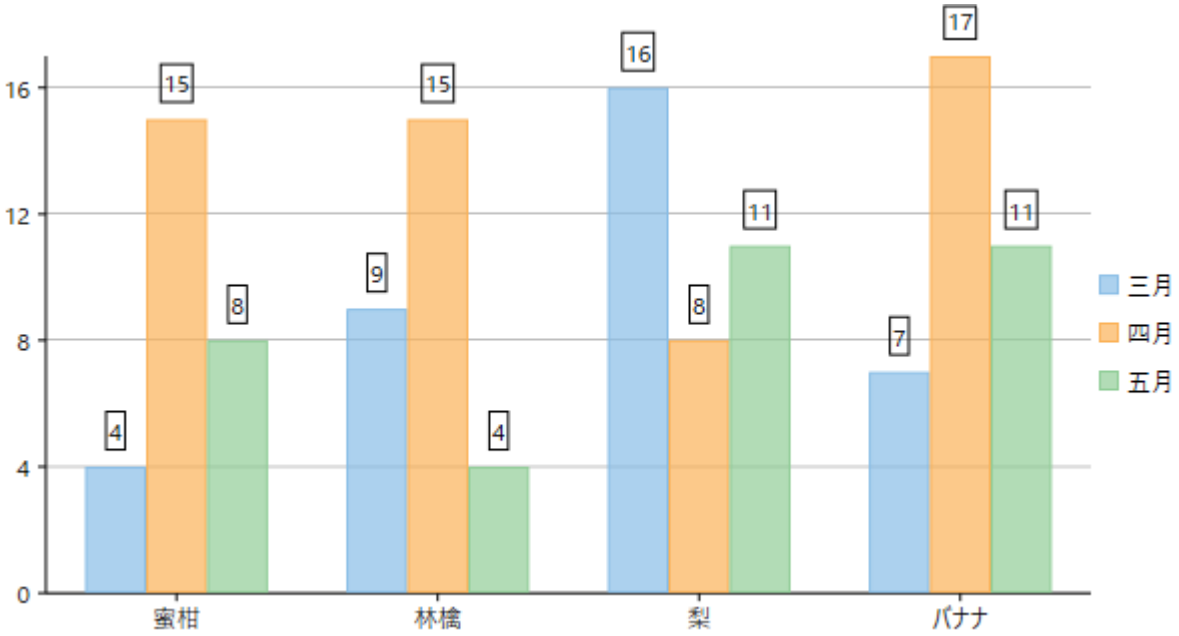
データラベルを対応するデータポイントから離して配置した場合は、引き出し線を使用してそれらを接続することができます。引き出し線は、データラベルをデータポイントに接続する線です。引き出し線は、特に、データラベルと関連するデータポイントとのつながりを視覚的に示す必要がある場合に使用されます。

FlexChart では、データラベルを追加しても、デフォルトでは引き出し線は表示されません。しかし、引き出し線を有効にし、適切な長さを設定して、視覚的にわかりやすいデータラベルのつながりをチャート内に表示できます。引き出し線を有効にするには、**ConnectingLine** プロパティを使用します。また、データラベルとデータポイントの距離を設定するには、**Offset** プロパティを使用します。

次のコードスニペットでは、この 2 つのプロパティを設定しています。

- **C#**

```
flexChart.DataLabel.ConnectingLine = true;
flexChart.DataLabel.Offset = 10;
```



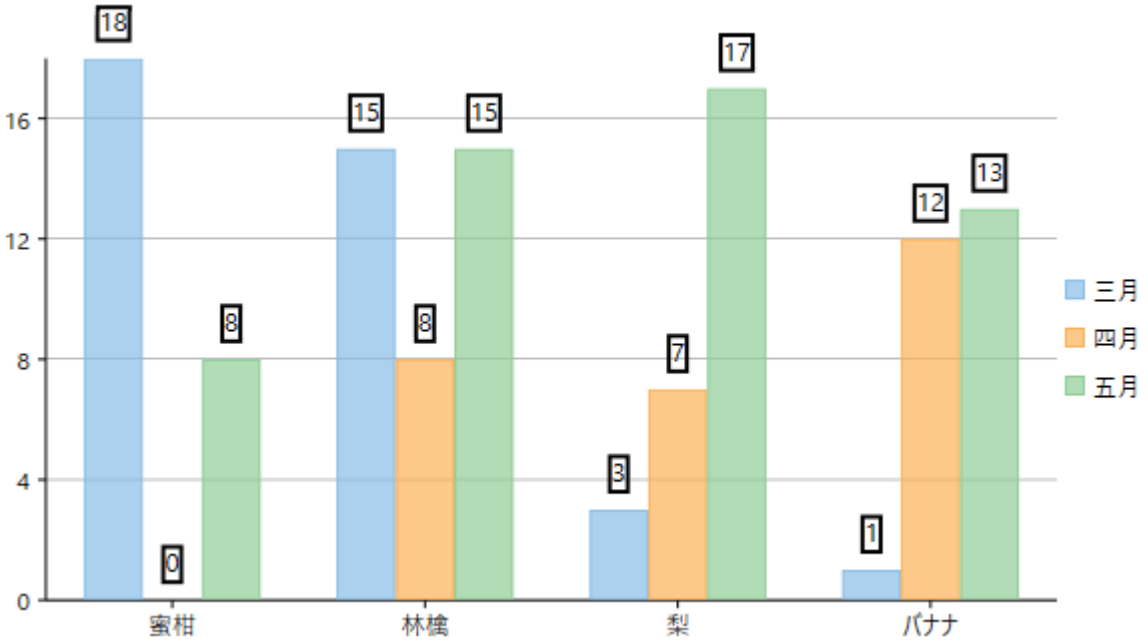
データラベルの外観の変更

チャートでデータラベルの表示方法を変更することで、強力で効果的なデータ視覚化が可能になります。FlexChart にはさまざまなスタイルオプションが含まれており、これを使用して、見栄えのよい明確なデータラベルを作成できます。データラベルの外観を変更するには、**Style** プロパティを使用します。

次のコードスニペットを参照してください。

- ***C#***

```
flexChart.DataLabel.Style.FontSize = 13;
flexChart.DataLabel.Style.StrokeThickness = 2;
```



重なったデータラベルの管理

A common issue pertaining to charts is the overlapping of data labels that represent data points. In most cases, overlapping occurs due to long text in data labels or large numbers of data points.

In case of overlapped data labels in FlexChart, it provides the following ways to manage the overlapping.

- **Auto Arrangement of Data Labels**
- **Hide Overlapped Labels**
- **Control Appearance of Overlapped Labels**
- **Rotate Data Labels**
- **Trim or Wrap Data Label**

Auto Arrangement of Data Labels

The easiest way to handle overlapping of data labels is to set the FlexChart to position the data labels automatically. For automatic positioning of data labels, you can set the **Position** property to Auto. Moreover, you can also set the **MaxAutoLabels** property to set the maximum number of labels that can be positioned automatically.

When the Position property is set to Auto, the number of created data labels is limited by MaxAutoLabels property which is 100 by default. You can increase the value of MaxAutoLabels property if necessary, but it may slow down the chart rendering since the label positioning algorithm becomes expensive in terms of performance when number of labels is large.

This approach may not provide an optimal layout when working with large data set and when there is not enough space for all data labels. In this case, it's recommended to reduce the number of data labels. For example, create a series with limited number of data points that should have labels, that is, chose to hide the labels at the individual series level.

C#

```
// PositionおよびMaxAutoLabelsプロパティを設定します
flexChart.DataLabel.Position = LabelPosition.Auto;
flexChart.DataLabel.MaxAutoLabels = 150;
```

Hide Overlapped Labels

In case of overlapped data labels in FlexChart, you can use the **Overlapping** property provided by DataLabel class. This approach is helpful when the developer wants to completely hide or show the overlapped data labels.

C#

```
// Overlappingプロパティを設定します
flexChart.DataLabel.Overlapping = LabelOverlapping.Hide;
```

The Overlapping property accepts the following values in the LabelOverlapping enumeration.

Enumeration	Description
Hide	Hide overlapped data labels.
Show	Show overlapped data labels.

Control Appearance of Overlapped Labels

Furthermore, you can use the **OverlappingOptions** property to specify additional label overlapping options that will help the user to effectively manage overlapping of data labels.

C#

```
// OverlappingOptionsプロパティを設定します
flexChart.DataLabel.OverlappingOptions =
LabelOverlappingOptions.OutsidePlotArea;
```

The OverlappingOptions property accepts the following values in the **LabelOverlappingOptions** enumeration.

Enumeration	Description
None	No overlapping is allowed.
OutsidePlotArea	Allow labels outside plot area.
OverlapDataPoints	Allow overlapping with data points.

Rotate Data Labels

Another option to manage overlapping of data labels in FlexChart is to use the **Angle** property. The Angle property enables the user to set a specific rotation angle for data labels.

C#

```
// Angleプロパティを設定します
flexChart.DataLabel.Angle = 50;
```

Trim or Wrap Data Label

To manage the content displayed in the data labels, in case of overlapping, you can either trim the data labels or wrap the data labels using **ContentOptions** property. Managing of data labels using the ContenOptions property is dependent on MaxWidth and MaxLines property.

The **MaxWidth** property allows you to set the maximum width of a data label. In case the width of data label text exceeds the specified width, then you can either trim the data labels or wrap the data labels using the ContentOptions property.

The **MaxLines** property allows you to set the maximum number of lines in data label. This property helps you to limit the wrapped text to grow vertically. In case the wrapped text does not fit within the specified MaxWidth and MaxLines property values, then the last line gets trimmed with an ellipses(...).

C#

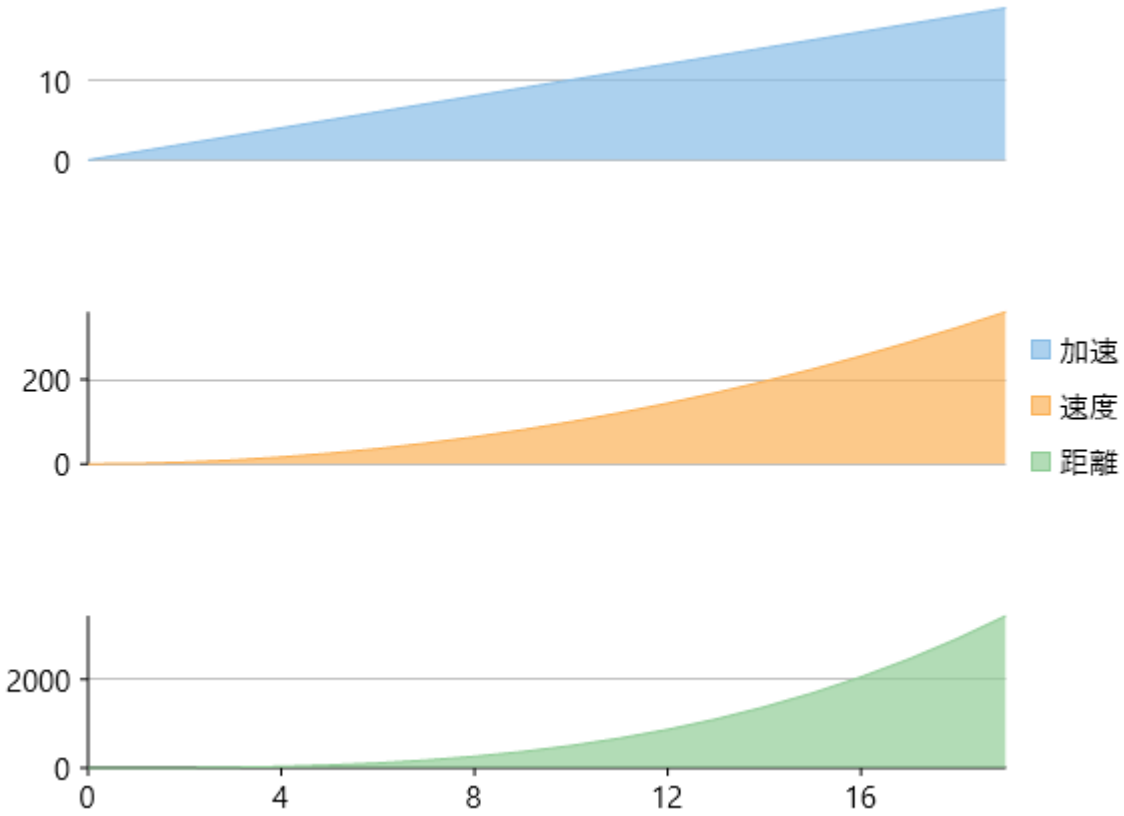
```
// MaxWidthプロパティを設定します
flexChart.DataLabel.MaxWidth = 20;
// ContentOptionsプロパティを設定します
flexChart.DataLabel.ContentOptions = ContentOptions.Trim;
```

複数のプロット領域

複数のプロット領域を使用すると、一方の軸は固定したまま、もう一方の軸方向には各系列を個別のプロット領域に表示して、データの可視性を向上させることができます。

FlexChart では、1 つのチャート領域内で、系列ごとに異なるプロット領域を作成できます。FlexChart で複数のプロット領域を作成し、それらを **C1FlexChart.PlotAreas** コレクションに追加します。さらに、プロット領域の行インデックス、列インデックス、高さ、幅をカスタマイズできます。

次の図に、複数のプロット領域を含む FlexChart を示します。各プロット領域には 1 つの系列のデータが表示されます。



次のコードでは、車の加速度、速度、距離、時間の 4 つのメトリックに関するデータを使用します。このコードは、FlexChart で複数のプロット領域を実装する方法を示します。

- **Visual Basic**

```
' 複数のプロット領域を作成して追加します。
flexChart.PlotAreas.Add(New PlotArea() With {
    .PlotAreaName = "plot1",
    .Row = 0
})
flexChart.PlotAreas.Add(New PlotArea() With {
    .PlotAreaName = "plot2",
    .Row = 2
})
flexChart.PlotAreas.Add(New PlotArea() With {
    .PlotAreaName = "plot3",
    .Row = 4
})

' チャートタイプを指定します。
flexChart.ChartType = C1.Chart.ChartType.Area

' 系列を作成、追加とバインドします。
```

```
flexChart.Series.Add(New Series() With {
    .SeriesName = "加速",
    .Binding = "Acceleration"
})

flexChart.Series.Add(New Series() With {
    .SeriesName = "速度",
    .Binding = "Velocity",
    .AxisY = New Axis() With {
        .Position = C1.Chart.Position.Left,
        .MajorGrid = True,
        .PlotAreaName = "plot2"
    }
})

flexChart.Series.Add(New Series() With {
    .SeriesName = "距離",
    .Binding = "Distance",
    .AxisY = New Axis() With {
        .Position = C1.Chart.Position.Left,
        .MajorGrid = True,
        .PlotAreaName = "plot3"
    }
})
```

- **C#**

```
// 複数のプロット領域を作成して追加します。
flexChart.PlotAreas.Add(new PlotArea { PlotAreaName = "plot1", Row = 0 });
flexChart.PlotAreas.Add(new PlotArea { PlotAreaName = "plot2", Row = 2 });
flexChart.PlotAreas.Add(new PlotArea { PlotAreaName = "plot3", Row = 4 });

// チャートタイプを指定します。
flexChart.ChartType = C1.Chart.ChartType.Area;

// 系列を作成、追加とバインドします。
flexChart.Series.Add(new Series()
{
    SeriesName = "加速",
    Binding = "Acceleration",

});

flexChart.Series.Add(new Series()
{
    SeriesName = "速度",
    Binding = "Velocity",
    AxisY = new Axis()
    {
        Position = C1.Chart.Position.Left,
        MajorGrid = true,
        PlotAreaName = "plot2"
    },

});

flexChart.Series.Add(new Series()
{
    SeriesName = "距離",
    Binding = "Distance",
    AxisY = new Axis()
    {
        Position = C1.Chart.Position.Left,
```

```
        MajorGrid = true,
        PlotAreaName = "plot3"
    }
});
```

近似曲線

Trend lines are an important tool for analyzing data. Trend line indicates the general rate of increase or decrease of Y data over X data in a chart. A common scenario is measuring the rate change of sales price over time. FlexChart control supports trend lines through a built-in **TrendLine** class for ease of use during implementation. Trend lines are most commonly used in Line, Column, Bar, or Scatter charts.

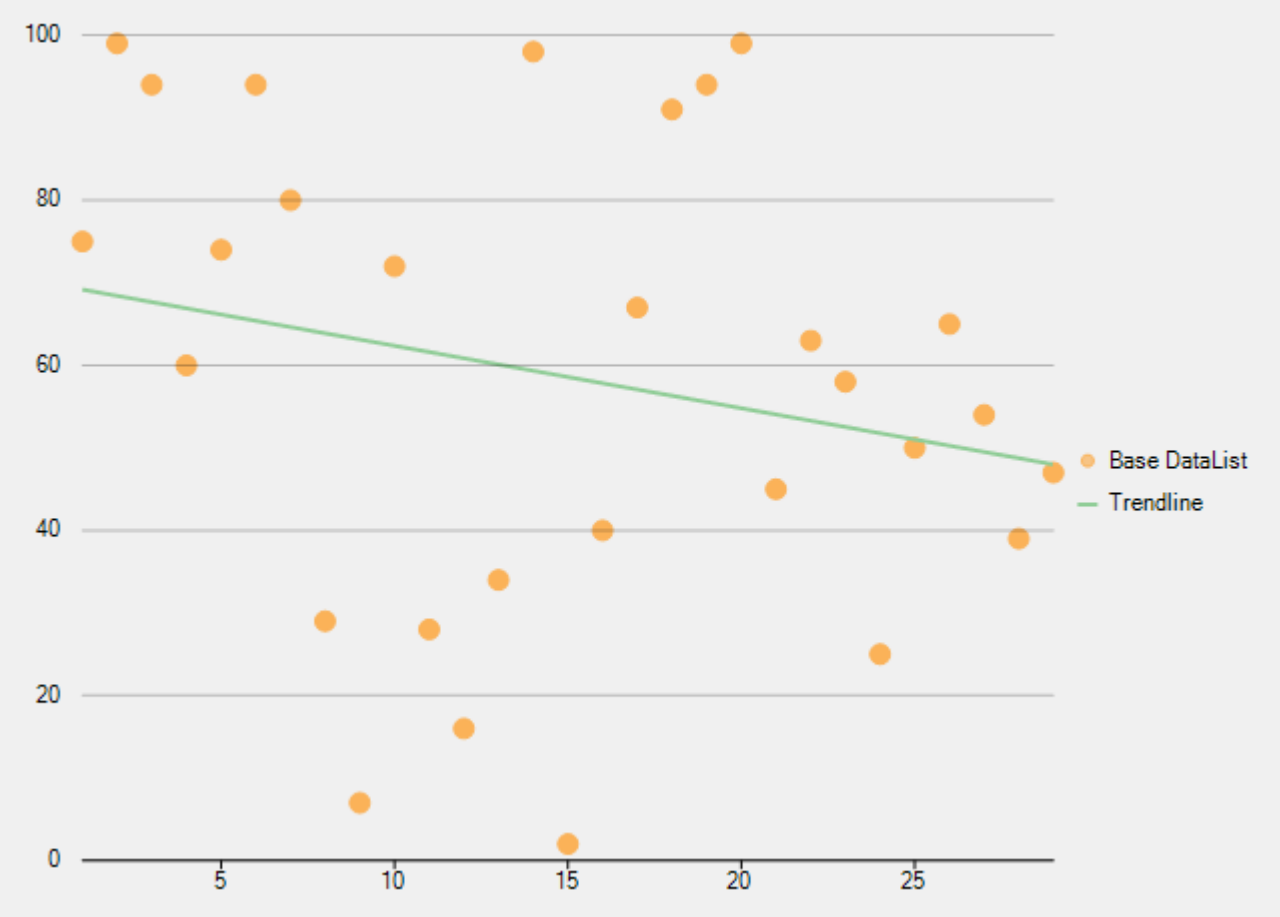
FlexChart supports the following regression and non-regression trend lines.

TrendLine.FitType	Description
Linear	A linear trend line is the straight line that most closely approximates the data in the chart. The data is linear, if the data pattern resembles a line. Equation - Y(x) = C0 + C1*x
Polynomial	Polynomial trend lines are curved lines that are used with fluctuating data. They are useful for analyzing gains or losses over a large data set. When using a polynomial trend line, it is important to also set the Order of the line, which can be determined by the number of fluctuations in the data. Equation - Y(x) = C0 + C1*x + C2*x2 + : + Cn-1*xn-1
Logarithmic	Logarithmic trend line is a best-fit curved line that is most useful when the rate of change in the data increases or decreases quickly and then levels out. A logarithmic trend line can use negative and/or positive values. Equation - Y(x) = C0 * ln(C1*x)
Power	Power trend line is a curved line that is best used with data sets that compare measurements that increase at a specific rate — for example, the acceleration of a race car at one-second intervals. You cannot create a power trend line if your data contains zero or negative values. Equation - Y(x) = C0 * pow(x, C1)
Exponent	Exponential trend line is a curved line that is most useful when data values rise or fall at increasingly higher rates. You cannot create an exponential trend line if your data contains zero or negative values. Equation - Y(x) = C0 * exp(C1*x)
Fourier	Fourier trend line identifies patterns or cycles in a series data set. It removes the effects of trends or other complicating factors from the data set, thereby providing a good estimate of the direction that the data under analysis will take in the future. Equation - Y(x) = C0 + C1 * cos(x) + C2 * sin(x) + C3 * cos(2*x) + C4 * sin(2*x) + ...
MinX	The minimum X-value on the chart.
MinY	The minimum Y-value on the chart.
MaxX	The maximum X-value on the chart.
MaxY	The maximum Y-value on the chart.
AverageX	The average X-value on the chart.

AverageY	The average Y-value on the chart.

To implement trend line in FlexChart, use the **TrendLine** class that inherits the **Series** class. To begin with, create an instance of TrendLine class, and then use the TrendLine object to specify its properties. Once, the associated properties are set, add the trend line to the FlexChart using Series.Add method.

The following image shows how FlexChart appears after adding a linear trend line.



XAML

```
<Chart:C1FlexChart x:Name="flexChart" RenderMode="Direct2D"
ChartType="LineSymbols" ItemsSource="{Binding Data}" ToolTipContent="{}{y}"
Grid.Row="1">
            <Chart:C1FlexChart.AxisY>
                <Chart:Axis Min="0" Max="100" AxisLine="False" MajorGrid="True"
MajorTickMarks="None" />
            </Chart:C1FlexChart.AxisY>
            <Chart:Series SeriesName="Base Data" BindingX="X" Binding="Y"/>
            <Chart:TrendLine SeriesName="Trend Line" x:Name="trendLine"
Binding="Y" BindingX="X" Order="4"/>
        </Chart:C1FlexChart>
```

Code

HTML

```
public partial class Trendline
    {
        ObservableCollection<DataItem> dataList = new
ObservableCollection<DataItem>();
        public Form1()
        {
            InitializeComponent();
            var rnt = new Random();
            for (int i = 1; i < 30; i++)
            {
                dataList.Add(new DataItem() { X = i, Y = rnt.Next(100) });
            }
        }
        public class DataItem
        {
            int _y;
            public int X { get; set; }

            public int Y
            {
                get { return _y; }
                set
                {
                    if (value == _y) return;
                    _y = value;
                }
            }}
        }
    }
```

エクスポート

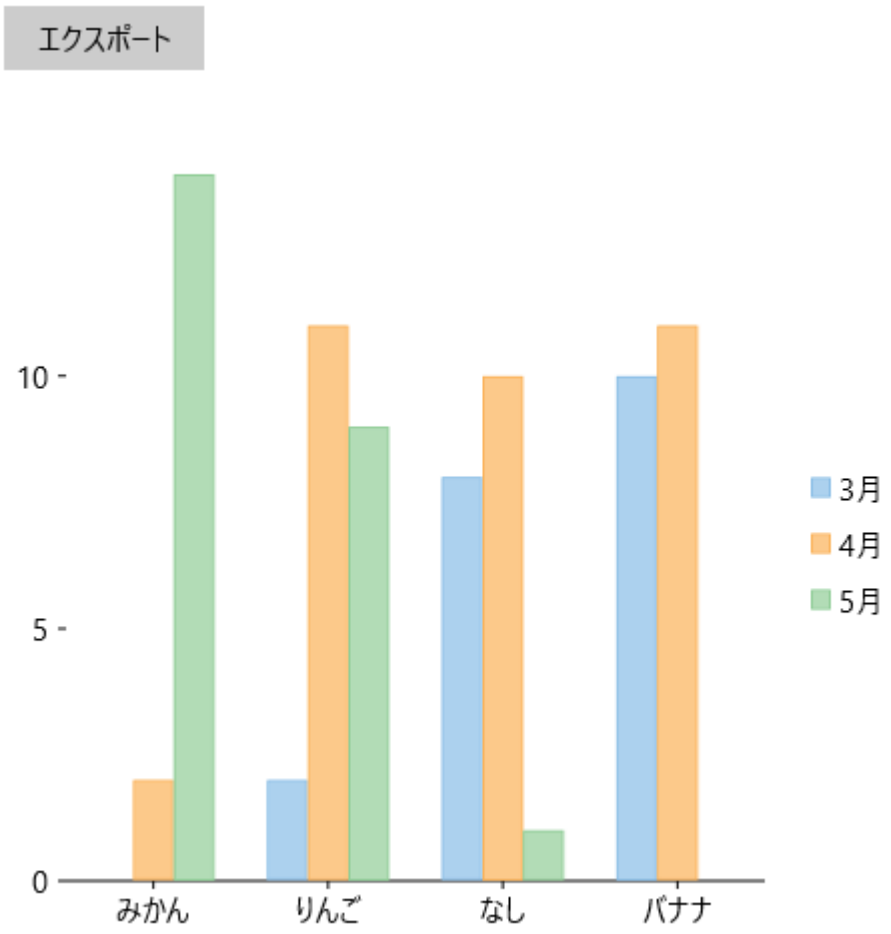
画像へのエクスポート

FlexChart for UWP では、FlexChart を複数の画像形式にエクスポートできます。サポートされる形式は、**PNG**、**JPEG**、**SVG** です。

FlexChart を画像形式にエクスポートするには、**SaveImage** メソッドを使用します。このメソッドは、指定されたストリームを使用してチャートを画像として保存します。SaveImage メソッドは、画像を保存するためのパラメータとして、ストリーム、画像形式、高さ、幅、背景色を受け取ります。

このトピックは、クイックスタートセクションで作成されたサンプルを使用して、ボタンクリックで FlexChart を画像にエクスポートするための実装方法を説明します。

次の図に、チャートをご希望の画像形式にエクスポートするためのボタンがクリックされたチャートを示します。



XAML

MainPage.xaml

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="93*"/>
            <RowDefinition Height="97*"/>
        </Grid.RowDefinitions>
        <StackPanel Orientation="Horizontal" Margin="10 0">
            <Button Content="エクスポート" Width="100" Margin="10,72,10,72.333"
Click="Button_Click_1"/>
        </StackPanel>
        <Chart:C1FlexChart x:Name="flexChart" HorizontalAlignment="Left"
Width="443" ItemsSource="{Binding DataContext.Data}"
                  BindingX="Fruit" Margin="93,106,0,70"
                  Grid.RowSpan="2">
            <Chart:C1FlexChart.Series>
                <Chart:Series SeriesName="3月" Binding="March"></Chart:Series>
                <Chart:Series SeriesName="4月" Binding="April"></Chart:Series>
                <Chart:Series SeriesName="5月" Binding="May"></Chart:Series>
            </Chart:C1FlexChart.Series>
            <Chart:C1FlexChart.AxisX>
                <Chart:Axis MajorGrid="False" Position="Bottom"></Chart:Axis>
```

```
            </Chart:C1FlexChart.AxisX>
            <Chart:C1FlexChart.AxisY>
                <Chart:Axis AxisLine="False" Position="Left" MajorUnit="5">
</Chart:Axis>
            </Chart:C1FlexChart.AxisY>
            <Chart:C1FlexChart.SelectionStyle>
                <Chart:ChartStyle Stroke="Red"></Chart:ChartStyle>
            </Chart:C1FlexChart.SelectionStyle>
        </Chart:C1FlexChart>
    </Grid>
```

Code

MainPage.xaml.cs

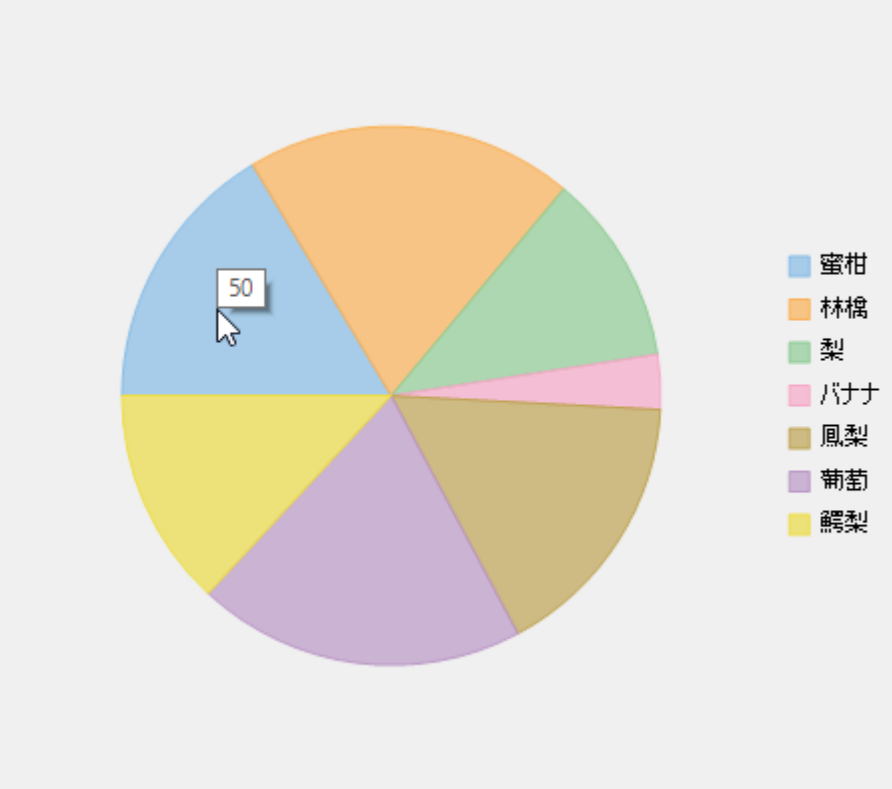
```
 public sealed partial class MainPage : Page
    {
        List _fruits;
        public MainPage()
        {
            this.InitializeComponent();
        }
        public List Data
        {
            get
            {
                if (_fruits == null)
                {
                    _fruits = DataCreator.CreateFruit();
                }

                return _fruits;
            }
        }
        private async void Button_Click_1(object sender, RoutedEventArgs e)
        {
            var picker = new FileSavePicker();
            Enum.GetNames(typeof(ImageFormat)).ToList().ForEach(fmt =>
            {
                picker.FileTypeChoices.Add(fmt.ToString().ToUpper(), new
List<string>() { "." + fmt.ToString().ToLower() });
            });
            StorageFile file = await picker.PickSaveFileAsync();
            if (file != null)
            {
                Stream stream = new MemoryStream();
                var extension = file.FileType.Substring(1, file.FileType.Length -
1);
                ImageFormat fmt = (ImageFormat)Enum.Parse(typeof(ImageFormat),
extension, true);
                flexChart.SaveImage(stream, fmt);
                using (Stream saveStream = await file.OpenStreamForWriteAsync())
```

```
                {
                    stream.CopyTo(saveStream);
                    stream.Seek(0, SeekOrigin.Begin);
                    saveStream.Seek(0, SeekOrigin.Begin);
                    saveStream.Flush();
                    saveStream.Dispose();
                }
            }
        }
    }
```

FlexPie

円グラフは、一般に系列内の項目のサイズを円全体のパーセンテージとして表現するために使用されます。理想的には、円グラフは、いくつかの正の値から成る 1 つの系列だけをプロットする場合に使用されます。カテゴリの数は 7 個以下です。

FlexPie コントロールを使用すると、データポイントを円グラフのいくつかのセグメントとして表すカスタマイズされた円グラフを作成できます。各セグメントの弧の長さが、そのセグメントの値を表現します。



主要な機能

- **ヘッダーとフッター**
  単純なプロパティを使用して、タイトルやフッターテキストを設定できます。
- **凡例**
  必要に応じて凡例の位置を変更できます。
- **選択**
  選択モードを変更し、円グラフのセグメントの外観をカスタマイズできます。
- **分割およびドーナツ円グラフ**
  単純なプロパティを使用して、円グラフを分割円グラフまたはドーナツ円グラフに変換できます。
- **Data Labels**: Add, style, format, set the position of data labels and manage the overlapped data labels on the chart.

クイックスタート

このクイックスタートでは、Visual Studio で単純な FlexPie アプリケーションを作成して実行する手順を説明します。

次の手順を実行して、アプリケーションの実行時に FlexPie がどのように表示されるかを理解してください。

- **手順 1：アプリケーションへの FlexPie の追加**

- **手順 2：データソースへの FlexPie の連結**
- **手順 3：アプリケーションの実行**

手順 1：アプリケーションへの FlexPie の追加

1. Visual Studio で、**[空のアプリケーション（ユニバーサル Windows）]**を作成します。
2. **C1FlexPie** コントロールを MainPage にドラッグアンドドロップします。
   次の dll が自動的にアプリケーションに追加されます。
   C1.UWP.dll
   C1.UWP.DX.dll
   C1.UWP.FlexChart.dll

   <Grid></Grid> タグ内の XAML マークアップは次のコードのようになります。

   - **XAML**

```
<Chart:C1FlexPie x:Name="flexPie" Binding="Value" BindingName="Name"
HorizontalAlignment="Left" Height="300" VerticalAlignment="Top" Width="300">
    <Chart:C1FlexPie.ItemsSource>
        <Chart:FlexPieSliceCollection>
            <Chart:FlexPieSlice Name="スライス1" Value="1"/>
            <Chart:FlexPieSlice Name="スライス2" Value="2"/>
            <Chart:FlexPieSlice Name="スライス3" Value="3"/>
            <Chart:FlexPieSlice Name="スライス4" Value="4"/>
        </Chart:FlexPieSliceCollection>
    </Chart:C1FlexPie.ItemsSource>
</Chart:C1FlexPie>
```

手順 2：データソースへの FlexPie の連結

1. クラス **DataCreator** を追加し、次のコードを追加します。

   - **Visual Basic**

```
Class DataCreator
    Public Shared Function CreateFruit() As List(Of FruitDataItem)
        Dim fruits = New String() {"蜜柑", "林檎", "梨", "バナナ"}
        Dim count = fruits.Length
        Dim result = New List(Of FruitDataItem)()
        Dim rnd = New Random()
        For i As Object = 0 To count - 1
            result.Add(New FruitDataItem() With {
                .Fruit = fruits(i),
                .March = rnd.[Next](20),
                .April = rnd.[Next](20),
                .May = rnd.[Next](20)
            })
        Next
        Return result
    End Function
End Class

Public Class FruitDataItem
    Public Property Fruit() As String
        Get
            Return m_Fruit
        End Get
        Set
            m_Fruit = Value
        End Set
    End Property
    Private m_Fruit As String
    Public Property March() As Double
```

```
            Get
                Return m_March
            End Get
            Set
                m_March = Value
            End Set
        End Property
        Private m_March As Double
        Public Property April() As Double
            Get
                Return m_April
            End Get
            Set
                m_April = Value
            End Set
        End Property
        Private m_April As Double
        Public Property May() As Double
            Get
                Return m_May
            End Get
            Set
                m_May = Value
            End Set
        End Property
        Private m_May As Double
    End Class
```

- **C#**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FlexPieQuickStart
{
    class DataCreator
    {
        public static List<FruitDataItem> CreateFruit()
        {
        var fruits = new string[] { "蜜柑", "林檎", "梨", "バナナ" };
            var count = fruits.Length;
            var result = new List<FruitDataItem>();
            var rnd = new Random();
            for (var i = 0; i < count; i++)
                result.Add(new FruitDataItem()
                {
                    Fruit = fruits[i],
                    March = rnd.Next(20),
                    April = rnd.Next(20),
                    May = rnd.Next(20),
                });
            return result;
        }
    }

    public class FruitDataItem
    {
        public string Fruit { get; set; }
        public double March { get; set; }
        public double April { get; set; }
        public double May { get; set; }
    }
```

```
}
```

2. **<Grid>** タグを編集してマークアップを次のように変更し、FlexPie にデータを提供します。

   - **XAML**

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Chart:C1FlexPie
        x:Name="flexPie"
        Binding="April"
        BindingName="Fruit"
        ItemsSource="{Binding DataContext.Data}">
        <Chart:C1FlexPie.SelectionStyle>
            <Chart:ChartStyle Stroke="Red"
                              StrokeThickness="2"/>
        </Chart:C1FlexPie.SelectionStyle>
        <Chart:C1FlexPie.DataLabel>
            <Chart:PieDataLabel Content="{}{y}"/>
        </Chart:C1FlexPie.DataLabel>
    </Chart:C1FlexPie>
</Grid>
```

> 連結ソースを指定するには、**DataContext = "{Binding RelativeSource={RelativeSource Mode=Self}}"** マークアップを **MainPage.xaml** ファイルの **<Page>** タグに追加する必要があります。

3. コードビュー（**MainPage.xaml.cs**）に切り替えて、次のコードを該当する名前空間内の **MainPage** クラスに追加します。

   - **Visual Basic**

```
Partial Public NotInheritable Class MainPage
    Inherits Page
    Private _data As List(Of FruitDataItem)
    Public Sub New()
        Me.InitializeComponent()
    End Sub
    Public ReadOnly Property Data() As List(Of FruitDataItem)
        Get
            If _data Is Nothing Then
                _data = DataCreator.CreateFruit()
            End If
            Return _data
        End Get
    End Property
End Class
```

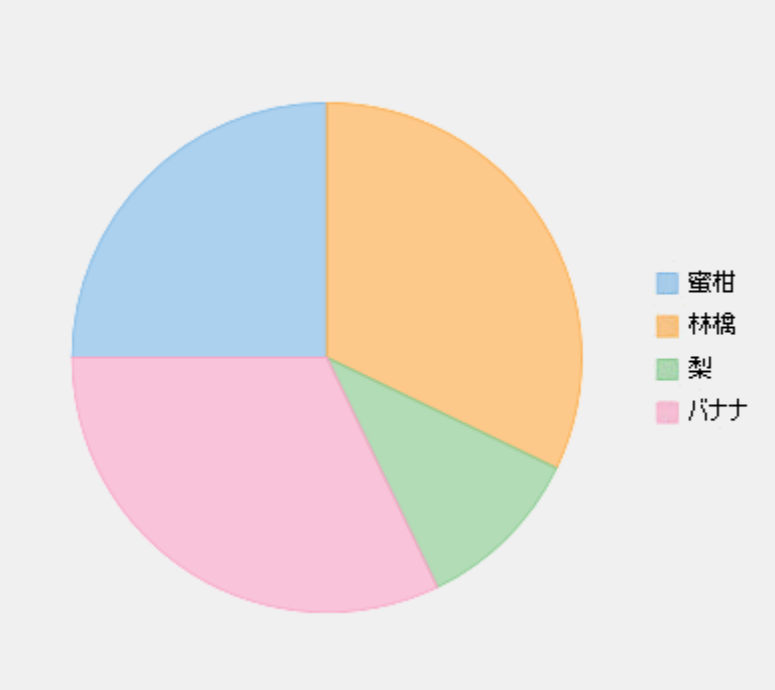
   - **C#**

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

namespace FlexPieQuickStart
{
    public sealed partial class MainPage : Page
```

```
        {
            List<FruitDataItem> _data;
            public MainPage()
            {
                this.InitializeComponent();
            }
            public List<FruitDataItem> Data
            {
                get
                {
                    if (_data == null)
                    {
                        _data = DataCreator.CreateFruit();
                    }
                    return _data;
                }
            }
        }
    }
```

手順 3:アプリケーションの実行

[F5]キーを押してアプリケーションを実行し、次のような出力を確認します.

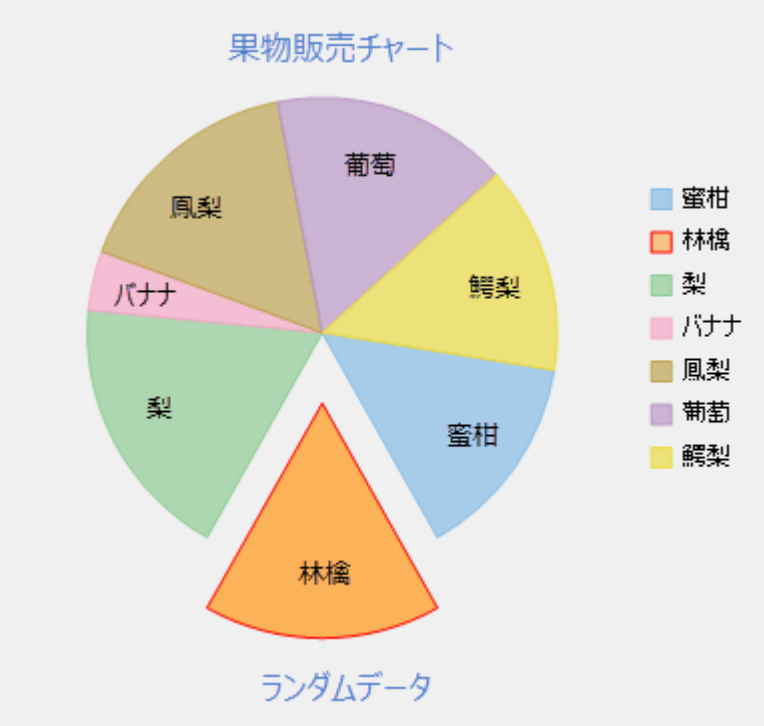


FlexPie の基本

FlexPie の主要な機能は次のとおりです。

- ヘッダーとフッター
- 凡例
- 選択
- ラベル位置

- 分割円グラフとドーナツ円グラフ



ヘッダーとフッター

ヘッダーとフッターを使用して、タイトルやフッターテキストを設定できます。**FlexChartBase** の **Header** プロパティを設定することで、**FlexPie** コントロールにヘッダーを追加できます。コントロールには、ヘッダーのほかにフッターも追加できます。それには、FlexChartBase の **Footer** プロパティを設定します。

凡例

FlexPie では、**FlexChartBase** の **LegendPosition** プロパティを使用して、凡例を表示する位置を指定できます。このプロパティには次のオプションがあります。

- Bottom
- Left
- Top
- Right
- Auto
- None

選択モード

コントロール上の任意の場所をクリックしたときに、**FlexPie** のどの要素を選択するかを決めることができます。それには、**SelectionMode** プロパティを設定します。このプロパティには次のオプションがあります。

- **None**：どの要素も選択されません。
- **Point**：ユーザーがクリックした円グラフセグメントが強調表示されます。
- **Series**：円全体を強調表示します。

SelectionMode プロパティを Point に設定すると、**SelectedItemPosition** プロパティを設定することで、選択されている円グラフセグメントの位置を変更できます。また、選択されている円グラフセグメントを FlexPie の中心から離すことができます。それには、**SelectedItemOffset** プロパティを設定します。

ラベル位置

次のオプションを使用して、FlexPie に対する **PieDataLabel.Position** の位置を選択できます。

- Center
- Inside
- None
- Outside

これらのプロパティの設定については、次の Xaml を参照してください。

XAML

```
<Chart:C1FlexPie x:Name="pieChart" Width="auto" Height="auto" Header="果物販売チャート"
Xaml:C1NagScreen.Nag="True" Footer="ランダムデータ" SelectedItemPosition="Bottom"
SelectedItemOffset="0.5" SelectionMode="Point" LegendPosition="Right" >
                <Chart:C1FlexPie.DataLabel>
                    <Chart:PieDataLabel Position="Inside"/>
                </Chart:C1FlexPie.DataLabel>
                <Chart:C1FlexPie.FooterStyle>
                    <Chart:ChartStyle FontSize="20" Stroke="BlueViolet"
FontStyle="Italic"/>
                </Chart:C1FlexPie.FooterStyle>
                <Chart:C1FlexPie.HeaderStyle>
                    <Chart:ChartStyle FontSize="20" Stroke="BlueViolet"
FontStyle="Italic"/>
                </Chart:C1FlexPie.HeaderStyle>
                <Chart:C1FlexPie.SelectionStyle>
                    <Chart:ChartStyle StrokeThickness="3" Stroke="BlueViolet" />
                </Chart:C1FlexPie.SelectionStyle>
            </Chart:C1FlexPie>
```

以下のセクションでは、分割円グラフとドーナツ円グラフの作成方法について説明します。

アニメーション

FlexPie allows you to control how the animation is applied to each series and series elements. It allows you to enable chart animation effects through a combination of different properties available in the FlexPie class. These properties allow you to apply duration, delay and an easing function for each animation. The animation effects are applied in one of the two scenarios, either while loading the chart for the first time or while the chart is redrawn after modifications.

The following image shows how animation works in the FlexPie control.

Use the following code to implement animation in the FlexPie control.

C#

```
// アニメーション
flexPie1.AnimationSettings = C1.Chart.AnimationSettings.Load;
flexPie1.AnimationUpdate.Easing = C1.Chart.Easing.Linear;
flexPie1.AnimationUpdate.Duration = 500;
flexPie1.AnimationLoad.Type = C1.Chart.AnimationType.Series;
```

Every animation scenario has its own set of options that can be set independently in the FlexChart control. These options include various properties describing the corresponding animation.

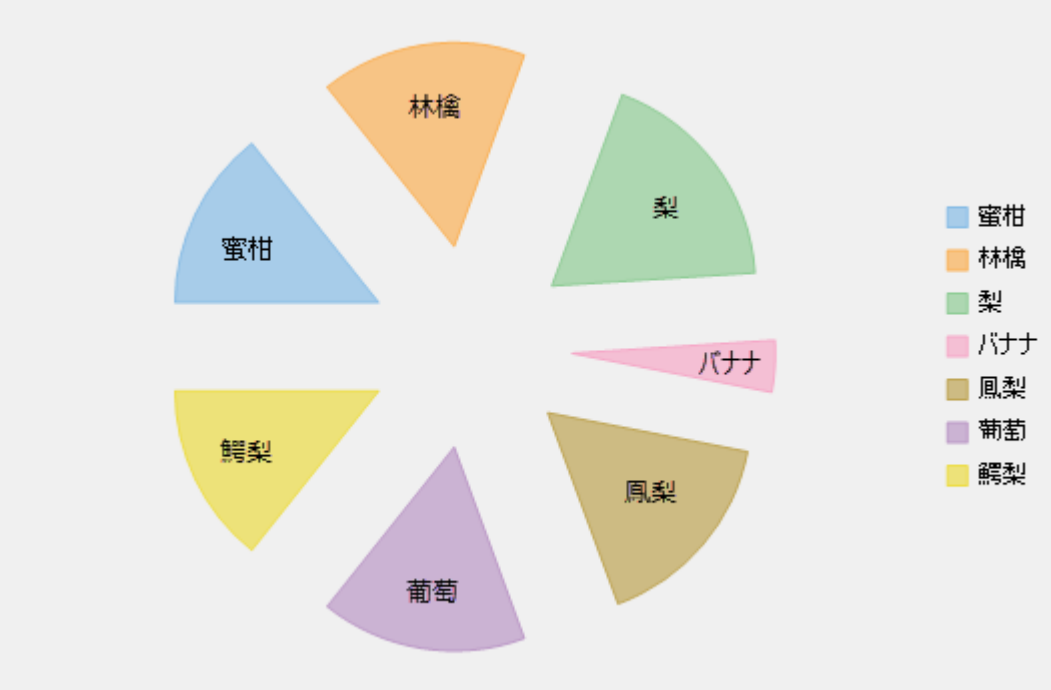
To implement animation in the FlexChart control, you need make use of the following properties.

1. **AnimationSettings** -This property allows the user to apply settings for the animation. It allows the user to specify that when to apply animation in the FlexChart control. This property accepts values from the **AnimationSettings** enumeration provided by the FlexChart class. The AnimationSettings enumeration has special flags to control axes animation (smooth transition) so that you can enable or disable smooth axis transition for loading or updating data.
2. AnimationOptions - The **AnimationLoad** and **AnimationUpdate** properties includes the following options.
    1. **Duration**: This property allows you to set the duration of animation in the FlexChart control. This property accepts an integer value which defines duration in milliseconds.
    2. **Easing**: This property allows the user to set different type of easing functions on the FlexChart control. This property accepts values from the **Easing** enumeration provided by the C1.Chart namespace.
    3. **Type**: This property allows you to set the animation type on the FlexChart control. This property accepts the following values from the **AnimationType** enumeration provided by the C1.Chart namespace.

- **All** - All plot elements animate at once from the bottom of the plot area.
- **Series** - Each series animates one at a time from the bottom of the plot area.
- **Points** - The plot elements appear one at a time from left to right.

分割円グラフ

Offset プロパティを使用して、FlexPie の中心から円グラフのセグメントを離して、分割円グラフを生成できます。このプロパティは、円グラフのセグメントを中心から離す距離を決定する double 値を受け入れます。



次のコードスニペットは、**Offset** プロパティを設定する方法を示します。

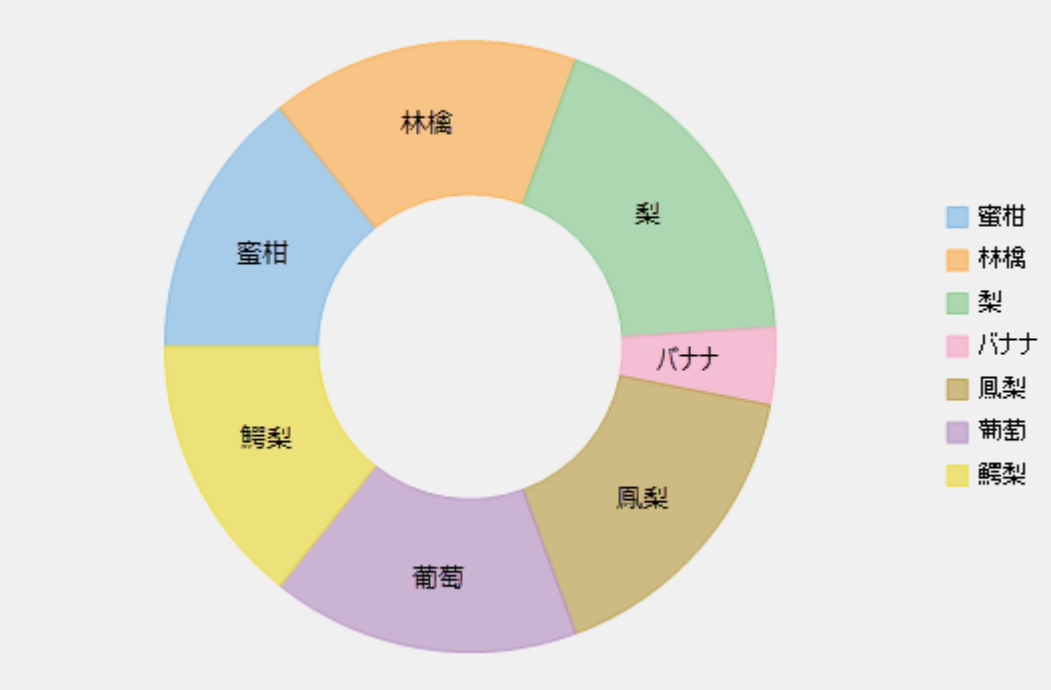
XAML

```
<Chart:C1FlexPie x:Name="pieChart" Width="auto" Height="auto" Offset="0.5"
LegendPosition="Right" </Chart:C1FlexPie>
```

ドーナツ円グラフ

FlexPie では、**InnerRadius** プロパティを使用してドーナツ円グラフを作成できます。

内側半径は、円グラフの半径に対する割合で表されます。 **InnerRadius** プロパティのデフォルト値は 0 です。その場合は円グラフが作成されます。このプロパティを 0 より大きい値に設定すると、中央に穴のある円グラフが作成されます。これは、**ドーナツグラフ**とも呼ばれます。



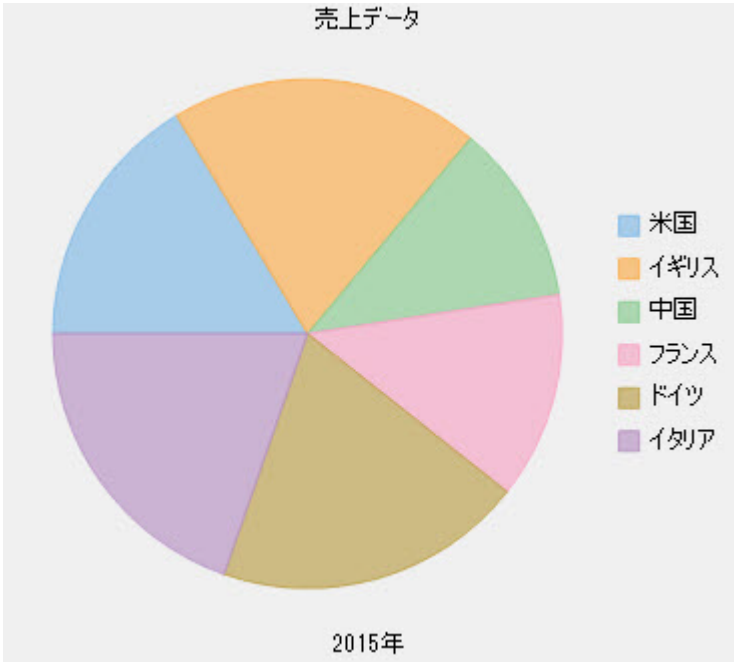
次のコードスニペットは、**InnerRadius** プロパティを設定する方法を示します。

XAML

```
<Chart:C1FlexPie x:Name="pieChart" Width="auto" Height="auto" InnerRadius="0.5"
LegendPosition="Right" </Chart:C1FlexPie>
```

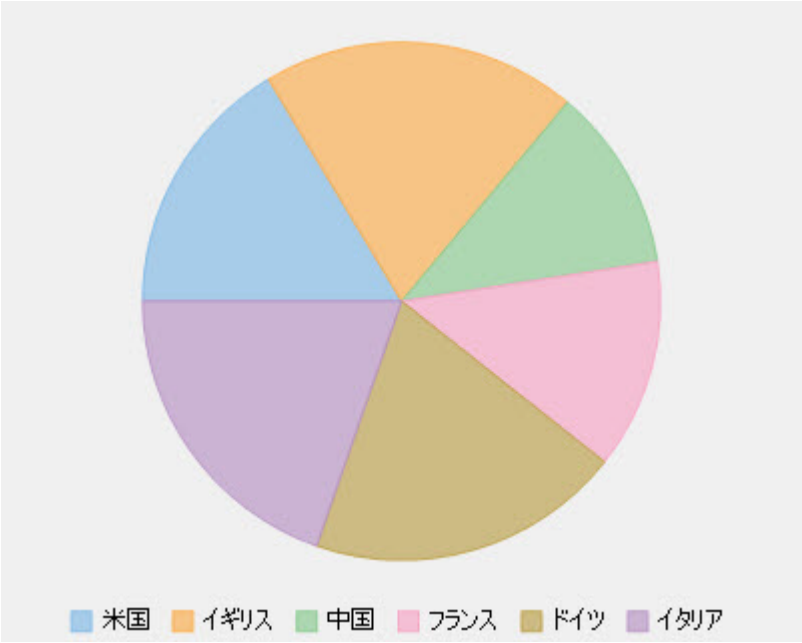
ヘッダーとフッター

FlexChartBase の **Header** プロパティを設定することで、**FlexPie** コントロールにヘッダーを追加できます。コントロールには、ヘッダーのほかにフッターも追加できます。それには、**FlexChartBase** の **Footer** プロパティを設定します。



凡例

FlexPie では、FlexChartBase の **LegendPostion** プロパティを使用して、凡例を表示する位置を指定できます。

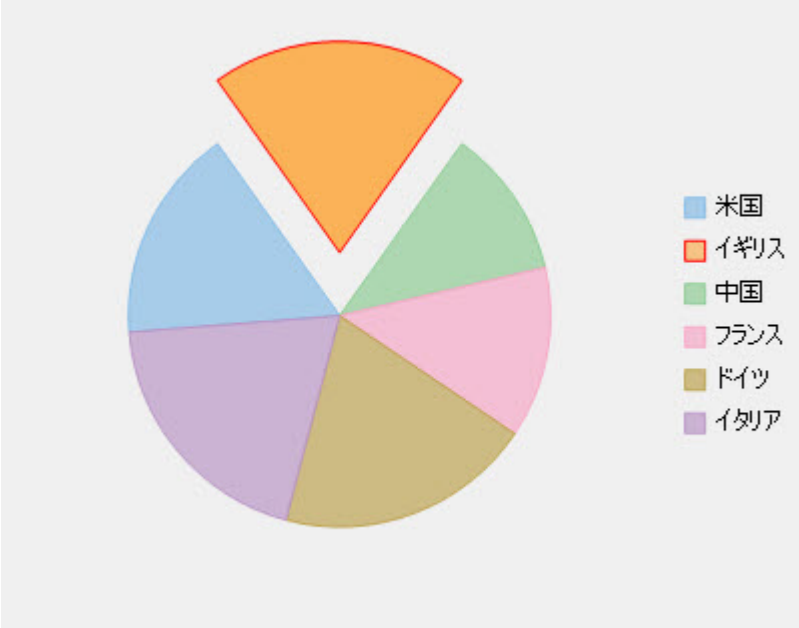


選択

コントロール上の任意の場所をクリックしたときに、**FlexPie** のどの要素を選択するかを決めることができます。それには、**SelectionMode** プロパティを設定します。このプロパティは 3 つのオプションを提供します。

- **None**：どの要素も選択されません。
- **Point**：ユーザーがクリックした円グラフセグメントが強調表示されます。
- **Series**：円全体を強調表示します。

SelectionMode プロパティを Point に設定すると、**SelectedItemPosition** プロパティを設定することで、選択されている円グラフセグメントの位置を変更できます。また、選択されている円グラフセグメントを FlexPie の中心から離すことができます。それには、**SelectedItemOffset** プロパティを設定します。



データラベル

Data labels provide additional information about the data points. These labels make a chart easier to understand because they show details about a slice in the pie.

To understand the working of data labels in FlexPie chart, refer to the following sections.

データラベルの追加と配置
: Learn how to add data labels and set their position on the chart.

データラベルの書式設定
: Learn how to perform styling and formatting of data labels.

重なったデータラベルの管理
: Learn how to manage overlapping data labels in FlexPie chart.

データラベルの追加と配置

With FlexPie chart, you can configure the arrangement and display properties for data labels depending on what suits your needs the best. By default, the data labels are not displayed on the chart, however, you can enable them by setting **Position** and **Content** properties of the DataLabel class. Here, the **Position** property sets position of the data labels by accepting values from the **PieLabelPosition** enumeration and the **Content** property gets or sets content for the data labels.

The example code below uses the **Position** and **Content** properties to enable data labels and set their position.

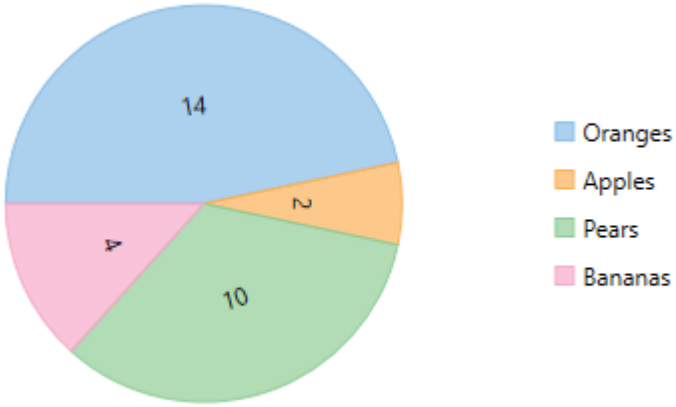
XAML

```
<c1:C1FlexPie.DataLabel> <c1:PieDataLabel Content="{}{y}" Position="Circular"/>
</c1:C1FlexPie.DataLabel>
```

Code

HTML

```
flexPie.DataLabel.Content = "{}{y}";
flexPie.DataLabel.Position = C1.Chart.PieLabelPosition.Circular;
```



データラベルの書式設定

FlexPie provides various options to format data labels according to your requirements. You can use connecting lines to connect the data labels, set and style borders of data labels, and customize the appearance of data labels.

The topic comprises of three sections:

- **Setting and Styling Borders**
- **Connecting DataLabels to Data Points**
- **Modifying Appearance**

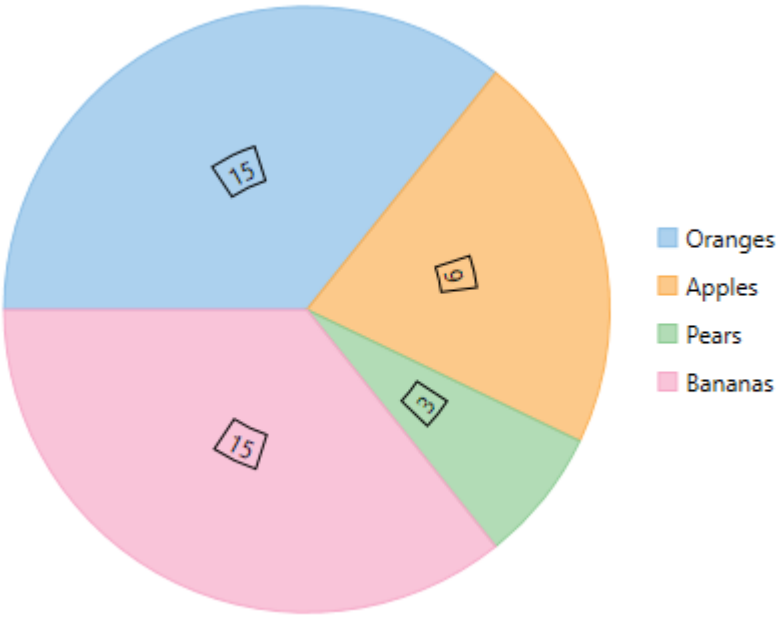
Setting and Styling Borders

To add and style borders to FlexPie data labels, set the **Border** and **BorderStyle** properties provided by **DataLabel** class.

Use the following code snippet to add borders to data labels of FlexPie.

C#

```
// 境界線を有効にします
flexPie.DataLabel.Border = true;
```



Connecting DataLabels to Data Points

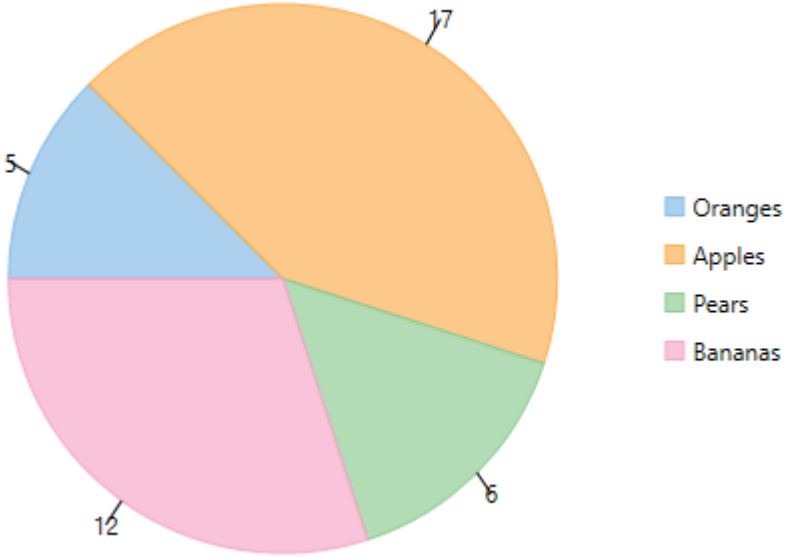
In case the data labels are placed away from the data points, you can connect them using connecting lines.

To enable connecting lines in FlexPie chart, you need to use the **ConnectingLine** property.

Use the following code snippet to set the connecting lines.

C#

```
// 接続線を有効にします
flexPie.DataLabel.ConnectingLine = true;
```



Modifying Appearance

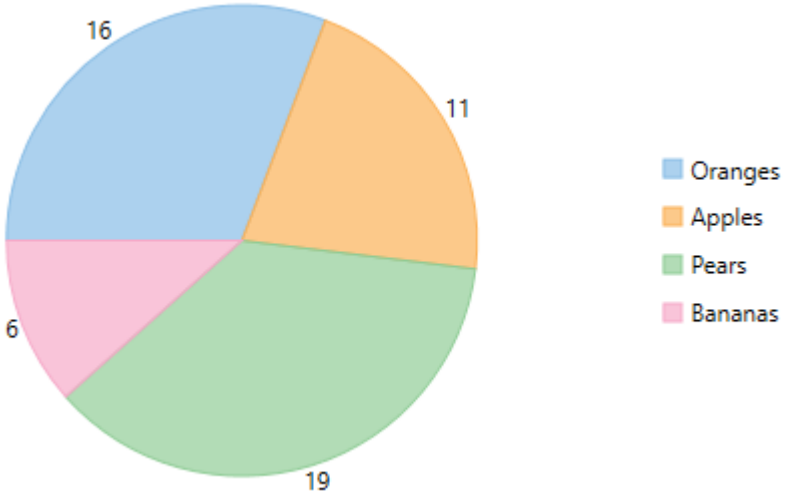
FlexPie includes various styling options, to enhance the clarity of data labels. To modify the appearance of FlexPie

chart, you need to use the **Style** property. It allows you to modify the font family, fill color, use stroke brush for data labels, set width for stroke brush and more.

In the example code, we have modified the font used in the chart and set the stroke width property. Use the following code snippet to modify the appearance of the chart.

C#

```
// データラベルの外観を変更します
flexPie.DataLabel.Style.FontFamily = new FontFamily("GenericSerif");
flexPie.DataLabel.Style.StrokeThickness = 2;
```



重なったデータラベルの管理

A common issue pertaining to charts is overlapping of data labels that represent data points. In most cases, overlapping occurs due to long text in data labels or large numbers of data points.

To manage overlapped data labels in FlexPie chart, you can make use of **Overlapping** property provided by **PieDataLabel** class. The Overlapping property accepts the following values from the PieLabelOverlapping enumeration.

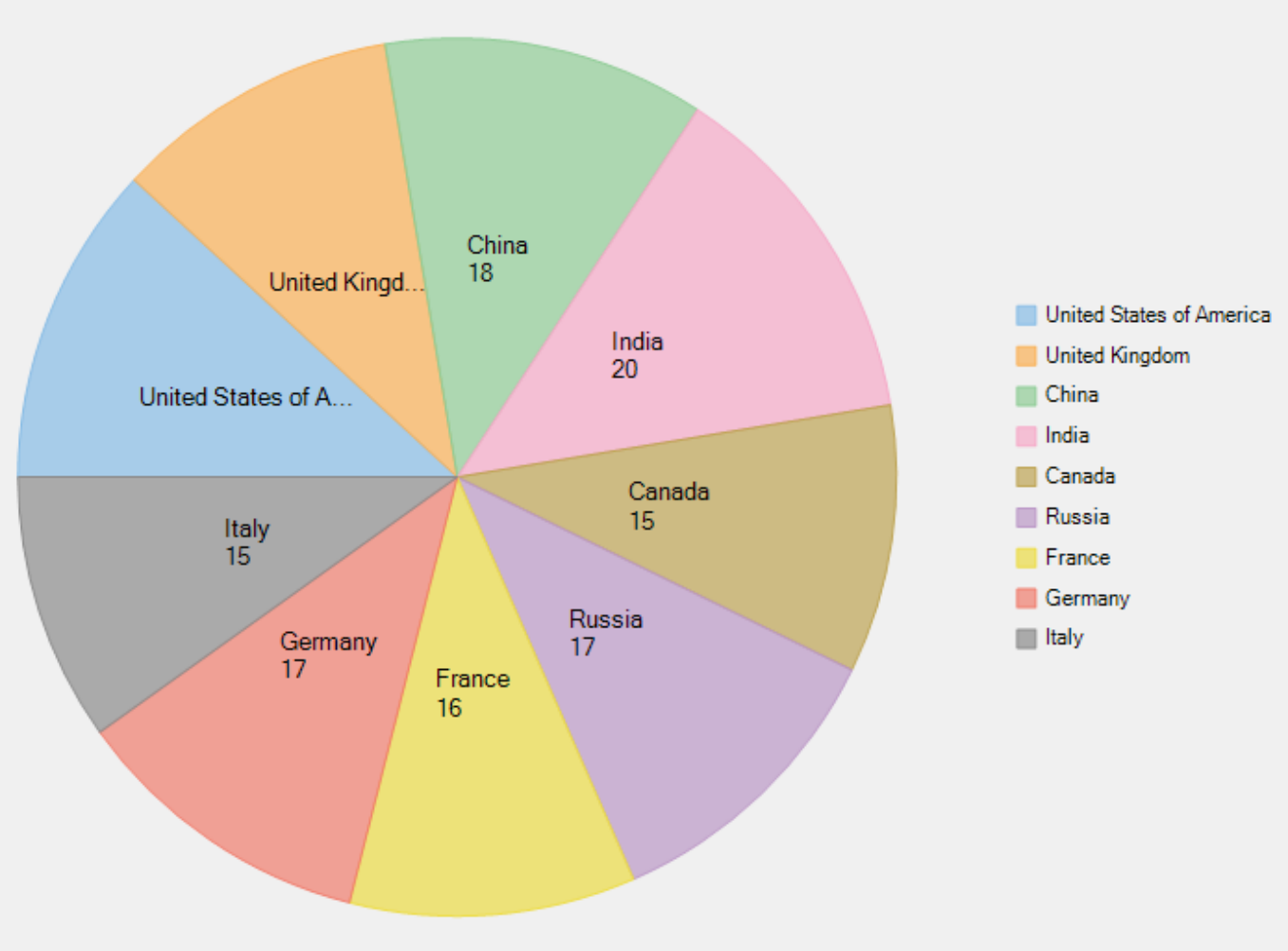
Enumeration	Description
PieLabelOverlapping.Default	Show all labels including the overlapping ones.
PieLabelOverlapping.Hide	Hides the overlapping labels, if its content is larger than the corresponding pie segment.
PieLabelOverlapping.Trim	Trim overlapping data labels, if its width is larger than the corresponding pie segment.

Use the following code to manage overlapping data labels.

C#

```
// Overlapping プロパティを設定します
flexPie.DataLabel.Overlapping = C1.Chart.PieLabelOverlapping.Trim;
```

The following image shows how FlexPie appears after setting the Overlapping property.



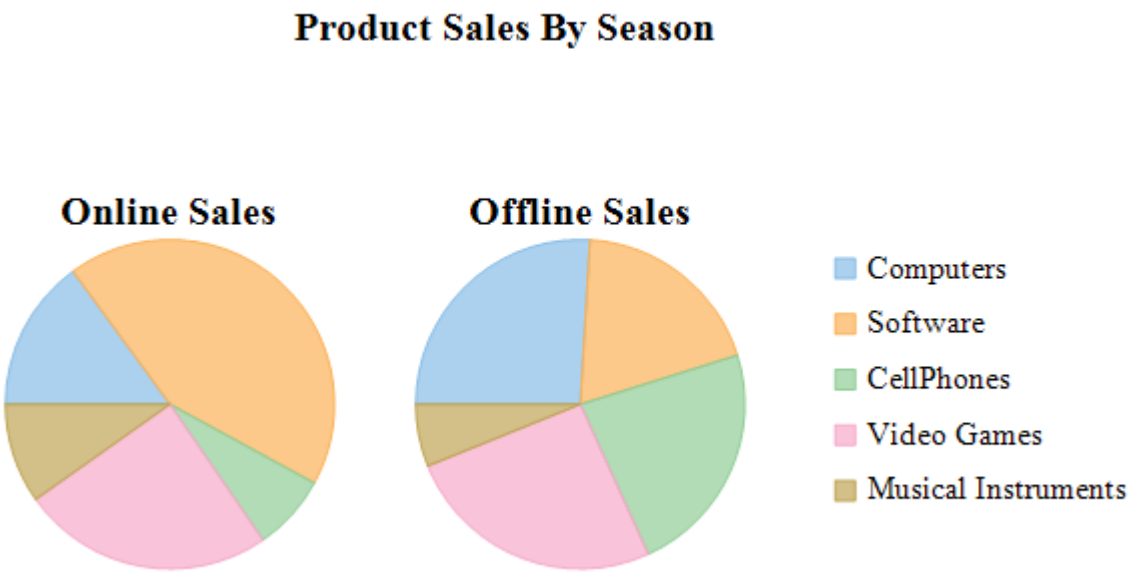
複数の円グラフ

In some scenarios, you might want to visualize a bit more than a single pie chart, and use multiple Pie Charts together. The **FlexPie** control empowers a user to create multiple Pie Charts based on the same data source. The user can create multiple pie charts by simply specifying several comma-separated strings of field names in the **Binding** property of **C1FlexPie** class. This means you don't have to add any additional FlexPie controls for multiple Pie Charts.

Multiple Pie Charts can help compare data across different groups. It is also the right visualization choice to present key indicators/facts about your topic of interest.

The following image shows two Pie Charts, Online Sales and Offline Sales for five different products. Computer, Software, CellPhones, Video Games and Musical Instruments.

Adding multiple pie charts using the same data source is depicted programmatically below:



Adding multiple pie charts using the same data source is depicted programmatically below:

C#

Code

```
flexPie.ItemsSource = data;
            flexPie.BindingName = "Name";
            flexPie.Binding = "Online,Offline";
            flexPie.Titles = new[] { "Online Sales", "Offline Sales" };
            flexPie.Header = "Product Sales By Season";
```

Visual Basic

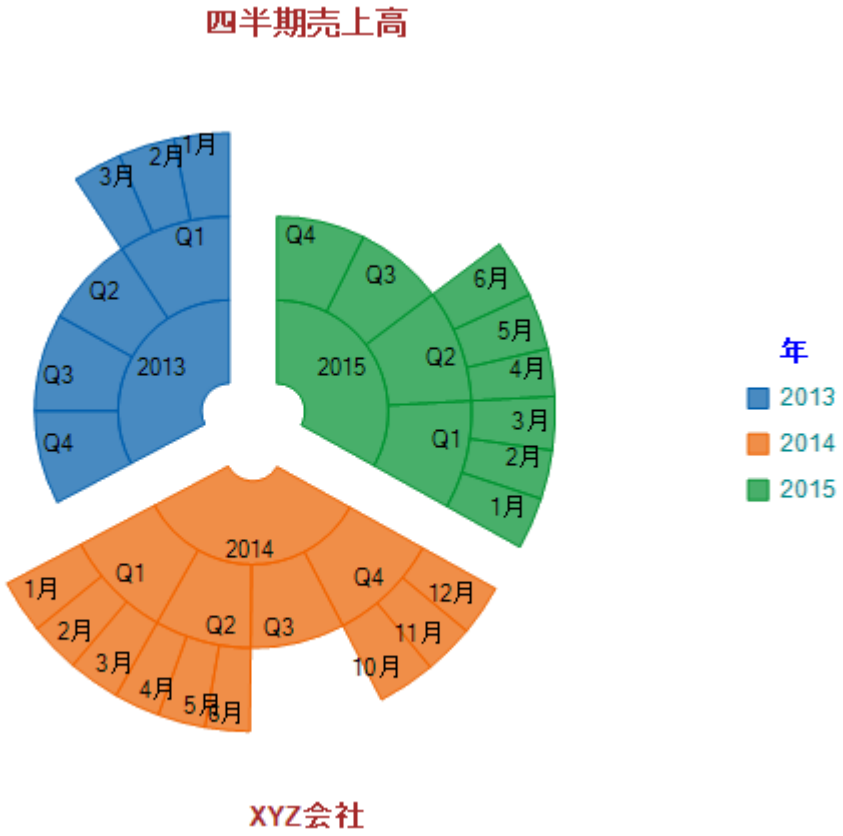
Code

```
Private Sub Page_Loaded(sender As Object, e As RoutedEventArgs)
        flexPie.ItemsSource = data
        flexPie.BindingName = "Name"
        flexPie.Binding = "Online,Offline"
        flexPie.Titles = {"Online Sales", "Offline Sales"}
        flexPie.Header = "Product Sales By Season"
    End Sub
```

Sunburst

Sunburst は、マルチレベル円グラフとも呼ばれ、同心円で表される複数の層から成る階層化データを視覚化するために適しています。中心の円はルートノードを表し、データは中心から外に向けて広がります。内側の円の 1 つのセクションは、外側の円のいくつかのセクションと階層的な関係を持ち、外側の円のセクションは親セクションの角度範囲に収まります。

Sunburst チャートを使用すると、外側の輪と内側の輪の関係を視覚化できます。たとえば、3 年間の各四半期の売上レポートを表示したいとします。Sunburst チャートを使用すると、特定の月の売上レポートを強調表示して、対応する四半期との関係を表すことができます。



Sunburst チャートの機能の詳細については、次の各リンクをクリックしてください。

- クイックスタート
- 主要な機能
- 凡例とタイトル
- 選択
- ドリルダウン
- Data Labels

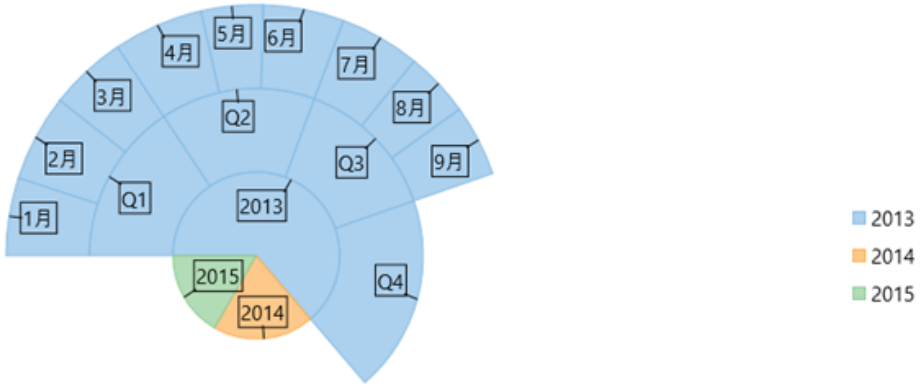
クイックスタート

このクイックスタートでは、Visual Studio で単純な Sunburst アプリケーションを作成して実行する手順を説明します。

Sunburst チャートの使用をすぐに開始し、アプリケーション実行時にどのように表示されるかを確認するには、次の手順に従います。

1. **アプリケーションへの Sunburst チャートの追加**
2. **データソースへの Sunburst チャートの連結**
3. **アプリケーションの実行**

次の図に、上記の手順を完了した後に、基本的な Sunburst チャートがどのように表示されるかを示します。



手順 1：アプリケーションへの Sunburst チャートの追加

1. Visual Studio で、**[空のアプリケーション（ユニバーサル Windows）]**を作成します。
2. **C1Sunburst** コントロールを MainPage にドラッグアンドドロップします。
   次の dll が自動的にアプリケーションに追加されます。
   C1.UWP.dll
   C1.UWP.DX.dll
   C1.UWP.FlexChart.dll

   <Grid></Grid> タグ内の XAML マークアップは次のコードのようになります。

   - **XAML**

```
<Grid>
    <Chart:C1Sunburst x:Name="flexPie"
                      Binding="Value"
                      BindingName="Name"
                      HorizontalAlignment="Left"
                      Height="300"
                      VerticalAlignment="Top"
                      Width="300">
        <Chart:C1Sunburst.ItemsSource>
            <Chart:FlexPieSliceCollection>
                <Chart:FlexPieSlice Name="スライス1" Value="1"/>
                <Chart:FlexPieSlice Name="スライス2" Value="2"/>
                <Chart:FlexPieSlice Name="スライス3" Value="3"/>
                <Chart:FlexPieSlice Name="スライス4" Value="4"/>
            </Chart:FlexPieSliceCollection>
        </Chart:C1Sunburst.ItemsSource>
    </Chart:C1Sunburst>
</Grid>
```

手順 2：データソースへの Sunburst チャートの連結

この手順では、まず、2013、2014、2015 年の各 4 四半期分（Q1、Q2、Q3、Q4）のランダムな売上データを生成する DataService クラスを作成します。次に、FlexChartBase クラスで提供される ItemsSource プロパティを使用して、作成したクラスに Sunburst を連結します。さらに、**FlexChartBase** クラスと **FlexPie** クラスのそれぞれ **Binding** プロパティと **BindingName** プロパティを使用して、Sunburst グラフセグメントの数値とラベルを指定します。

1. クラス **DataService** を追加し、次のコードを追加します。

   - **Visual Basic**

```
Public Class DataService
    Private rnd As New Random()
    Shared _default As DataService

    Public Shared ReadOnly Property Instance() As DataService
        Get
            If _default Is Nothing Then
                _default = New DataService()
            End If

            Return _default
        End Get
    End Property
```

```
Public Shared Function CreateHierarchicalData() As List(Of DataItem)
    Dim rnd As Random = Instance.rnd

    Dim years As New List(Of String)()
    Dim times As New List(Of List(Of String))() From {
        New List(Of String)() From {
            "1月",
            "2月",
            "3月"
        },
        New List(Of String)() From {
            "4月",
            "5月",
            "6月"
        },
        New List(Of String)() From {
            "7月",
            "8月",
            "9月"
        },
        New List(Of String)() From {
            "10月",
            "11月",
            "12月"
        }
    }

    Dim items As New List(Of DataItem)()
    Dim yearLen = Math.Max(CInt(Math.Round(Math.Abs(5 - Instance.rnd.NextDouble() * 10))), 3)
    Dim currentYear As Integer = DateTime.Now.Year
    For i As Integer = yearLen To 1 Step -1
        years.Add((currentYear - i).ToString())
    Next
    Dim quarterAdded = False

    years.ForEach(Function(y)
                      Dim i = years.IndexOf(y)
                      Dim addQuarter = Instance.rnd.NextDouble() > 0.5
                      If Not quarterAdded AndAlso i = years.Count - 1 Then
                          addQuarter = True
                      End If
                      Dim year = New DataItem() With {
                          .year = y
                      }
                      If addQuarter Then
                  quarterAdded = True
                          times.ForEach(Function(q)
                                  Dim addMonth = Instance.rnd.NextDouble() > 0.5
                                  Dim idx As Integer = times.IndexOf(q)
                                  Dim quar As String = "Q" + (idx + 1).ToString()
                                  Dim quarters = New DataItem() With {
                                         .year = y,
                                         .Quarter = quar
                                             }
                       If addMonth Then
                         q.ForEach(Function(m)
                                  quarters.Items.Add(New DataItem() With {
                                          .year = y,
                                          .Quarter = quar,
                                         .Month = m,
                                          .Value = rnd.[Next](20, 30)
                                                          })

                                                 End Function)
                                     Else
                            quarters.Value = rnd.[Next](80, 100)
                                     End If
                                      year.Items.Add(quarters)

                                   End Function)
                      Else
                          year.Value = rnd.[Next](80, 100)
                      End If
                      items.Add(year)

                  End Function)
```

```
    Return items
End Function

Public Shared Function CreateFlatData() As List(Of FlatDataItem)
    Dim rnd As Random = Instance.rnd
    Dim years As New List(Of String)()
    Dim times As New List(Of List(Of String))() From {
        New List(Of String)() From {
            "1月",
            "2月",
            "3月"
        },
        New List(Of String)() From {
            "4月",
            "5月",
            "6月"
        },
        New List(Of String)() From {
            "7月",
            "8月",
            "9月"
        },
        New List(Of String)() From {
            "10月",
            "11月",
            "12月"
        }
    }

    Dim items As New List(Of FlatDataItem)()
    Dim yearLen = Math.Max(CInt(Math.Round(Math.Abs(5 - rnd.NextDouble() * 10))), 3)
    Dim currentYear As Integer = DateTime.Now.Year
    For i As Integer = yearLen To 1 Step -1
        years.Add((currentYear - i).ToString())
    Next
    Dim quarterAdded = False
    years.ForEach(Function(y)
                      Dim i = years.IndexOf(y)
                      Dim addQuarter = rnd.NextDouble() > 0.5
                      If Not quarterAdded AndAlso i = years.Count - 1 Then
                          addQuarter = True
                  End If
                  If addQuarter Then
                       quarterAdded = True
                       times.ForEach(Function(q)
                           Dim addMonth = rnd.NextDouble() > 0.5
                           Dim idx As Integer = times.IndexOf(q)
                           Dim quar As String = "Q" + (idx + 1).ToString()
                           If addMonth Then
                                        q.ForEach(Function(m)
                                                items.Add(New FlatDataItem() With {
                                    .Year = y,
                                    .Quarter = quar,
                                    .Month = m,
                                    .Value = rnd.[Next](30, 40)
                                           })

                                                                End Function)
                   Else
                        items.Add(New FlatDataItem() With {
                                    .Year = y,
                                    .Quarter = quar,
                                    .Value = rnd.[Next](80, 100)
                                          })
                                End If

                                        End Function)
                    Else
                        items.Add(New FlatDataItem() With {
                            .Year = y.ToString(),
                            .Value = rnd.[Next](80, 100)
                        })
                    End If

                End Function)
```

```
        Return items
    End Function
End Class
```

- **C#**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SunburstQuickStart
{
    public class DataService
    {
        Random rnd = new Random();
        static DataService _default;

        public static DataService Instance
        {
            get
            {
                if (_default == null)
                {
                    _default = new DataService();
                }

                return _default;
            }
        }

        public static List<DataItem> CreateHierarchicalData()
        {
            Random rnd = Instance.rnd;

            List<string> years = new List<string>();
            List<List<string>> times = new List<List<string>>()
            {
                new List<string>() { "1月", "2月", "3月"},
                new List<string>() { "4月", "5月", "6月"},
                new List<string>() { "7月", "8月", "9月"},
                new List<string>() { "10月", "11月", "12月" }
            };

            List<DataItem> items = new List<DataItem>();
            var yearLen = Math.Max((int)Math.Round(Math.Abs(5 - Instance.rnd.NextDouble() * 10)), 3);
            int currentYear = DateTime.Now.Year;
            for (int i = yearLen; i > 0; i--)
            {
                years.Add((currentYear - i).ToString());
            }
            var quarterAdded = false;

            years.ForEach(y =>
            {
                var i = years.IndexOf(y);
                var addQuarter = Instance.rnd.NextDouble() > 0.5;
                if (!quarterAdded && i == years.Count - 1)
                {
                    addQuarter = true;
                }
                var year = new DataItem() { Year = y };
                if (addQuarter)
                {
                    quarterAdded = true;
                    times.ForEach(q =>
                    {
                        var addMonth = Instance.rnd.NextDouble() > 0.5;
                        int idx = times.IndexOf(q);
                        var quar = "Q" + (idx + 1);
                        var quarters = new DataItem() { Year = y, Quarter = quar };
                        if (addMonth)
                        {
                            q.ForEach(m =>
                            {
                                quarters.Items.Add(new DataItem()
                                {
                                    Year = y,
```

```
                            Quarter = quar,
                            Month = m,
                            Value = rnd.Next(20, 30)
                        });
                    });
                }
                else
                {
                    quarters.Value = rnd.Next(80, 100);
                }
                year.Items.Add(quarters);
            });
        }
        else
        {
            year.Value = rnd.Next(80, 100);
        }
        items.Add(year);
    });

    return items;
}

public static List<FlatDataItem> CreateFlatData()
{
    Random rnd = Instance.rnd;
    List<string> years = new List<string>();
    List<List<string>> times = new List<List<string>>()
    {
        new List<string>() { "1月", "2月", "3月"},
        new List<string>() { "4月", "5月", "6月"},
        new List<string>() { "7月", "8月", "9月"},
        new List<string>() { "10月", "11月", "12月" }
    };

    List<FlatDataItem> items = new List<FlatDataItem>();
    var yearLen = Math.Max((int)Math.Round(Math.Abs(5 - rnd.NextDouble() * 10)), 3);
    int currentYear = DateTime.Now.Year;
    for (int i = yearLen; i > 0; i--)
    {
        years.Add((currentYear - i).ToString());
    }
    var quarterAdded = false;
    years.ForEach(y =>
    {
        var i = years.IndexOf(y);
        var addQuarter = rnd.NextDouble() > 0.5;
        if (!quarterAdded && i == years.Count - 1)
        {
            addQuarter = true;
        }
        if (addQuarter)
        {
            quarterAdded = true;
            times.ForEach(q =>
            {
                var addMonth = rnd.NextDouble() > 0.5;
                int idx = times.IndexOf(q);
                var quar = "Q" + (idx + 1);
                if (addMonth)
                {
                    q.ForEach(m =>
                    {
                        items.Add(new FlatDataItem()
                        {
                            Year = y,
                            Quarter = quar,
                            Month = m,
                            Value = rnd.Next(30, 40)
                        });
                    });
                }
                else
                {
                    items.Add(new FlatDataItem()
                    {
                        Year = y,
```

```
                                Quarter = quar,
                                Value = rnd.Next(80, 100)
                            });
                        }
                    });
                }
                else
                {
                    items.Add(new FlatDataItem()
                    {
                        Year = y.ToString(),
                        Value = rnd.Next(80, 100)
                    });
                }
            });

            return items;
        }
    }
}
```

2. クラス **SunburstViewModel** を追加し、次のコードを追加します。

   - **Visual Basic**

```
Imports System.Linq
Imports System.ComponentModel
Imports System.Collections.Generic
Imports System
Imports C1.Chart

Public Class SunburstViewModel

    Public ReadOnly Property HierarchicalData() As List(Of DataItem)
        Get
            Return DataService.CreateHierarchicalData()
        End Get
    End Property

    Public ReadOnly Property FlatData() As List(Of FlatDataItem)
        Get
            Return DataService.CreateFlatData()
        End Get
    End Property

    Public ReadOnly Property Positions() As List(Of String)
        Get
            Return [Enum].GetNames(GetType(Position)).ToList()
        End Get
    End Property

    Public ReadOnly Property Palettes() As List(Of String)
        Get
            Return [Enum].GetNames(GetType(Palette)).ToList()
        End Get
    End Property
End Class
```

   - **C#**

```
using C1.Chart;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;

namespace SunburstQuickStart
{
    public class SunburstViewModel
    {
        public List<DataItem> HierarchicalData
        {
            get
            {
                return DataService.CreateHierarchicalData();
            }
        }

        public List<FlatDataItem> FlatData
        {
            get
```

```
            {
                return DataService.CreateFlatData();
            }
        }

        public List<string> Positions
        {
            get
            {
                return Enum.GetNames(typeof(Position)).ToList();
            }
        }

        public List<string> Palettes
        {
            get
            {
                return Enum.GetNames(typeof(Palette)).ToList();
            }
        }
    }
}
```

3. クラス **DataItem** を追加し、次のコードを追加します。

    - **Visual Basic**

```
Public Class DataItem
    Private _items As List(Of DataItem)

    Public Property Year() As String
        Get
            Return m_Year
        End Get
        Set
            m_Year = Value
        End Set
    End Property
    Private m_Year As String
    Public Property Quarter() As String
        Get
            Return m_Quarter
        End Get
        Set
            m_Quarter = Value
        End Set
    End Property
    Private m_Quarter As String
    Public Property Month() As String
        Get
            Return m_Month
        End Get
        Set
            m_Month = Value
        End Set
    End Property
    Private m_Month As String
    Public Property Value() As Double
        Get
            Return m_Value
        End Get
        Set
            m_Value = Value
        End Set
    End Property
    Private m_Value As Double
    Public ReadOnly Property Items() As List(Of DataItem)
        Get
            If _items Is Nothing Then
                _items = New List(Of DataItem)()
            End If

            Return _items
        End Get
    End Property
End Class

Public Class FlatDataItem
    Public Property Year() As String
```

```
        Get
            Return m_Year
        End Get
        Set
            m_Year = Value
        End Set
    End Property
    Private m_Year As String
    Public Property Quarter() As String
        Get
            Return m_Quarter
        End Get
        Set
            m_Quarter = Value
        End Set
    End Property
    Private m_Quarter As String
    Public Property Month() As String
        Get
            Return m_Month
        End Get
        Set
            m_Month = Value
        End Set
    End Property
    Private m_Month As String
    Public Property Value() As Double
        Get
            Return m_Value
        End Get
        Set
            m_Value = Value
        End Set
    End Property
    Private m_Value As Double
End Class
```

- **C#**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SunburstQuickStart
{
    public class DataItem
    {
        List<DataItem> _items;

        public string Year { get; set; }
        public string Quarter { get; set; }
        public string Month { get; set; }
        public double Value { get; set; }
        public List<DataItem> Items
        {
            get
            {
                if (_items == null)
                {
                    _items = new List<DataItem>();
                }

                return _items;
            }
        }
    }

    public class FlatDataItem
    {
        public string Year { get; set; }
        public string Quarter { get; set; }
        public string Month { get; set; }
        public double Value { get; set; }
    }
}
```

4. クラス **Converter** を追加し、次のコードを追加します。

- **Visual Basic**

```
Imports Windows.UI.Xaml.Data
Imports System
Imports C1.Chart

Public Class EnumToStringConverter
    Implements IValueConverter
    Public Function Convert(value As Object, targetType As Type,
parameter As Object, language As String) As Object Implements IValueConverter.Convert
        Return value.ToString()
    End Function

    Public Function ConvertBack(value As Object, targetType As Type,
parameter As Object, language As String) As Object Implements IValueConverter.ConvertBack
        If targetType Is GetType(Position) Then
            Return CType([Enum].Parse(GetType(Position), value.ToString()), Position)
        Else
            Return CType([Enum].Parse(GetType(Palette), value.ToString()), Palette)
        End If
    End Function
End Class

Public Class StringToEnumConverter
    Implements IValueConverter
    Public Function Convert(value As Object, targetType As Type,
parameter As Object, language As String) As Object Implements IValueConverter.Convert
        If targetType Is GetType(Position) Then
            Return CType([Enum].Parse(GetType(Position), value.ToString()), Position)
        End If

        Return Nothing
    End Function

    Public Function ConvertBack(value As Object, targetType As Type,
parameter As Object, language As String) As Object Implements IValueConverter.ConvertBack
        Throw New NotImplementedException()
    End Function
End Class
```

- **C#**

```
using C1.Chart;
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Windows.UI.Xaml.Data;

namespace SunburstQuickStart
{
    public class EnumToStringConverter : IValueConverter
    {
        public object Convert(object value,
            Type targetType, object parameter, string language)
        {
            throw new NotImplementedException();
        }

        public object Convert(object value,
            Type targetType, object parameter, CultureInfo culture)
        {
            return value.ToString();
        }

        public object ConvertBack(object value,
            Type targetType, object parameter, string language)
        {
            throw new NotImplementedException();
        }

        public object ConvertBack(object value,
            Type targetType, object parameter, CultureInfo culture)
        {
            if (targetType == typeof(Position))
                return (Position)Enum.Parse(typeof(Position), value.ToString());
            else
                return (Palette)Enum.Parse(typeof(Palette), value.ToString());
```

```
        }
    }
}
```

5. **<Grid>** タグを編集してマークアップを次のように変更し、Sunburst にデータを提供します。

   - **XAML**

```
<Page
    x:Class="SunburstQuickStart.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:SunburstQuickStart"
    xmlns:Chart="using:C1.Xaml.Chart"
    xmlns:Xaml="using:C1.Xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">
    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Grid.Resources>
            <local:EnumToStringConverter x:Key="PaletteConverter" />
        </Grid.Resources>
        <Grid.DataContext>
            <local:SunburstViewModel />
        </Grid.DataContext>
        <Chart:C1Sunburst x:Name="sunburst"
                          Offset="0"
                          ItemsSource="{Binding HierarchicalData}"
                          Binding="Value"
                          BindingName="Year,Quarter,Month"
                          ChildItemsPath="Items"
                          ToolTipContent="{}{name}&#x000A;{y}"
                          Margin="91,0,0,356"  >
            <Chart:C1Sunburst.DataLabel>
                <Chart:PieDataLabel Position="Inside"
                                    Content="{}{name}"
                                    ConnectingLine="True"
                                    Border="True">
                </Chart:PieDataLabel>
            </Chart:C1Sunburst.DataLabel>
        </Chart:C1Sunburst>
    </Grid>
</Page>
```

手順 3：アプリケーションの実行

[F5]キーを押してアプリケーションを実行し、Sunburst チャートがどのように表示されるかを確認します.

主な機能

Sunburst コントロールは、本格的な見栄えのアプリケーションを作成できる効率的で便利なコントロールです。以下に示す多くの機能が含まれています。

- **ドーナツ Sunburst チャート**：**InnerRadius** プロパティを設定して、ドーナツ Sunburst チャートを作成できます。このプロパティのデフォルト値は 0 です。このプロパティを 0 より大きい値に設定すると、中央に穴が開き、ドーナツ Sunburst チャートが作成されます。
- **分割 Sunburst チャート**：**Offset** プロパティを設定して、分割 Sunburst チャートを作成できます。このプロパティのデフォルト値は 0 です。このプロパティを設定すると、Sunburst チャートの中心からセグメントが離され、これにより、分割 Sunburst チャートが作成されます。
- **反転 Sunburst チャート**：**Reversed** プロパティを設定して、反転した Sunburst チャートを作成できます。このプロパティは、デフォルト値として false を受け取ります。このプロパティを true に設定すると、反時計回り方向の角度で描画される反転 Sunburst チャートが作成されます。
- **開始角度**：**StartAngle** プロパティを設定して、開始角度を指定できます。このプロパティは、double 型の値を受け取ります。開始角度は、9 時の位置から時計回り方向に Sunburst グラフセグメントの描画を開始する角度（度単位）です。
- **パレット**：さまざまなカラーパレットを使用して、Sunburst チャートをさらに効果的で見栄えのするチャートにすることができます。チャートパレットを指定するには、**Palette** プロパティを設定します。このプロパティで、セグメントをレンダリングする際に使用されるデフォルトの色の配列を指定できます。このプロパティは、**Palette** 列挙に含まれる値を受け取ります。

- **凡例**: Sunburst チャートの凡例に対して、方向、位置、スタイルの設定など、さまざまなカスタマイズを行うことができます。詳細については、「凡例とタイトル」を参照してください。
- **ヘッダーとフッター**: 単純なプロパティを使用して、Sunburst チャートのヘッダーとフッターを設定およびカスタマイズできます。詳細については、「凡例とタイトル」を参照してください。
- **選択**: 選択モードを変更したり、選択されているセグメントの位置や外観をカスタマイズできます。詳細については、「選択」を参照してください。
- **Data Labels**: Add, style, format, set the position of data labels and manage the overlapped data labels on the chart. For more information, refer Data Labels.

凡例とタイトル

凡例

凡例は、系列のエントリを名前と定義済みの記号で表示します。Sunburst では、次に示すように、凡例に対してさまざまなカスタマイズを行うことができます。

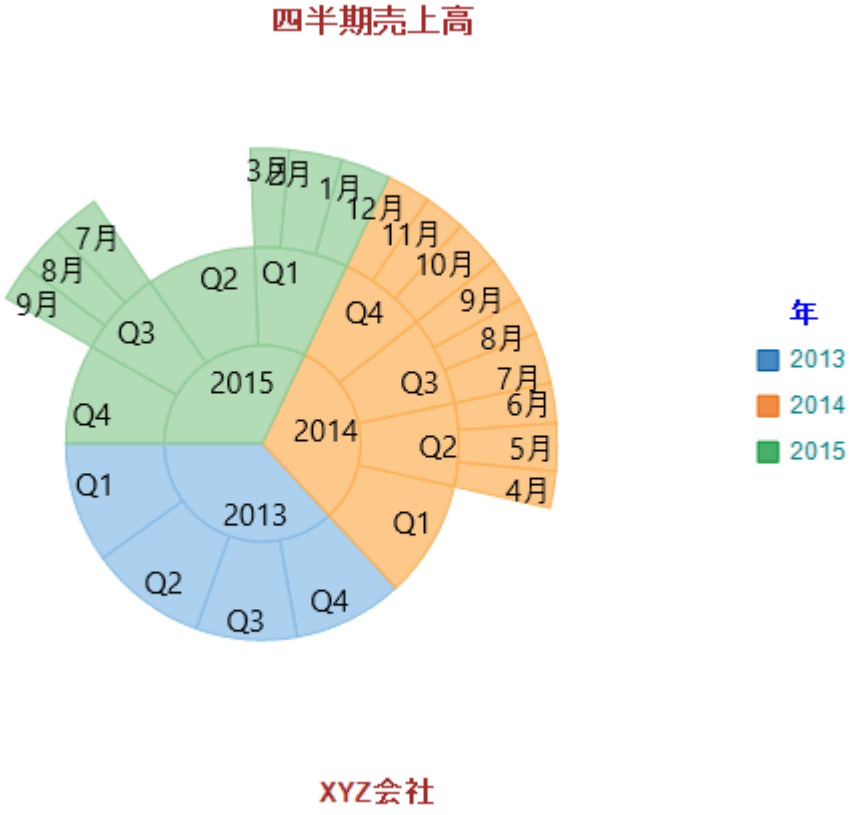
- **方向**: **FlexChartBase** クラスで提供されている **LegendOrientation** プロパティを使用して、凡例の方向を、水平、垂直、または自動に設定できます。このプロパティは、**Orientation** 列挙に含まれる値のいずれかに設定できます。
- **位置**: **Position** 列挙に含まれる値を受け取る **LegendPosition** プロパティを使用して、凡例の位置を、上、下、左、右、または自動配置に設定できます。Position プロパティを None に設定すると、凡例は非表示になります。
- **スタイル設定**: **LegendStyle** プロパティからアクセスできるスタイル設定 プロパティを使用して、ストローク色の設定、フォントの変更など、凡例の外観全体をカスタマイズできます。スタイル設定プロパティ **Stroke**、**FontSize**、および **FontStyle** は、**ChartStyle** クラスで提供されています。
- **タイトルおよびタイトルのスタイル設定**: 凡例タイトルは、文字列を受け取る **LegendTitle** プロパティを使用して指定できます。タイトルを設定したら、**LegendTitleStyle** プロパティを使用してタイトルのスタイルを設定できます。このプロパティから、ChartStyle クラスのカスタマイズプロパティにアクセスできます。

ヘッダーとフッター

ヘッダーとフッターは、チャートの上と下に表示される説明テキストで、チャートデータ全体に関する情報を提供します。Sunburst チャートのヘッダーとフッターには、FlexChartBase クラスの **Header** プロパティと **Footer** プロパティをそれぞれ使用してアクセスできます。ヘッダーとフッターに対しては、以下のカスタマイズが可能です。

- **フォント**: ヘッダーとフッターのフォントファミリ、フォントサイズ、フォントスタイルを変更できます。それには、FlexChartBase クラスの**HeaderStyle** プロパティまたは **FooterStyle** プロパティからアクセスできる ChartStyle クラスのさまざまなフォントプロパティを使用します。
- **ストローク**: Stroke プロパティを使用してタイトルのストロークを設定し、見栄えをさらによくすることができます。

次の図に、凡例とタイトルを設定してカスタマイズした Sunburst チャートを示します。



次のコードスニペットは、凡例およびタイトルをカスタマイズするためにそれぞれのプロパティを設定する方法を示しています。このコードでは、「クイックスタート」で作成したサンプルを使用します。

XAML

- **タブキャプション**

```
<Chart:C1Sunburst x:Name="sunburst"
                  SelectionMode="Point"
                  SelectedItemOffset="0.1"
                  SelectedItemPosition="Top"
                  LegendPosition="None"
                  Header="四半期売上高"
                  Footer="XYZ会社"
                  InnerRadius="0.1"
                  Reversed="True"
                  Palette="Dark"
                  Offset="0.1"
                  ItemsSource="{Binding HierarchicalData}"
                  Binding="Value"
                  BindingName="Year,Quarter,Month"
                  ChildItemsPath="Items"
                  ToolTipContent="{}{name}&#x000A;{y}"
                  Margin="-376,0,0,0"
                  LegendTitle="年"  >
    <Chart:C1Sunburst.LegendTitleStyle>
        <Chart:ChartStyle FontFamily="Arial"
                          FontSize="10"
                          FontStretch="Normal"
                          FontWeight="Bold"/>
    </Chart:C1Sunburst.LegendTitleStyle>
    <Chart:C1Sunburst.HeaderStyle>
        <Chart:ChartStyle FontFamily="Arial"
```

```
                            FontSize="12"
                            FontStretch="Normal"
                            FontWeight="Bold"/>
    </Chart:C1Sunburst.HeaderStyle>
    <Chart:C1Sunburst.FooterStyle>
        <Chart:ChartStyle FontFamily="Arial"
                            FontSize="10"
                            FontStretch="Normal"
                            FontWeight="Bold"/>
    </Chart:C1Sunburst.FooterStyle>
    <Chart:C1Sunburst.DataLabel>
        <Chart:PieDataLabel Position="Inside"
                            Content="{}{name}"
                            ConnectingLine="True"
                            Border="True">
        </Chart:PieDataLabel>
    </Chart:C1Sunburst.DataLabel>
</Chart:C1Sunburst>
```

コード

C# copyCode

```
// 凡例の位置を設定します。
sunburst.LegendPosition = C1.Chart.Position.None;

// ヘッダーを設定します。
sunburst.Header = "四半期売上高";

// フッターを設定します。
sunburst.Footer = "XYZ会社";
```

VB copyCode

```
' 凡例の位置を設定します。
sunburst.LegendPosition = C1.Chart.Position.None

' ヘッダーを設定します。
sunburst.Header = "四半期売上高"

' フッターを設定します。
sunburst.Footer = "XYZ会社"
```

選択

Sunburst グラフセグメントをクリックして、データポイントを選択できます。**FlexChartBase** クラスで提供されている **SelectionMode** プロパティを **ChartSelectionMode** 列挙に含まれる次の値のいずれかに設定できます。

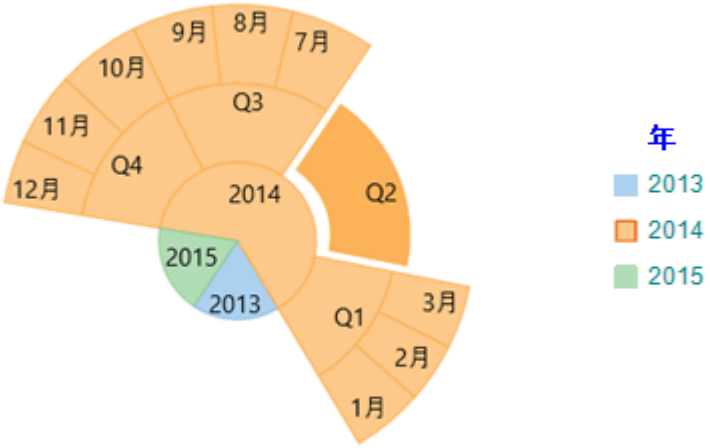
- **None**（デフォルト）：選択は無効です。
- **Point**：ポイントが選択されます。

選択をカスタマイズするには、**C1FlexPie** クラスで提供されている **SelectedItemOffset** プロパティと **SelectedIemPosition**プロパティを使用します。SelectedItemOffset プロパティを使用すると、選択されている Sunburst グラフセグメントをコントロールの中心から離す距離を指定できます。また、SelectedItemPosition プロパティを使用すると、選択されている Sunburst グラフセグメントの位置を指定できます。SelectedItemPosition プロパティは、Position 列挙に含まれる値

を受け取ります。このプロパティを「None」以外の設定すると、項目が選択されたときに円が回転します。

さらに、**C1Sunburst** クラスで提供されている **ChildItemsPath** プロパティを使用すると、階層化データ内の子項目を含むコレクションを指定できます。また、FlexChartBaseクラスで提供されている **SelectionStyle** プロパティを使用すると、**ChartStyle** クラスで提供されるプロパティにアクセスして、Sunburst チャートのスタイルを設定できます。

次の図に、データポイントが選択されている Sunburst チャートを示します。



次のコードスニペットでは、上記のプロパティを設定しています。

XAML

- **タブキャプション**

```
<Chart:C1Sunburst x:Name="sunburst"
                  SelectionMode="Point"
                  SelectedItemOffset="0.1"
                  SelectedItemPosition="Top"
                  LegendPosition="None"
                  Header="四半期売上高"
                  Footer="XYZ会社"
                  InnerRadius="0.1"
                  Reversed="True"
                  Palette="Dark"
                  Offset="0.1"
                  ItemsSource="{Binding HierarchicalData}"
                  Binding="Value"
                  BindingName="Year,Quarter,Month"
                  ChildItemsPath="Items"
                  ToolTipContent="{}{name}&#x000A;{y}"
                  Margin="-376,0,0,0"
                  LegendTitle="年"  >
    <Chart:C1Sunburst.LegendTitleStyle>
        <Chart:ChartStyle FontFamily="Arial"
                          FontSize="10"
                          FontStretch="Normal"
                          FontWeight="Bold"/>
    </Chart:C1Sunburst.LegendTitleStyle>
    <Chart:C1Sunburst.HeaderStyle>
```

```
        <Chart:ChartStyle FontFamily="Arial"
                          FontSize="12"
                          FontStretch="Normal"
                          FontWeight="Bold"/>
    </Chart:C1Sunburst.HeaderStyle>
    <Chart:C1Sunburst.FooterStyle>
        <Chart:ChartStyle FontFamily="Arial"
                          FontSize="10"
                          FontStretch="Normal"
                          FontWeight="Bold"/>
    </Chart:C1Sunburst.FooterStyle>
    <Chart:C1Sunburst.DataLabel>
        <Chart:PieDataLabel Position="Inside"
                            Content="{}{name}"
                            ConnectingLine="True"
                            Border="True">
        </Chart:PieDataLabel>
    </Chart:C1Sunburst.DataLabel>
</Chart:C1Sunburst>
```

コード

C# copyCode

```
// SelectionModeプロパティを設定します。
sunburst.SelectionMode = C1.Chart.ChartSelectionMode.Point;

// SelectedItemOffsetプロパティを設定します。
sunburst.SelectedItemOffset = 0.1;

// SelectedItemPositionプロパティを設定します。
sunburst.SelectedItemPosition = C1.Chart.Position.Top;
```

VB copyCode

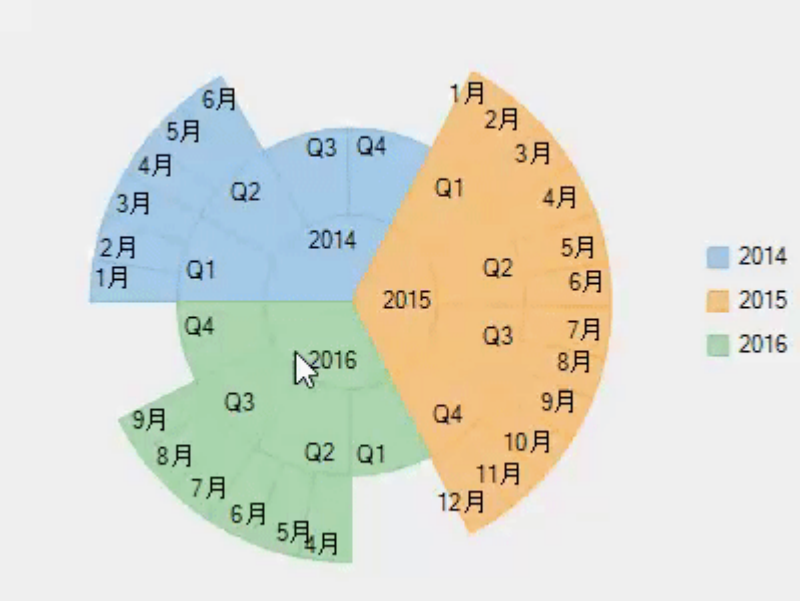
```
' SelectionModeプロパティを設定します。
sunburst.SelectionMode = C1.Chart.ChartSelectionMode.Point

' SelectedItemOffsetプロパティを設定します。
sunburst.SelectedItemOffset = 0.1

' SelectedItemPositionプロパティを設定します。
sunburst.SelectedItemPosition = C1.Chart.Position.Top
```

ドリルダウン

データをドリルダウンして詳細まで掘り下げ、Sunburst チャートのデータ階層の下位レベルにアクセスすると、分析に極めて便利です。Sunburst チャートには、実行時のデータのドリルダウン/ドリルアップ機能を提供する **Drilldown** プロパティがあります。

エンドユーザーは、Sunburst チャートで目的のスライスをクリックするだけで、特定のデータ項目に注目してドリルダウンすることができます。階層を上に戻るには、プロット領域を右クリックするだけです。

次の gif 画像では、ドリルダウンの例として、クリックされた Sunburst グラフセグメントのデータポイントを表示しています。



Sunburst のドリルダウン機能は、Sunburst グラフセグメントの選択が無効の場合、つまり **SelectionMode** プロパティが **None** に設定されている場合にのみ機能します。選択の詳細については、Sunburst での選択を参照してください。

データラベル

Data labels provide additional information about the data points. These labels make a chart easier to understand because they show details about a slice in the pie.

To understand the working of data labels in FlexPie chart, refer to the following sections.

Adding and Positioning Data Labels
: Learn how to add data labels and set their position on the chart.

Formatting Data Labels
: Learn how to perform styling and formatting of data labels.

Managing Overlapped Data Labels
: Learn how to manage overlapping data labels in FlexPie chart.

データラベルの追加と配置

With Sunburst chart, you can configure the arrangement and display properties for data labels depending on what suits your needs the best. By default, the data labels are not displayed on the chart, however, you can enable them by setting the Position and Content properties of DataLabel class.

The example code below uses the **Position** and **Content** properties to enable data labels and set their position.

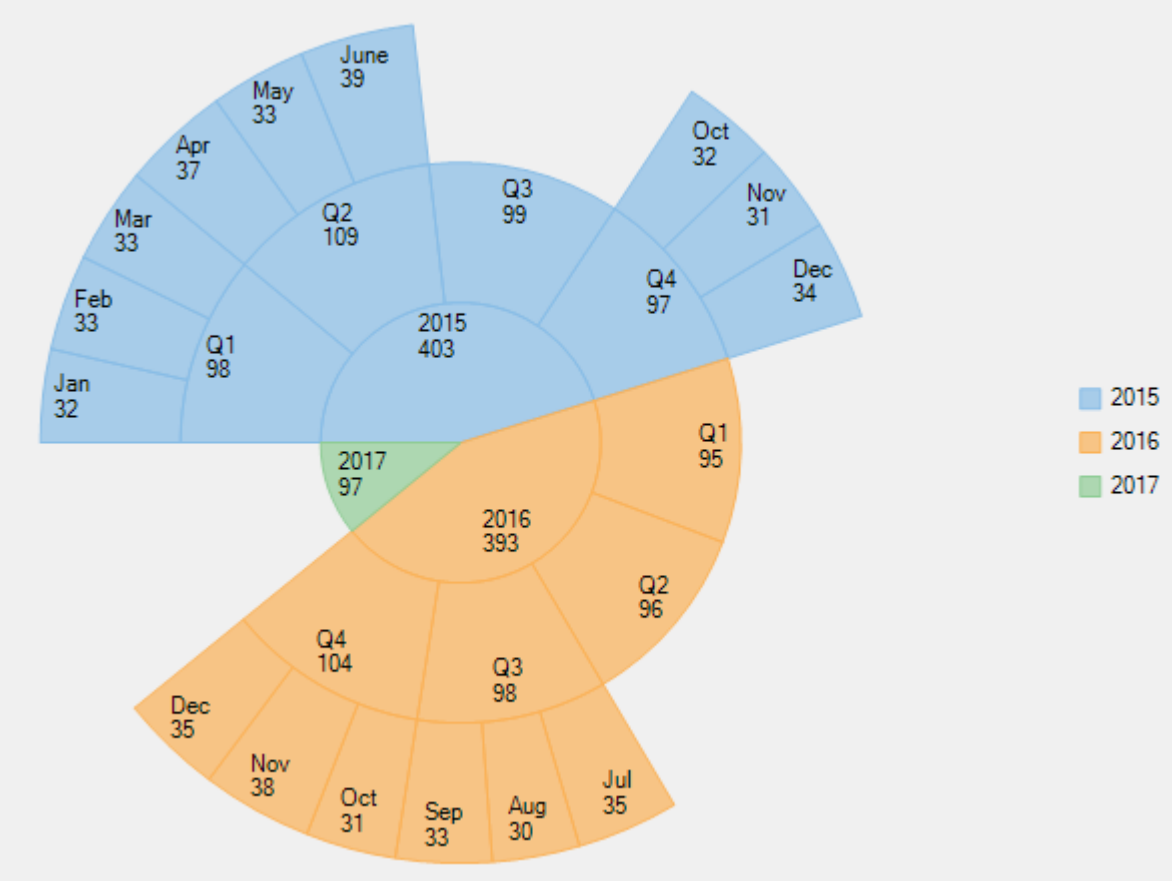
XAML

```
<c1:C1Sunburst.DataLabel>
<c1:PieDataLabel Position="Inside"
Content="{}{name}">
</c1:PieDataLabel>
</c1:C1Sunburst.DataLabel>
```

Code

HTML

```
sunburst.DataLabel.Content = "{Name}{value}";sunburst.DataLabel.Position =
C1.Chart.PieLabelPosition.Inside;
```



データラベルの書式設定

Sunburst provides various options to format data labels according to your requirements. You can use connecting lines to connect the data labels, set and style borders of data labels, and customize the appearance of data labels.

The topic comprises of three sections:

- **Setting and Styling Borders**
- **Connecting DataLabels to Data Points**
- **Modifying Appearance**

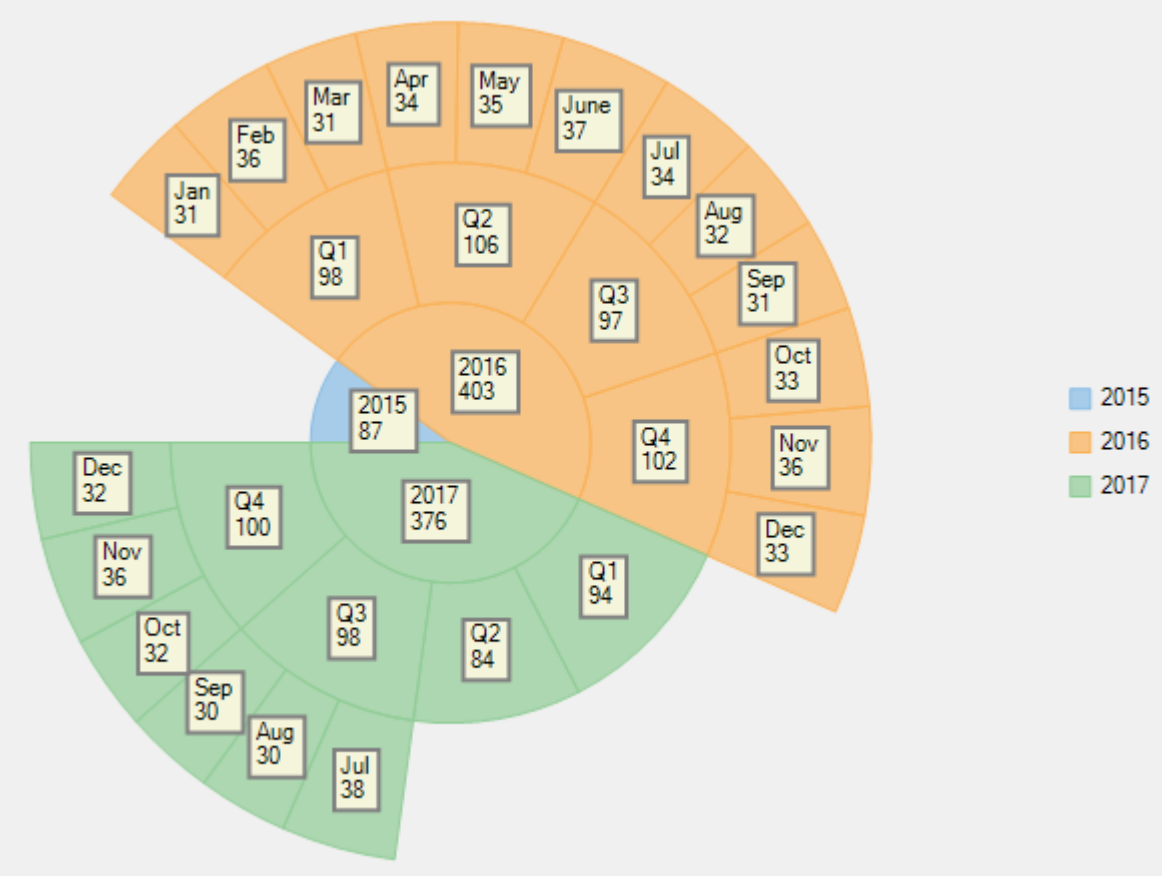
Setting and Styling Borders

To add and style borders to Sunburst data labels, set the **Border** and **BorderStyle** properties provided by **DataLabelBase** class.

Use the following code snippet to add borders to data labels of Sunburst.

C#

```
// 境界線を有効にします
sunburst.DataLabel.Border =
true;sunburst.DataLabel.BorderStyle.StrokeThickness = 2;
```



Connecting DataLabels to Data Points

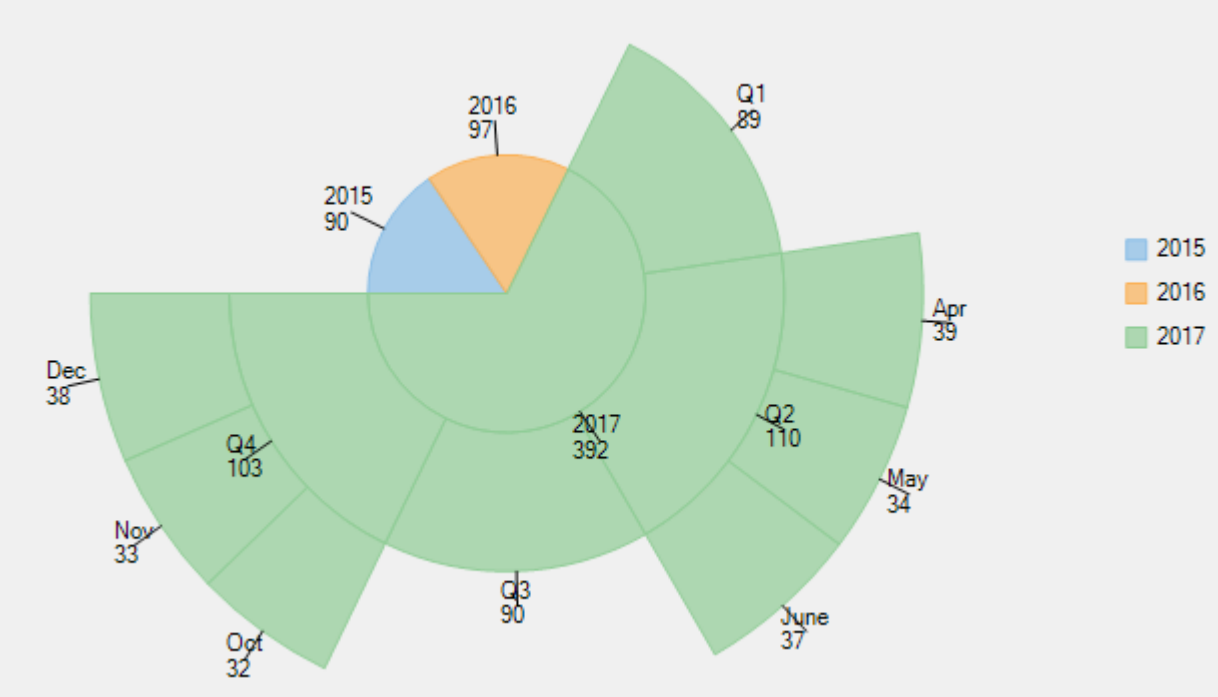
In case the data labels are placed away from the data points, you can connect them using connecting lines.

To enable connecting lines in Sunburst chart, you need to use the **ConnectingLine** property.

Use the following code snippet to set the connecting lines.

C#

```
// 接続線を有効にします
sunburst.DataLabel.ConnectingLine = true;
```



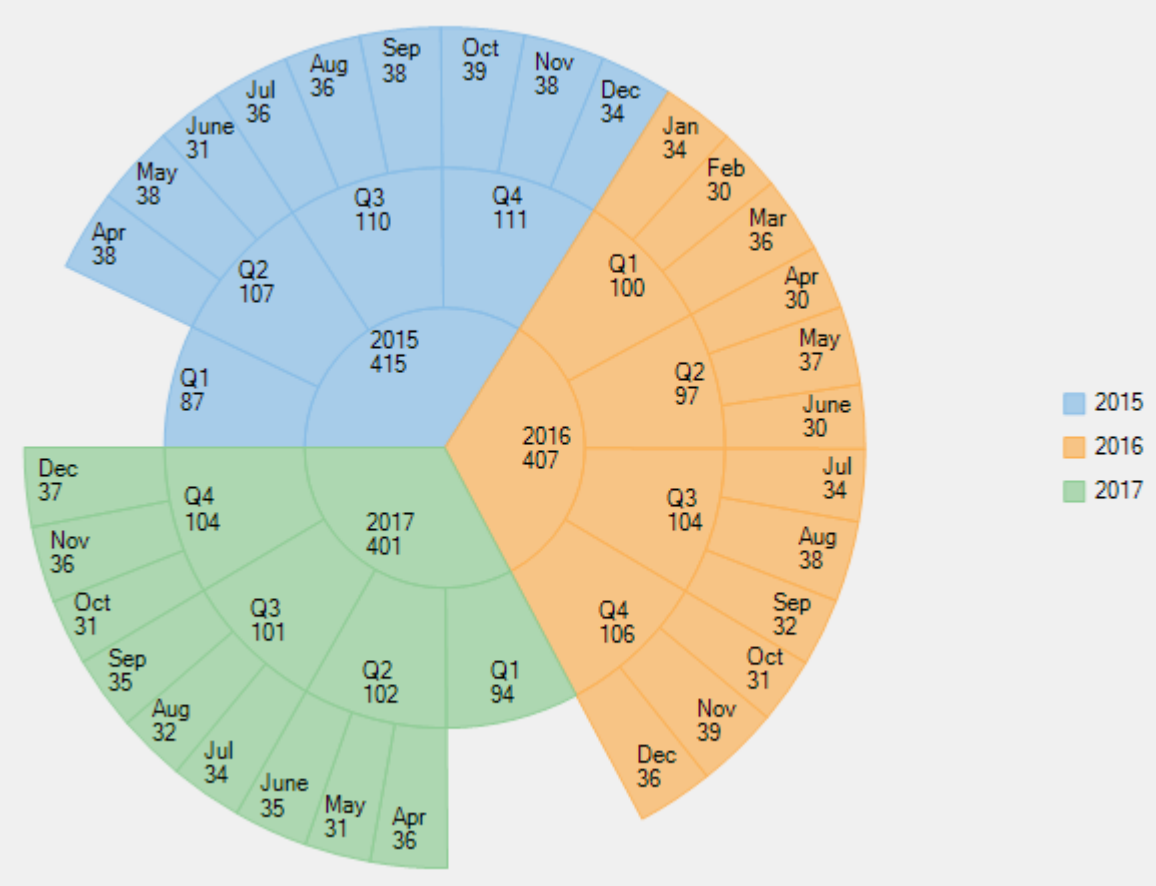
Modifying Appearance

Sunburst includes various styling options, to enhance the clarity of data labels. To modify the appearance of Sunburst chart, you need to use the **Style** property. It allows you to modify the font family, fill color, use stroke brush for data labels, set width for stroke brush and more.

In the example code, we have modified the font used in the chart and set the stroke width property. Use the following code snippet to modify the appearance of the chart.

C#

```
// データラベルの外観を変更します
sunburst.DataLabel.Style.FontFamily = new
FontFamily("GenericSansSerif");sunburst.DataLabel.Style.StrokeThickness
= 2;
```



重なったデータラベルの管理

A common issue pertaining to charts is overlapping of data labels that represent data points. In most cases, overlapping occurs due to long text in data labels or large numbers of data points.

To manage overlapped data labels in Sunburst chart, you can make use of **Overlapping** property provided by **PieDataLabel** class. The Overlapping property accepts the following values from the **PieLabelOverlapping** enumeration.

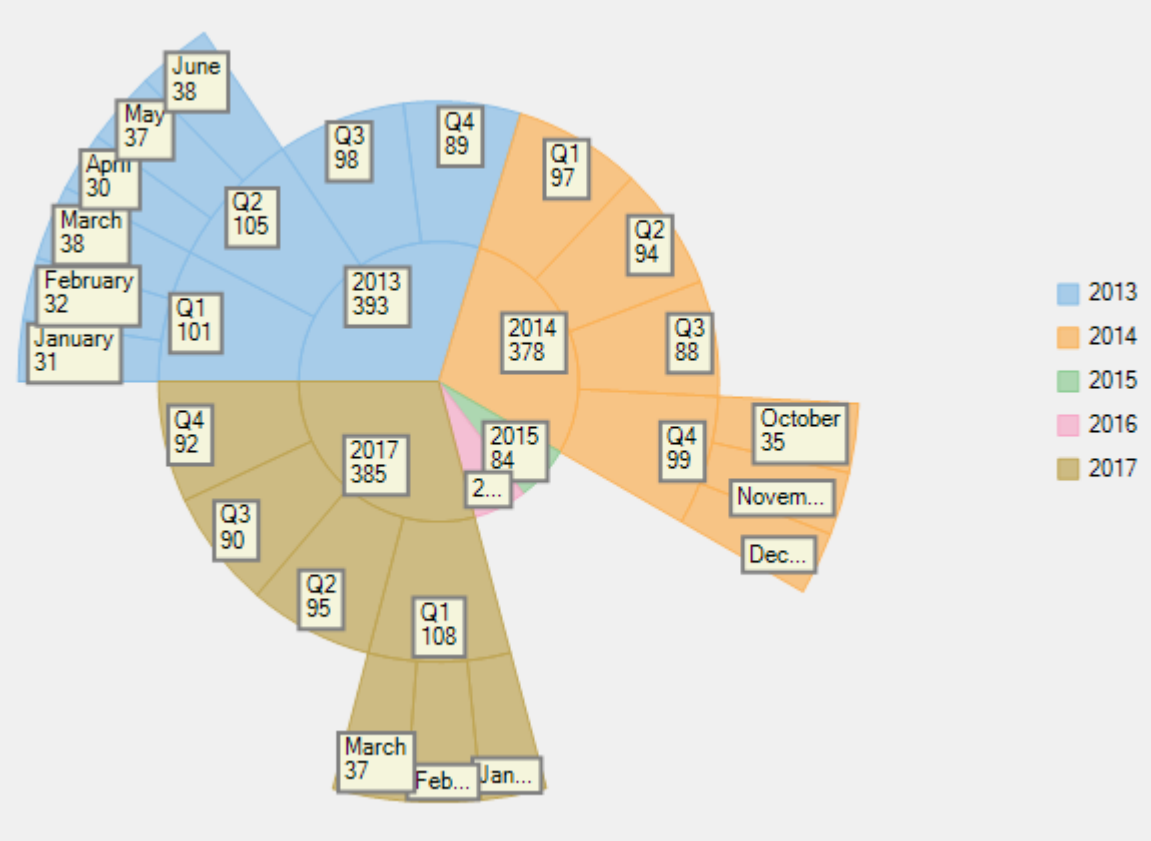
Enumeration	Description
PieLabelOverlapping.Default	Show all labels including the overlapping ones.
PieLabelOverlapping.Hide	Hides the overlapping labels, if its content is larger than the corresponding pie segment.
PieLabelOverlapping.Trim	Trim overlapping data labels, if its width is larger than the corresponding pie segment.

Use the following code to manage overlapping data labels.

C#

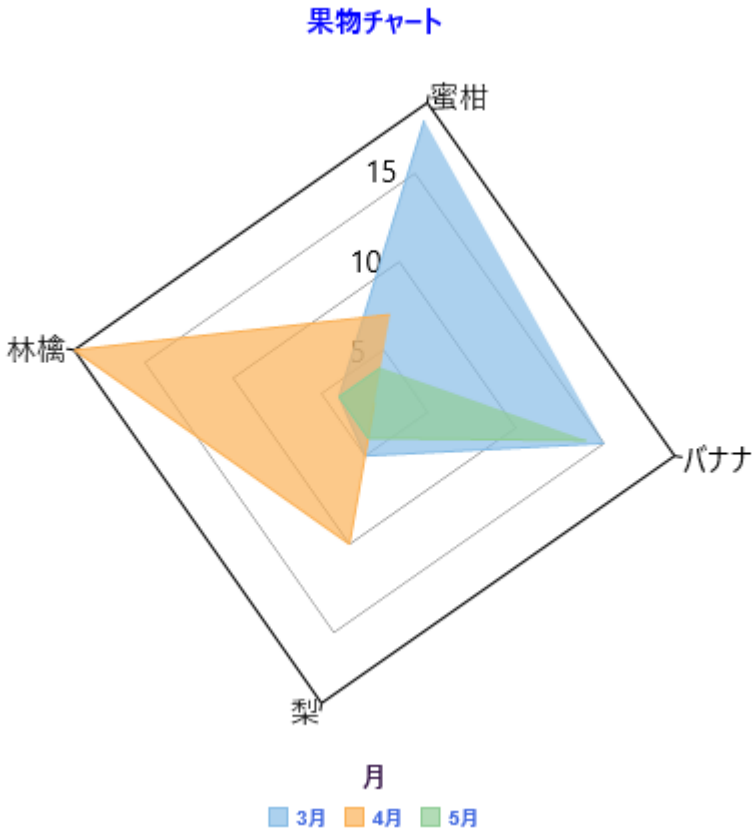
```
// Overlapping プロパティを設定します
sunburst.DataLabel.Overlapping = C1.Chart.PieLabelOverlapping.Trim;
```

The following image shows how Sunburst appears after setting the Overlapping property.



FlexRadar

FlexRadar はレーダーチャートですが、その外観から、ポーラチャート、スターチャート、クモの巣チャート、スパイダーチャートなどとも呼ばれます。このチャートでは、中心から外側の円まで伸びる軸ごとに個別のカテゴリの値がプロットされます。すべての軸は、等間隔の放射状に配置され、すべての軸で同じスケールが使用されます。各カテゴリの値は個別の軸にプロットされ、すべての値は多角形の形に接続されます。FlexRadar のビジネスアプリケーションとしては、従業員のスキル分析、製品比較などによく使用されます。X 値が角度値（度単位）を指定する数値なら、FlexRadar コントロールがポーラチャートになることは重要です。



FlexRadar の詳細については、次の各リンクをクリックしてください。

- クイックスタート
- 主要な機能
- チャートタイプ
- 凡例とタイトル

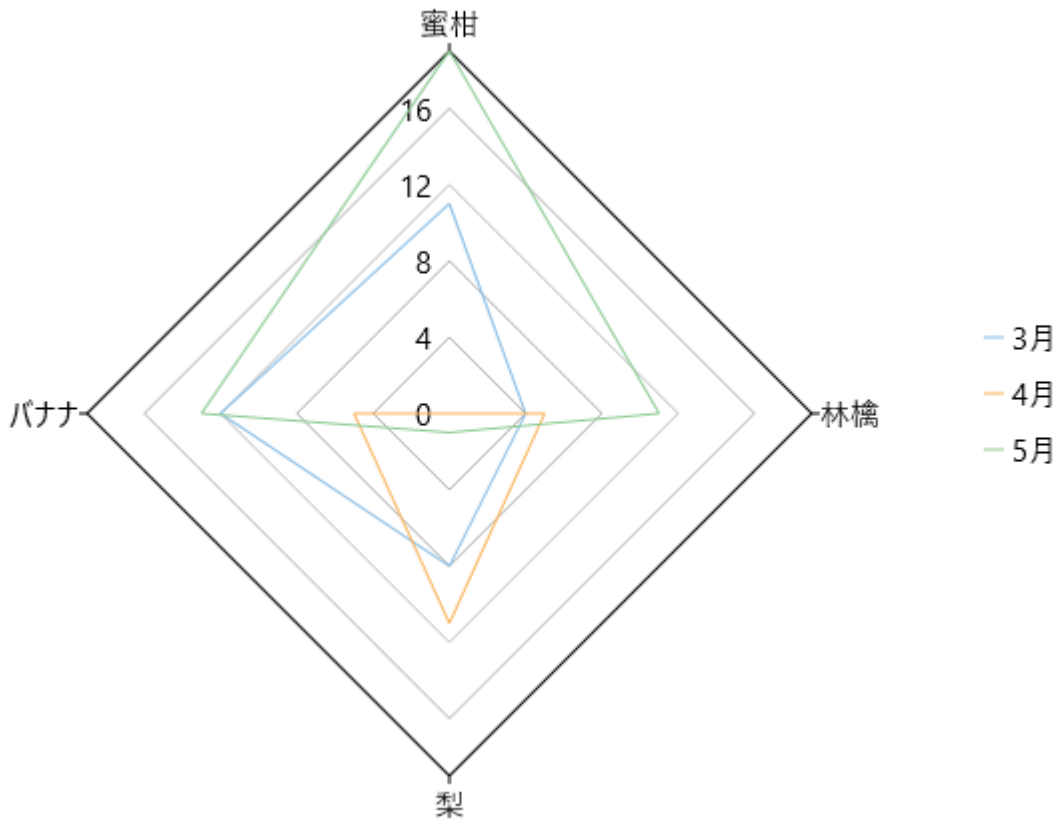
クイックスタート

このクイックスタートでは、Visual Studio で単純な FlexRadar アプリケーションを作成して実行する手順を説明します。

FlexRadar の使用をすぐに開始し、アプリケーション実行時にどのように表示されるかを確認するには、次の手順に従います。

1. **アプリケーションへの FlexRadar の追加**
2. **データソースへの FlexRadar の連結**
3. **アプリケーションの実行**

次の図に、上記の手順を完了した後に、基本的な FlexRadar がどのように表示されるかを示します。



手順 1:アプリケーションへの FlexRadar の追加

1. Visual Studio で **UWP アプリケーション**を作成します。
2. **C1FlexRadar** コントロールを**ツールボックス**から MainPage にドラッグアンドドロップします。
   次の .dll ファイルが自動的にアプリケーションに追加されます。
   - **C1.UWP.dll**
   - **C1.UWP.DX.dll**
   - **C1.UWP.FlexChart.dll**

   <Grid></Grid> タグ内の XAML マークアップは次のコードのようになります。

   - **XAML**

```
<Chart:C1FlexRadar HorizontalAlignment="Left"
                   Height="100"
                   Margin="0"
                   VerticalAlignment="Top"
                   Width="100"/>
```

手順 2:データソースへの FlexRadar の連結

この手順では、まず、ある年の連続する 3 か月間の果物の売上データを生成する DataCreator クラスを作成します。次に、**FlexChartBase** クラスで提供される **ItemsSource** プロパティを使用して、作成したデータソースに FlexRadar を連結します。さらに、FlexRadar の X 値を含むフィールドと Y 値を含むフィールドを、それぞれ **BindingX** プロパティと **Binding** プロパティを使用して指定します。

1. クラス **DataCreator** を追加し、次のコードを追加します。

   - **Visual Basic**

```
Class DataCreator
    Public Shared Function CreateFruit() As List(Of FruitDataItem)
```

```
        Dim fruits = New String() {"蜜柑", "林檎", "梨", "バナナ"}
        Dim count = fruits.Length
        Dim result = New List(Of FruitDataItem)()
        Dim rnd = New Random()

        For i As Object = 0 To count - 1
            result.Add(New FruitDataItem() With {
                .Fruit = fruits(i),
                .March = rnd.[Next](20),
                .April = rnd.[Next](20),
                .May = rnd.[Next](20),
                .Size = rnd.[Next](5)
            })
        Next
        Return result
    End Function
End Class
Public Class FruitDataItem
    Public Property Fruit() As String
        Get
            Return m_Fruit
        End Get
        Set
            m_Fruit = Value
        End Set
    End Property
    Private m_Fruit As String
    Public Property March() As Double
        Get
            Return m_March
        End Get
        Set
            m_March = Value
        End Set
    End Property
    Private m_March As Double
    Public Property April() As Double
        Get
            Return m_April
        End Get
        Set
            m_April = Value
        End Set
    End Property
    Private m_April As Double
    Public Property May() As Double
        Get
            Return m_May
        End Get
        Set
            m_May = Value
        End Set
    End Property
    Private m_May As Double
    Public Property Size() As Integer
        Get
            Return m_Size
        End Get
        Set
            m_Size = Value
        End Set
    End Property
    Private m_Size As Integer
```

```
End Class
Public Class DataPoint
    Public Property XVals() As Double
        Get
            Return m_XVals
        End Get
        Set
            m_XVals = Value
        End Set
    End Property
    Private m_XVals As Double
    Public Property YVals() As Double
        Get
            Return m_YVals
        End Get
        Set
            m_YVals = Value
        End Set
    End Property
    Private m_YVals As Double
End Class
```

- **C#**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FlexRadarQuickStart
{
    class DataCreator
    {
        public static List<FruitDataItem> CreateFruit()
        {
        var fruits = new string[] { "蜜柑", "林檎", "梨", "バナナ" };
            var count = fruits.Length;
            var result = new List<FruitDataItem>();
            var rnd = new Random();

            for (var i = 0; i < count; i++)
                result.Add(new FruitDataItem()
                {
                    Fruit = fruits[i],
                    March = rnd.Next(20),
                    April = rnd.Next(20),
                    May = rnd.Next(20),
                    Size = rnd.Next(5),
                });
            return result;
        }
    }
    public class FruitDataItem
    {
        public string Fruit { get; set; }
        public double March { get; set; }
        public double April { get; set; }
        public double May { get; set; }
        public int Size { get; set; }
    }
    public class DataPoint
    {
        public double XVals { get; set; }
        public double YVals { get; set; }
```

```
    }
}
```

2. XAML マークアップを編集して、FlexRadar にデータを提供します。

   - **XAML**

```
<Page
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="using:FlexRadarQuickStart"
xmlns:Chart="using:C1.Xaml.Chart"
x:Class="FlexRadarQuickStart.MainPage"
DataContext="{Binding RelativeSource={RelativeSource Mode=Self}}">

<Grid>
    <Chart:C1FlexRadar ItemsSource="{Binding DataContext.Data}"
                       BindingX="Fruit"
                       Margin="0,220,20,130">
        <Chart:C1FlexRadar.Series>
            <Chart:Series SeriesName="3月"
                          Binding="March"></Chart:Series>
            <Chart:Series SeriesName="4月"
                          Binding="April"></Chart:Series>
            <Chart:Series SeriesName="5月"
                          Binding="May"></Chart:Series>
        </Chart:C1FlexRadar.Series>
    </Chart:C1FlexRadar>
</Grid>

</Page>
```

3. **コードビュー**に切り替えて、次のコードを追加します。

   - **Visual Basic**

```
Partial Public NotInheritable Class MainPage
    Inherits Page
    Private _fruits As List(Of FruitDataItem)

    Public Sub New()
        Me.InitializeComponent()
    End Sub

    Public ReadOnly Property Data() As List(Of FruitDataItem)
        Get
            If _fruits Is Nothing Then
                _fruits = DataCreator.CreateFruit()
            End If

            Return _fruits
        End Get
    End Property
End Class
```

   - **C#**

```
using System.Collections.Generic;
using Windows.UI.Xaml.Controls;

namespace FlexRadarQuickStart
{
    public sealed partial class MainPage : Page
    {
        List<FruitDataItem> _fruits;
```

```
            public MainPage()
            {
                this.InitializeComponent();
            }

            public List<FruitDataItem> Data
            {
                get
                {
                    if (_fruits == null)
                    {
                        _fruits = DataCreator.CreateFruit();
                    }

                    return _fruits;
                }
            }
        }
    }
```

手順 3:アプリケーションの実行

[F5]キーを押してアプリケーションを実行し、出力を確認します。

主要な機能

FlexRadar の主要な機能をいくつか示します。

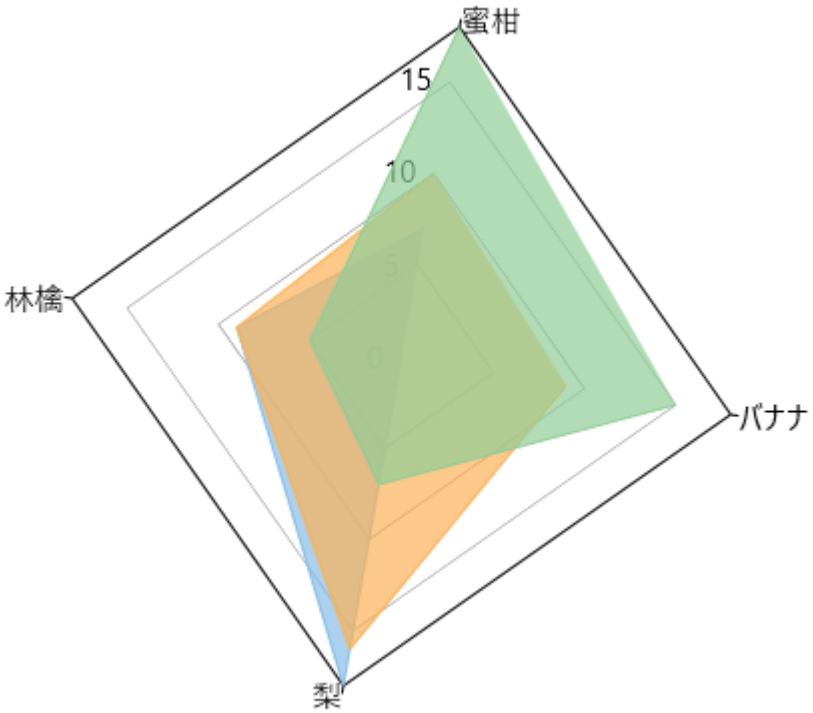
- **反転 FlexRadar**:**Reversed** プロパティを true に設定して、反転した FlexRadar を作成できます。反転した FlexRadar では、プロットの方向が逆になります。
- **開始角度**:**C1FlexRadar** クラスで提供される **StartAngle** プロパティを double 型の値に設定することで、FlexRadar の開始角度を設定できます。開始角度は、時計回り方向に放射軸の描画を開始する角度(度単位)です。
- **ヘッダーとフッター**:単純なプロパティを使用して、FlexRadar のヘッダーとフッターを設定およびカスタマイズできます。詳細については、「凡例とタイトル」を参照してください。
- **凡例**:FlexRadar の凡例に対して、方向、位置、スタイルの設定など、さまざまなカスタマイズを行うことができます。詳細については、「凡例とタイトル」を参照してください。
- **チャートタイプ**:FlexRadar に含まれるさまざまなチャートタイプを使用して、データを視覚化できます。詳細については、「チャートタイプ」を参照してください。

チャートタイプ

FlexRadar では、データ視覚化ニーズに応じてさまざまなチャートタイプを使用することができます。Area から Scatter までさまざまなチャートタイプを使用して、FlexRadar のデータ内の領域を色やパターンで塗りつぶして表示できます。**C1FlexRadar** の **ChartType** プロパティを **RadarChartType** 列挙に含まれる次の値のいずれかに設定することで、FlexRadar のさまざまなチャートタイプを設定できます。

- **Area**:線の下の領域を色で塗りつぶして表示します。
- **Line**:一定期間のトレンドまたはカテゴリ間のトレンドを表示します。
- **LineSymbols**:各データポイントにシンボルが付いた折れ線グラフを表示します。
- **Scatter**:X 座標と Y 座標を使用して、データに含まれるパターンを表示します。

次の図に、チャートタイプを Area に設定した FlexRadar を示します。



次のコードスニペットでは、ChartType プロパティを設定しています。

XAML

```
<Chart:C1FlexRadar Header="果物チャート"
                   LegendOrientation="Horizontal"
                   LegendPosition="Bottom"
                   ChartType="Area"
                   StartAngle="10"
                   Reversed="True"
                   ItemsSource="{Binding DataContext.Data}"
                   BindingX="Fruit"
                   Margin="0,220,20,130" LegendTitle="月">
    <Chart:C1FlexRadar.LegendStyle>
        <Chart:ChartStyle FontFamily="Arial"
                          FontSize="10"
                          FontWeight="Bold"
                          Stroke="RoyalBlue"/>
    </Chart:C1FlexRadar.LegendStyle>
    <Chart:C1FlexRadar.LegendTitleStyle>
        <Chart:ChartStyle FontFamily="Arial"
                          FontSize="13"
                          FontWeight="Bold"
                          Stroke="#FF371649"/>
    </Chart:C1FlexRadar.LegendTitleStyle>
    <Chart:C1FlexRadar.HeaderStyle>
        <Chart:ChartStyle FontFamily="Arial"
                          FontSize="14"
                          FontWeight="Bold"
                          Stroke="Blue"/>
    </Chart:C1FlexRadar.HeaderStyle>
    <Chart:C1FlexRadar.Series>
        <Chart:Series SeriesName="3月"
                      Binding="March"></Chart:Series>
        <Chart:Series SeriesName="4月"
                      Binding="April"></Chart:Series>
```

```
        <Chart:Series SeriesName="5月"
                      Binding="May"></Chart:Series>
    </Chart:C1FlexRadar.Series>
</Chart:C1FlexRadar>
```

コード

C# copyCode

```
// FlexRadarのチャートタイプを設定します。
flexRadar.ChartType = C1.Chart.RadarChartType.Area;
```

VB copyCode

```
' FlexRadarのチャートタイプを設定します。
flexRadar.ChartType = C1.Chart.RadarChartType.Area
```

凡例とタイトル

凡例

凡例は、系列のエントリを名前と定義済みの記号で表示します。FlexRadar では、次に示すように、凡例に対してさまざまなカスタマイズを行うことができます。

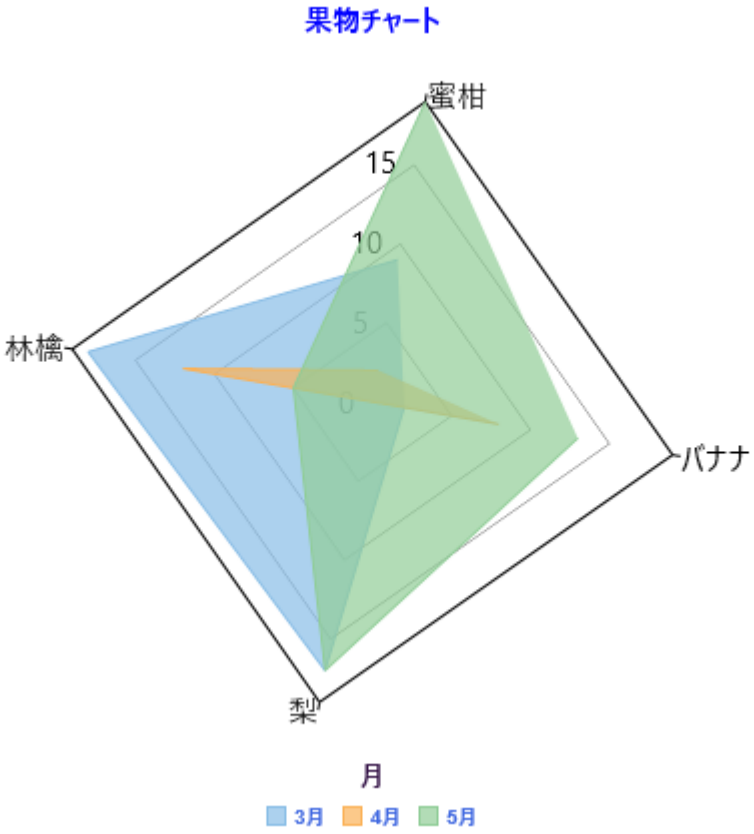
- **方向**:**FlexChartBase** クラスで提供されている **LegendOrientation** プロパティを使用して、凡例の方向を、水平、垂直、または自動に設定できます。このプロパティは、**Orientation** 列挙に含まれる値のいずれかに設定できます。
- **位置**:**Position** 列挙に含まれる値を受け取る **LegendPosition** プロパティを使用して、凡例の位置を、上、下、左、右、または自動配置に設定できます。Position プロパティを None に設定すると、凡例は非表示になります。
- **スタイル設定**:**LegendStyle** プロパティからアクセスできるスタイル設定プロパティを使用して、ストローク色の設定、フォントの変更など、凡例の外観全体をカスタマイズできます。スタイル設定プロパティ **Stroke**、**FontSize**、および **FontStyle** は、**ChartStyle** クラスで提供されています。
- **タイトルおよびタイトルのスタイル設定**:凡例タイトルは、文字列を受け取る **LegendTitle** プロパティを使用して指定できます。タイトルを設定したら、**LegendTitleStyle** プロパティを使用してタイトルのスタイルを設定できます。このプロパティから、ChartStyle クラスのカスタマイズプロパティにアクセスできます。

ヘッダーとフッター

ヘッダーとフッターは、チャートの上と下に表示される説明テキストで、チャートデータ全体に関する情報を提供します。FlexRadar のヘッダーとフッターには、FlexChartBase クラスの **Header** プロパティと **Footer** プロパティを使用してそれぞれアクセスできます。ヘッダーとフッターに対しては、以下のカスタマイズが可能です。

- **フォント**:ヘッダーとフッターのフォントファミリ、フォントサイズ、フォントスタイルを変更できます。それには、FlexChartBase クラスの **HeaderStyle** プロパティまたは **FooterStyle** プロパティからアクセスできる ChartStyle クラスのさまざまなフォントプロパティを使用します。
- **ストローク**:**Stroke** プロパティを使用してタイトルのストロークを設定し、見栄えをさらによくすることができます。

次の図に、凡例とタイトルを設定した FlexRadar を示します。



次のコードスニペットは、さまざまなプロパティを設定する方法を示します。

XAML

```
<Chart:C1FlexRadar Header="果物チャート"
                   LegendOrientation="Horizontal"
                   LegendPosition="Bottom"
                   ChartType="Area"
                   StartAngle="10"
                   Reversed="True"
                   ItemsSource="{Binding DataContext.Data}"
                   BindingX="Fruit"
                   Margin="0,220,20,130" LegendTitle="月">
    <Chart:C1FlexRadar.LegendStyle>
        <Chart:ChartStyle FontFamily="Arial"
                          FontSize="10"
                          FontWeight="Bold"
                          Stroke="RoyalBlue"/>
    </Chart:C1FlexRadar.LegendStyle>
    <Chart:C1FlexRadar.LegendTitleStyle>
        <Chart:ChartStyle FontFamily="Arial"
                          FontSize="13"
                          FontWeight="Bold"
                          Stroke="#FF371649"/>
    </Chart:C1FlexRadar.LegendTitleStyle>
    <Chart:C1FlexRadar.HeaderStyle>
        <Chart:ChartStyle FontFamily="Arial"
                          FontSize="14"
                          FontWeight="Bold"
                          Stroke="Blue"/>
    </Chart:C1FlexRadar.HeaderStyle>
    <Chart:C1FlexRadar.Series>
```

```
        <Chart:Series SeriesName="3月"
                      Binding="March"></Chart:Series>
        <Chart:Series SeriesName="4月"
                      Binding="April"></Chart:Series>
        <Chart:Series SeriesName="5月"
                      Binding="May"></Chart:Series>
    </Chart:C1FlexRadar.Series>
</Chart:C1FlexRadar>
```

コード

C# copyCode

```
// FlexRadarのヘッダーを設定します。
flexRadar.Header = "果物チャート";

// 凡例の方向を設定します。
flexRadar.LegendOrientation = C1.Chart.Orientation.Horizontal;

// 凡例の位置を設定します。
flexRadar.LegendPosition = C1.Chart.Position.Bottom;
```

VB copyCode

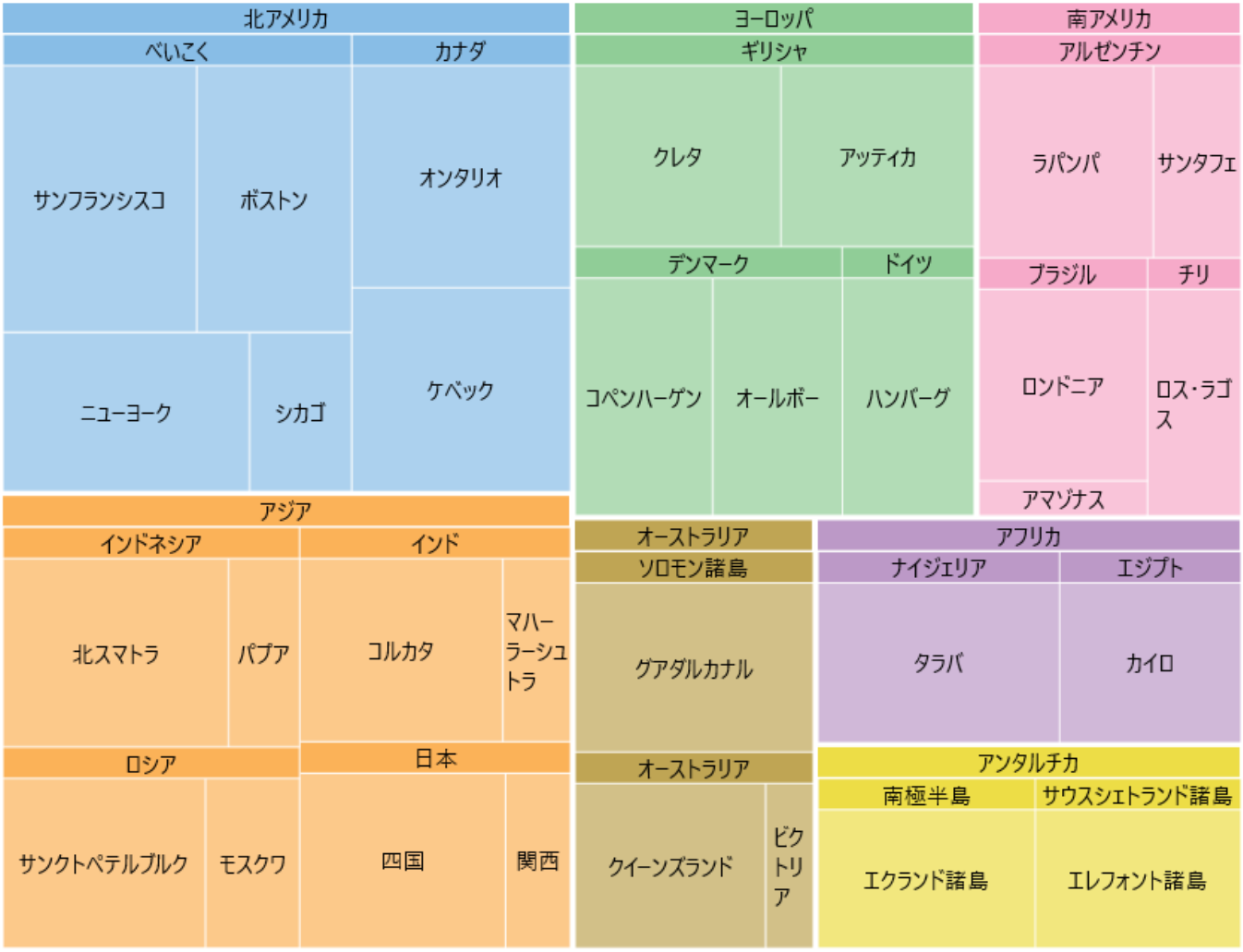
```
' FlexRadarのヘッダーを設定します。
flexRadar.Header = "果物チャート"

' 凡例の方向を設定します。
flexRadar.LegendOrientation = C1.Chart.Orientation.Horizontal

' 凡例の位置を設定します。
flexRadar.LegendPosition = C1.Chart.Position.Bottom
```

TreeMap

階層的な情報やデータは、家系図、プログラミング、組織図、ディレクトリなど、さまざまな階層や組織で役立ちます。このようなデータスポット情報を視覚化することは、特にデータが大きな場合は難しいタスクです。Treemap チャートを使用すると、階層化データを限られたスペースで入れ子の四角形として視覚化できます。大きなデータに含まれるパターンを簡単に把握したり、比率を比較するために便利です。

TreeMap チャートコントロールは、データの連結による階層の表示をサポートし、データをさまざまなレベルにドリルダウンして詳細に分析することができます。このコントロールは、構成している四角形を横長、縦長、正方形のレイアウトでデータを表示するようにカスタマイズできます。

TreeMap と Sunburst チャートはどちらも階層化データの表示と視覚化に理想的ですが、ツリーマップは大量のデータを限られた領域に表示できるため、スペースに制約がある場合に便利です。



次のトピックでは、TreeMap コントロールを理解し、高度な機能を説明します。

主な機能

TreeMap コントロールには、限られた領域に階層化データを表示し、ツリーノード（入れ子の四角形）のサイズを比較することでデータを分析するためのさまざまな機能があります。次のものがあります。

- **階層的なデータ表現**
  TreeMap コントロールは、階層内のデータ値を視覚化し、比率を比較するために役立ちます。

- **レイアウト**
  TreeMap は、複数の表示配置をサポートします。ツリーブランチを正方形、横長長方形、縦長長方形などで示すことができます。
- **階層レベルのカスタマイズ**
  TreeMap コントロールでは、データの**深さ**を変化させて、データを視覚化した上でドリルダウン（または逆ドリルダウン）することで、分析と比較を行うことができます。
- **外観のカスタマイズ**
  TreeMap では、コントロールをスタイル設定し、好みに応じて外観を変化させることができます。ツリーマップチャートでカテゴリをわかりやすく表示するために、さまざまなカラー**パレット**を使用できます。
- **データ連結**
  TreeMap は、さまざまなサイズのデータを含むデータソースに連結でき、このようなデータを限られた四角形の領域に表示できます。
- **選択**
  TreeMap では、ツリーノードやノードのグループを選択して、階層化データの特定のデータ項目に注目することができます。
- **スペース活用の最適化**
  TreeMap は、大量のデータをコンパクトに表示して視覚化する場合に理想的です。ツリーマップチャートを構成する入れ子の四角形やグループは、表示領域に合わせてサイズが調整されます。

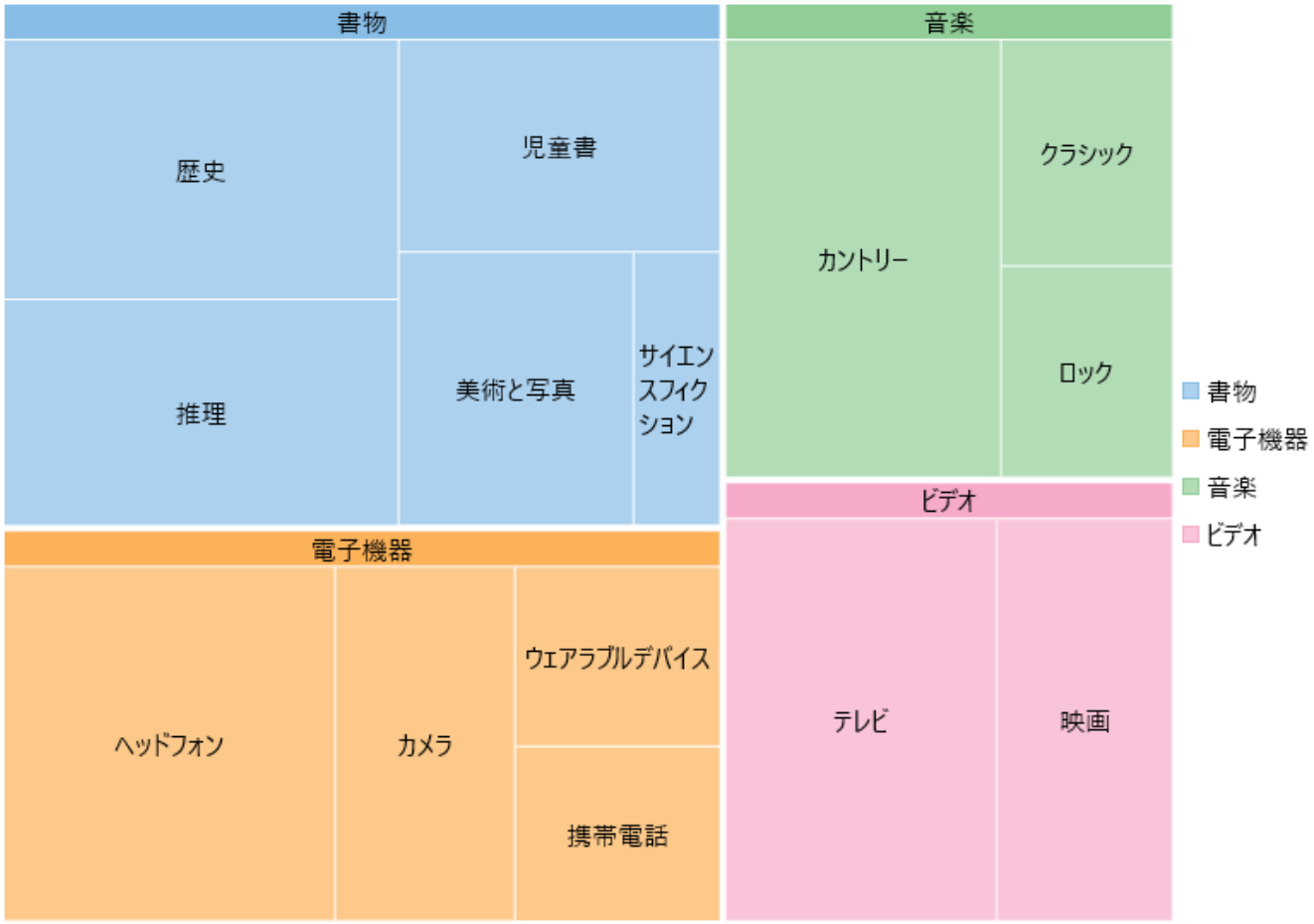
クイックスタート

このクイックスタートでは、TreeMapチャートをUWPアプリケーションに追加し、階層データを表示する手順を説明します。このトピックでは、特定の年にXYZという市で書籍、音楽、ビデオ、ガジェット（コンピュータやタブレットなど）の販売を比較するとします。

TreeMapコントロールに階層データを表示する手順は次のようになります。

- **手順 1：プロジェクトへの TreeMap の追加**
- **手順 2：階層データソースの作成**
- **手順 3：データソースへの TreeMap の連結**
- **手順 4：プロジェクトの実行**

次の図は、特定の年にXYZ市でさまざまな書籍、音楽、ビデオ、ガジェット（コンピュータやタブレットなど）の販売を展示して比較しています。



先頭に戻る

チャートにプロットされるデータポイントを含むオブジェクトのコレクションを指すように、**FlexChartBase**クラスの**ItemsSource**プロパティを設定する必要があります。データ項目を生成してTreeMapチャートに表示するには、**BindingName**プロパティと**Binding**プロパティを設定します。**BindingName** プロパティを、グラフの四角形として表示するデータ項目の名前を指定する文字列値に設定します。そして、**Binding** プロパティを、チャート値（ツリーノードのサイズを計算するのに役立つ数値）を含むチャートアイテムのプロパティの名前を指定する文字列値に設定します。

階層項目のレベルをドリルダウンに指定してチャートに表示するには、**MaxDepth**プロパティを設定します。また、TreeMapの表示レイアウトは**ChartType**プロパティで指定します。カラーパレットを使用してコントロールをスタイル設定し、その外観を変更することもできます。

手順 1：プロジェクトへのTreeMap の追加

1. Visual Studio で**ユニバーサルアプリケーション** を作成します。
2. **TreeMap** コントロールを ツールボックスからページにドラッグアンドドロップします。
   次の dll が自動的にアプリケーションに追加されます。
   - **C1.UWP.dll**
   - **C1.UWP.DX.dll**
   - **C1.UWP.FlexChart.dll**

先頭に戻る

手順 2：階層データソースの作成

コードビューに切り替えて、書物、音楽、電子機器、ビデオ、コンピュータおよびタブレットの販売データを生成するために次のコードを追加します。

- **Visual Basic**

```
Private rnd As New Random()
Private Function rand() As Integer
    Return rnd.[Next](10, 100)
End Function

Public ReadOnly Property Data() As Object()
    Get
        Dim data__1 = New Object() {New With {
         .type = "音楽",
         .items = New () {New With {
             .type = "カントリー",
             .items = New () {New With {
                 .type = "クラシックカントリー",
                 .sales = rand()
            }}
        }, New With {
             .type = "ロック",
             .items = New () {New With {
                 .type = "ファンクロック",
                 .sales = rand()
            }}
        }, New With {
             .type = "クラシック",
             .items = New () {New With {
                 .type = "交響曲",
                 .sales = rand()
            }}
        }}
    }, New With {
         .type = "書物",
         .items = New () {New With {
             .type = "美術と写真",
             .items = New () {New With {
                 .type = "建築",
                 .sales = rand()
            }}
        }, New With {
             .type = "児童書",
             .items = New () {New With {
                 .type = "読者を始めよう",
                 .sales = rand()
            }}
        }, New With {
             .type = "歴史",
             .items = New () {New With {
                 .type = "古代",
                 .sales = rand()
            }}
        }, New With {
             .type = "推理",
             .items = New () {New With {
                 .type = "スリラー",
                 .sales = rand()
            }}
        }, New With {
             .type = "サイエンスフィクション",
             .items = New () {New With {
                 .type = "ファンタジー",
                 .sales = rand()
            }}
        }}
    }, New With {
         .type = "電子機器",
```

```
            .items = New () {New With {
                .type = "ウェアラブルデバイス",
                .items = New () {New With {
                    .type = "アクティビティログ",
                    .sales = rand()
                }}
            }, New With {
                .type = "携帯電話",
                .items = New () {New With {
                    .type = "アクセサリ",
                    .sales = rand()
                }}
            }, New With {
                .type = "ヘッドフォン",
                .items = New () {New With {
                    .type = "イヤホン",
                    .sales = rand()
                }}
            }, New With {
                .type = "カメラ",
                .items = New () {New With {
                    .type = "デジタルカメラ",
                    .sales = rand()
                }}
            }}
        }, New With {
            .type = "ビデオ",
            .items = New () {New With {
                .type = "映画",
                .items = New () {New With {
                    .type = "子ども",
                    .sales = rand()
                }}
            }, New With {
                .type = "テレビ",
                .items = New () {New With {
                    .type = "コメディー",
                    .sales = rand()
                }}
            }}
        }}
        Return data__1
    End Get
End Property
```

- **C#**

```
static Random rnd = new Random();
static int rand()
{
    return rnd.Next(10, 100);
}
public static object[] Data
{
    get
    {
        var data = new object[] { new {
    type = "音楽",
    items = new [] { new {
        type = "カントリー",
        items= new [] { new {
            type= "クラシックカントリー",
            sales = rand()
```

```
            }}
        }, new {
            type= "ロック",
            items= new [] { new {
                type= "ファンクロック",
                sales= rand()
             } }
        }, new {
            type= "クラシック",
            items= new [] { new {
                type= "交響曲",
                sales= rand()
                } }
  }}
}, new {
        type= "書物",
  items= new [] { new {
            type= "美術と写真",
    items= new [] { new {
                type= "建築",
      sales= rand()
    }}
  }, new {
            type= "児童書",
    items= new [] { new {
                type= "読者を始めよう",
      sales= rand()
    } }
  }, new {
            type= "歴史",
    items= new [] { new {
                type= "古代",
      sales= rand()
    } }
  }, new {
            type= "推理",
    items= new [] { new {
                type= " スリラー ",
      sales= rand()
             } }
  }, new {
            type= "サイエンスフィクション",
    items= new [] { new {
                type= "ファンタジー",
      sales= rand()
    }}
  } }
}, new {
        type= "電子機器",
  items= new [] { new {
            type= "ウェアラブルデバイス",
    items= new [] { new {
                type= "アクティビティログ",
      sales= rand()
    }}
  }, new {
            type= "携帯電話",
    items= new [] { new {
                type= "アクセサリ",
      sales= rand()
    } }
  }, new {
            type= "ヘッドフォン",
```

```
    items= new [] { new {
                type= "イヤホン",
      sales= rand()
    } }
  }, new {
            type= "カメラ",
    items= new [] { new {
                type= "デジタルカメラ",
      sales= rand()
             } }
  } }
}, new {
        type= "ビデオ",
  items= new [] { new {
            type= "映画",
    items= new [] { new {
                type= "子ども",
      sales= rand()
    } }
  }, new {
            type= "テレビ",
    items= new [] { new {
                type= "コメディー",
      sales= rand()
    } }
  } }
} };
            return data;
        }
    }
```

先頭に戻る

手順 3：データソースへの TreeMap の連結

次のコードを使用して、データソースへのTreeMapコントロールを連結します。

XAML copyCode

```
    <Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:UwpTreeMapCS"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:Chart="using:C1.Xaml.Chart"
    x:Class="UwpTreeMapCS.QuickStart"
    mc:Ignorable="d"
    DataContext="{Binding RelativeSource={RelativeSource Mode=Self}}">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

        <Chart:C1TreeMap Binding="sales"
                         BindingName="type"
                         ChildItemsPath="items"
                         ItemsSource="{Binding DataContext.Data}"
                    MaxDepth="2">

            <Chart:C1TreeMap.DataLabel>
                <Chart:DataLabel Content="{}{name}"
```

```
                                Position="Center">
                    <Chart:DataLabel.Style>
                        <Chart:ChartStyle/>
                    </Chart:DataLabel.Style>
                </Chart:DataLabel>
            </Chart:C1TreeMap.DataLabel>
        </Chart:C1TreeMap>
    </Grid>
</Page>
```

先頭に戻る

手順 4:プロジェクトの実行

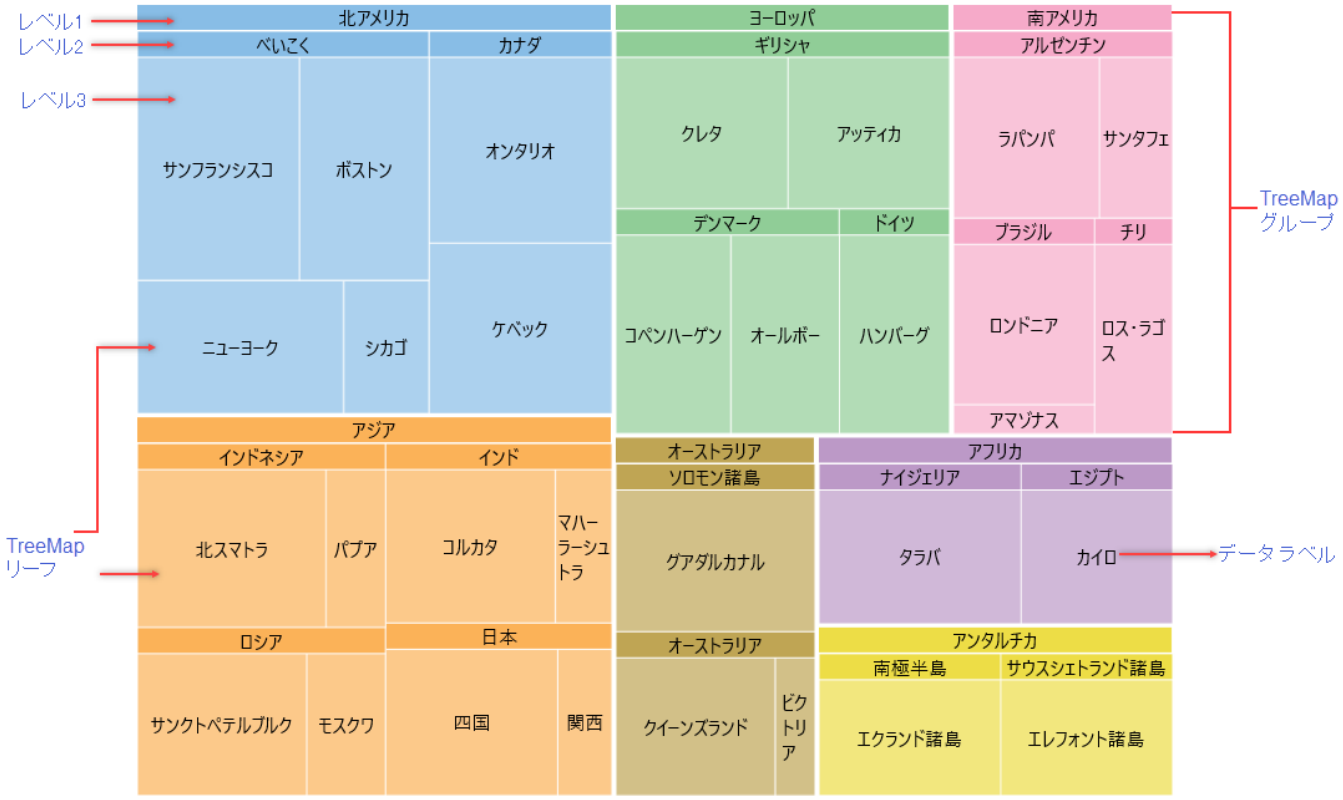
1. **[ビルド]→[ソリューションのビルド]**をクリックして、プロジェクトをビルドします。
2. **[F5]**を押してプロジェクトを実行します。

先頭に戻る

要素

ツリーマップチャートは、個別のデータ項目を表す長方形で構成され、データの階層的な性質を表すカテゴリにグループ化されます。グループを構成する個別のデータ項目は、リーフノードと呼ばれます。ノードのサイズは、表すデータに比例します。

次の図に、TreeMap コントロールのメイン要素を示します。



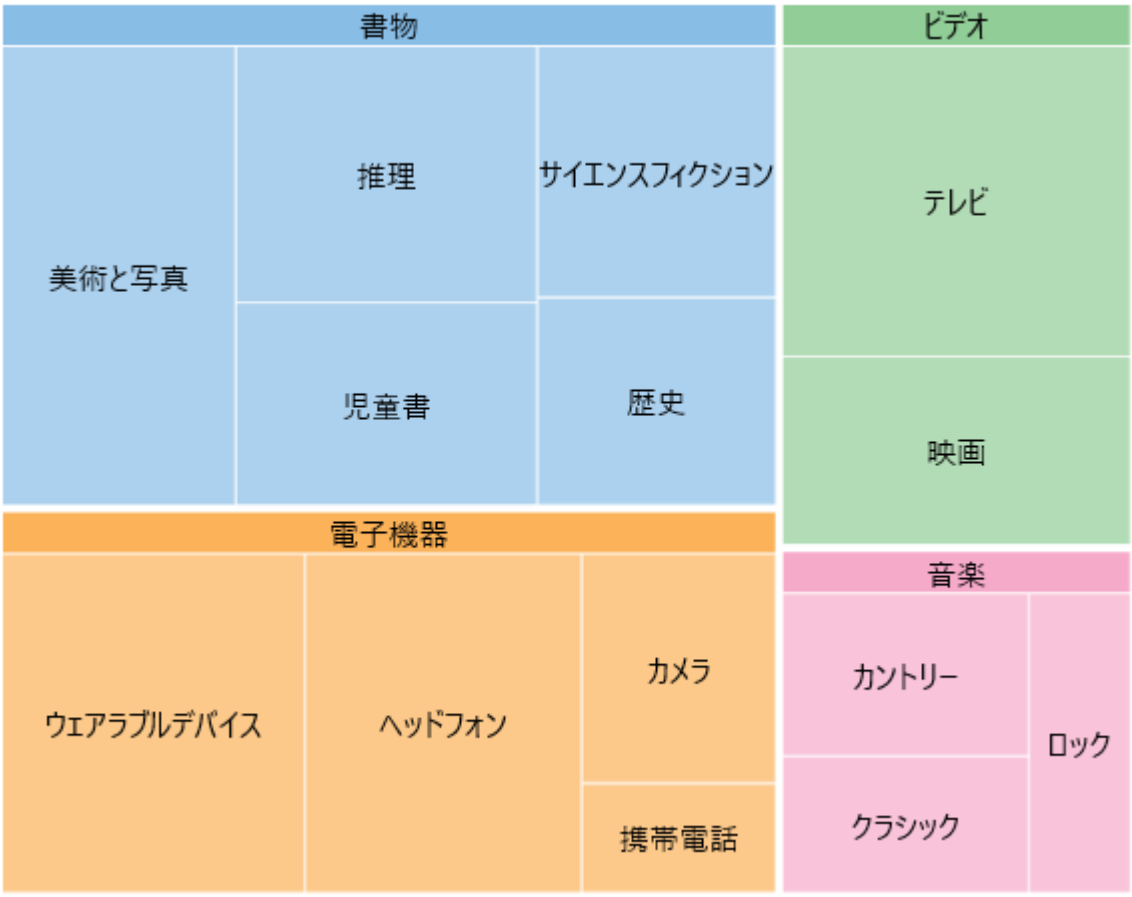
レイアウト

TreeMap では、データ項目とグループを四角形で表し、さまざまな配置で表示できます。ツリーマップの四角形は、正方形、横

長、縦長のレイアウトに配置できます。ツリーマップで目的のレイアウトを設定するには、**C1TreeMap** クラスの **ChartType** プロパティを使用します。このプロパティは、**TreeMapType** 列挙を受け取ります。TreeMap チャートコントロールのデフォルトのレイアウトは、正方形です。

正方形

正方形のレイアウトは、ツリーマップの四角形（データ項目とグループ）をほぼ正方形として配置します。このレイアウトでは、正方形の配置によって表現の正確さが向上するため、比較やパターンの把握が容易になります。このレイアウトは、大きなデータセットでたいへん便利です。



XAML copyCode

```
<Chart:C1TreeMap Binding="Value"
    BindingName="Name"
    ChartType="Squarified" Margin="0,0,80,400">
```

水平方向

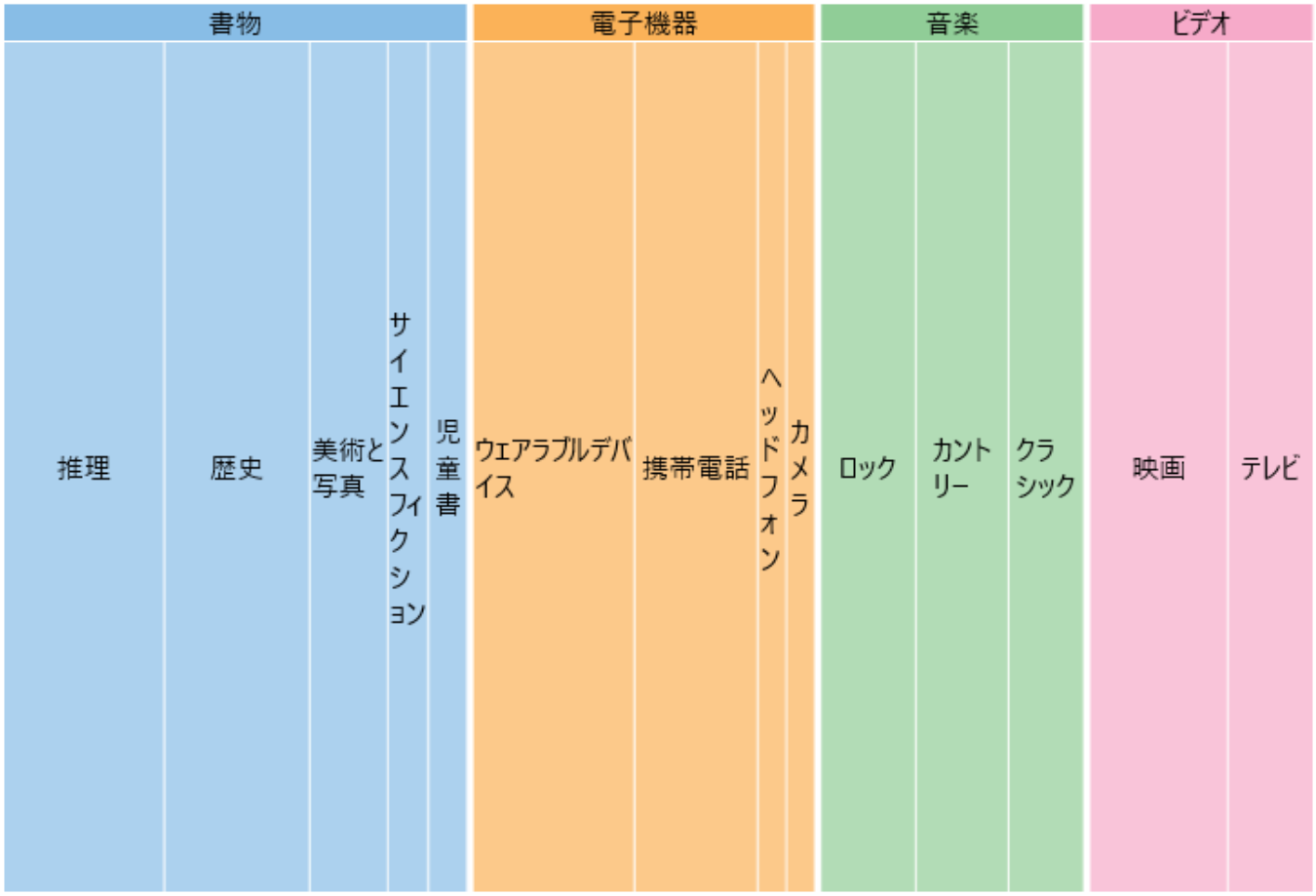
水平方向のレイアウトは、ツリーマップの四角形を積み重ねて行の形式にします。四角形の幅が高さより大きくなります。

XAML copyCode

```
<Chart:C1TreeMap Binding="Value"
    BindingName="Name"
    ChartType="Horizontal" Margin="205,220,-55,105">
```

垂直方向

垂直方向のレイアウトでは、ツリーマップの四角形を隣接させて列の形式にします。四角形の高さが幅より大きくなります。



XAML	copyCode

```
<Chart:C1TreeMap Binding="Value"
                 BindingName="Name"
                 ChartType="Vertical" Margin="25,405,110,0">
```

水平方向と垂直方向のツリーマップは、情報の順序を維持して表示する場合に役立ちます。

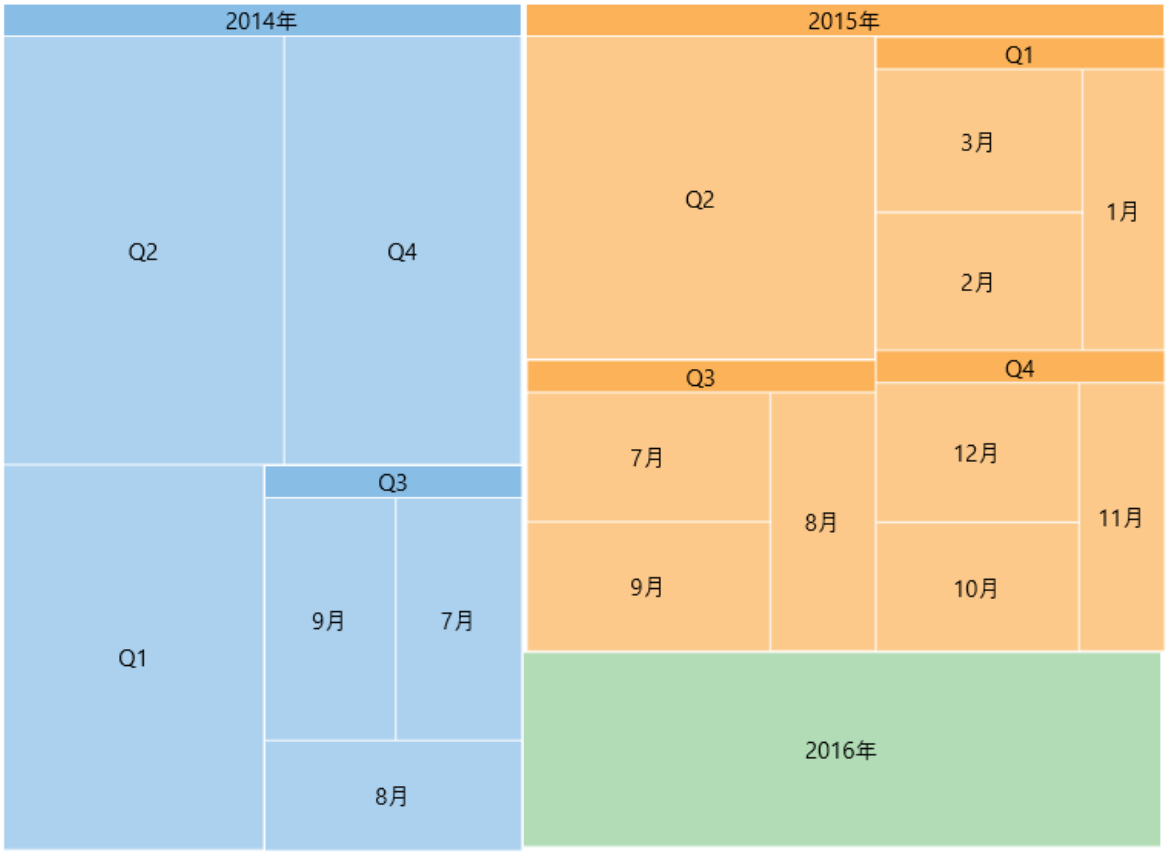
データ連結

TreeMap チャートコントロールは、階層化データに連結して、ツリー型のデータの要素を入れ子の四角形として表します。データソースに連結してデータ項目を四角形として表示すると、構成要素となる四角形のサイズと色に基づいてデータ項目を分析および比較できます。

FlexChartBase クラスは、**ItemsSource** プロパティを公開しています。このプロパティは、データを含むオブジェクトのコレクションを受け取り、ツリーマップチャートを生成します。**Binding** および **BindingName** プロパティを使用して、データ項目やそれぞれのカテゴリまたはグループに対応する四角形ノードを生成します。**Binding** プロパティは、四角形ノードのサイズ計算に使用される数値データ値を含むデータ項目のプロパティの名前を表す文字列値を受け取ります。一方、**BindingName** は、データ項目の名前を表す文字列値を受け取ります。**ChildItemPath** プロパティは、データ内の子項目をコントロールに伝えることで、指定されたデータコレクションの階層化構造を維持するために使用されます。

ツリーマップチャートでデータがどのように生成されるかを詳細に説明するために、複数のブランドの小売店の年次売上（販売数）を比較します。ツリーマップチャートを使用すると、分析をさらに四半期、月にドリルダウンできます。年次の売上が最上位レベルの四角形で表され、これらの年の四半期の売上が次のレベルを表し、次のレベルである月の売上がツリーマップのリーフノードを構成します。

次の図に、小売店の売上（販売数）を TreeMap チャートコントロールで示します。画像に階層化データが最大 3 つのレベル、つまり各年の四半期内の月のレベルまで表示されています。



先頭に戻る

この例では、DataSource.cs クラスで生成されるデータがツリーマップチャートのソースになります。**DataSource** プロパティは、DataSource.cs クラスで生成される階層化データのコレクションを受け取ります。

1. **階層化データソースを作成する**
    1. コードビューで、次のコードで示すように、DataService クラスを作成して階層化データを生成します。
        - **Visual Basic**

```
Public Class DataService
    Private rnd As New Random()
    Shared _default As DataService

    Public Shared ReadOnly Property Instance() As DataService
        Get
            If _default Is Nothing Then
                _default = New DataService()
            End If

            Return _default
        End Get
    End Property

    Public Shared Function CreateHierarchicalData() As List(Of DataItem)
        Dim rnd As Random = Instance.rnd

        Dim years As New List(Of String)()
        Dim times As New List(Of List(Of String))() From {
            New List(Of String)() From {
                "1月",
                "2月",
                "3月"
            },
            New List(Of String)() From {
                "4月",
                "5月",
                "6月"
            },
            New List(Of String)() From {
                "7月",
```

```
                "8月",
                "9月"
            },
            New List(Of String)() From {
                "10月",
                "11月",
                "12月"
            }
        }

        Dim items As New List(Of DataItem)()
        Dim yearLen = Math.Max(CInt(Math.Round(Math.Abs(5 - Instance.rnd.NextDouble()
         * 10))), 3)
        Dim currentYear As Integer = DateTime.Now.Year
        For i As Integer = yearLen To 1 Step -1
            years.Add((currentYear - i).ToString())
        Next
        Dim quarterAdded = False

        years.ForEach(Function(y)
                      Dim i = years.IndexOf(y + "年")
                      Dim addQuarter = Instance.rnd.NextDouble() > 0.5
                      If Not quarterAdded AndAlso i = years.Count - 1 Then
                          addQuarter = True
                      End If
                      Dim year = New DataItem() With {
                          .Year = y
                      }
                      If addQuarter Then
                          quarterAdded = True
                          times.ForEach(Function(q)
                                            Dim addMonth = Instance.rnd.NextDouble() > 0.5
                                            Dim idx As Integer = times.IndexOf(q)
                                            Dim quar As String
                                            quar = "Q" + (idx + 1).ToString
                                            Dim quarters = New DataItem() With {
                                                .Year = y,
                                                .Quarter = quar
                                            }
                                             If addMonth Then
                                                q.ForEach(Function(m)
                                                  quarters.Items.Add(New DataItem() With {
                                                              .Year = y,
                                                              .Quarter = quar,
                                                              .Month = m,
                                                              .Value = rnd.[Next](20, 30)
                                                          })

                                                           End Function)
                                            Else
                                                quarters.Value = rnd.[Next](80, 100)
                                            End If
                                            year.Items.Add(quarters)
                                        End Function)
                      Else
                         year.Value = rnd.[Next](80, 100)
                      End If
                      items.Add(year)

                  End Function)

        Return items
    End Function

End Class
```

- **C#**

```
public class DataService
{
    Random rnd = new Random();
    static DataService _default;

    public static DataService Instance
    {
```

```
        get
        {
            if (_default == null)
            {
                _default = new DataService();
            }

            return _default;
        }
    }

    public static List<DataItem> CreateHierarchicalData()
    {
        Random rnd = Instance.rnd;

        List<string> years = new List<string>();
        List<List<string>> times = new List<List<string>>()
        {
            new List<string>() { "1月", "2月", "3月"},
            new List<string>() { "4月", "5月", "6月"},
            new List<string>() { "7月", "8月", "9月"},
            new List<string>() { "10月", "11月", "12月" }
        };

        List<DataItem> items = new List<DataItem>();
        var yearLen = Math.Max((int)Math.Round(Math.Abs(5 -
        Instance.rnd.NextDouble() * 10)), 3);
        int currentYear = DateTime.Now.Year;
        for (int i = yearLen; i > 0; i--)
        {
            years.Add((currentYear - i).ToString());
        }
        var quarterAdded = false;

        years.ForEach(y =>
        {
            var i = years.IndexOf(y);
            var addQuarter = Instance.rnd.NextDouble() > 0.5;
            if (!quarterAdded && i == years.Count - 1)
            {
                addQuarter = true;
            }
            var year = new DataItem() { Year = y+"年"};
            if (addQuarter)
            {
                quarterAdded = true;
                times.ForEach(q =>
                {
                    var addMonth = Instance.rnd.NextDouble() > 0.5;
                    int idx = times.IndexOf(q);
                    var quar = "Q" + (idx + 1);
                    var quarters = new DataItem() { Year = y, Quarter = quar };
                    if (addMonth)
                    {
                        q.ForEach(m =>
                        {
                            quarters.Items.Add(new DataItem()
                            {
                                Year = y,
                                Quarter = quar,
                                Month = m,
                                Value = rnd.Next(20, 30)
                            });
                        });
                    }
                    else
                    {
                        quarters.Value = rnd.Next(80, 100);
                    }
                    year.Items.Add(quarters);
                });
            }
            else
```

```
            {
                year.Value = rnd.Next(80, 100);
            }
            items.Add(year);
        });

        return items;
    }

}
```

2. DataItem クラスを作成して、データ項目とカテゴリを表すオブジェクトのリストを定義します。
   - **Visual Basic**

```
Public Class DataItem
    Private _items As List(Of DataItem)

    Public Property Year() As String
        Get
            Return m_Year
        End Get
        Set
            m_Year = Value
        End Set
    End Property
    Private m_Year As String
    Public Property Quarter() As String
        Get
            Return m_Quarter
        End Get
        Set
            m_Quarter = Value
        End Set
    End Property
    Private m_Quarter As String
    Public Property Month() As String
        Get
            Return m_Month
        End Get
        Set
            m_Month = Value
        End Set
    End Property
    Private m_Month As String
    Public Property Value() As Double
        Get
            Return m_Value
        End Get
        Set
            m_Value = Value
        End Set
    End Property
    Private m_Value As Double
    Public ReadOnly Property Items() As List(Of DataItem)
        Get
            If _items Is Nothing Then
                _items = New List(Of DataItem)()
            End If

            Return _items
        End Get
    End Property
End Class
```

   - **C#**

```
public class DataItem
{
    List<DataItem> _items;

    public string Year { get; set; }
    public string Quarter { get; set; }
    public string Month { get; set; }
    public double Value { get; set; }
    public List<DataItem> Items
    {
        get
```

```
            {
                if (_items == null)
                {
                    _items = new List<DataItem>();
                }

                return _items;
            }
        }
    }
```

先頭に戻る

2. **データソースへの TreeMap の連結**

次のコードを使用して、データソースへのTreeMapコントロールを連結します。

XAML copyCode

```
<Page
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="using:UwpTreeMapCS"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:Chart="using:C1.Xaml.Chart"
x:Class="UwpTreeMapCS.DataBinding"
mc:Ignorable="d"
DataContext="{Binding RelativeSource={RelativeSource Mode=Self}}">

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.DataContext>
        <local:TreeMapViewModel />
    </Grid.DataContext>
    <Chart:C1TreeMap Binding="Value"
                  BindingName="Year,Quarter,Month"
                  ChildItemsPath="Items"
                  ItemsSource="{Binding HierarchicalData}"
                  MaxDepth="3">
```

先頭に戻る

- **Visual Basic**

```
Public Class TreeMapViewModel
    Public ReadOnly Property HierarchicalData() As List(Of DataItem)
        Get
            Return DataService.CreateHierarchicalData()
        End Get
    End Property
End Class
```

- **C#**

```
public class TreeMapViewModel
{
    public List<DataItem> HierarchicalData
    {
        get
        {
            return DataService.CreateHierarchicalData();
        }
    }
}
```

3. **プロジェクトの実行**
    1. **[ビルド]→[ソリューションのビルド]**をクリックして、プロジェクトをビルドします。
    2. **[F5]**を押してプロジェクトを実行します。

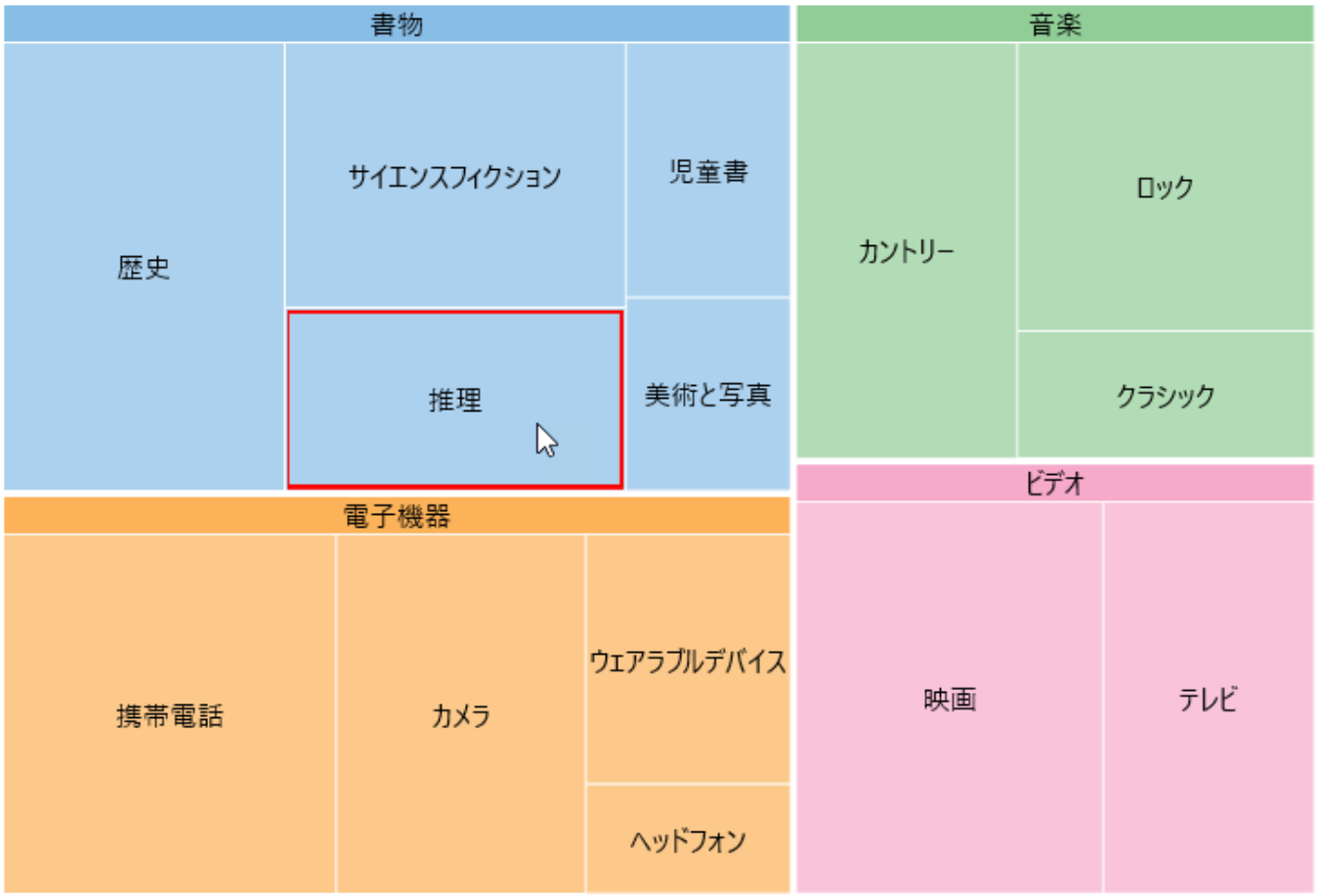
先頭に戻る

選択

TreeMap チャートでは、データ項目とグループを選択できます。クリックするだけで、ノードを選択してフォーカスを設定できます。**FlexChartBase** クラスで提供されている **SelectionMode** プロパティを **ChartSelectionMode** 列挙に含まれる次の値のいずれかに設定する必要があります。

- **None（デフォルト）**：選択は無効です。
- **Point**：ポイントが選択されます。

次の図に、TreeMapにデフォルトで選択したところを示します。



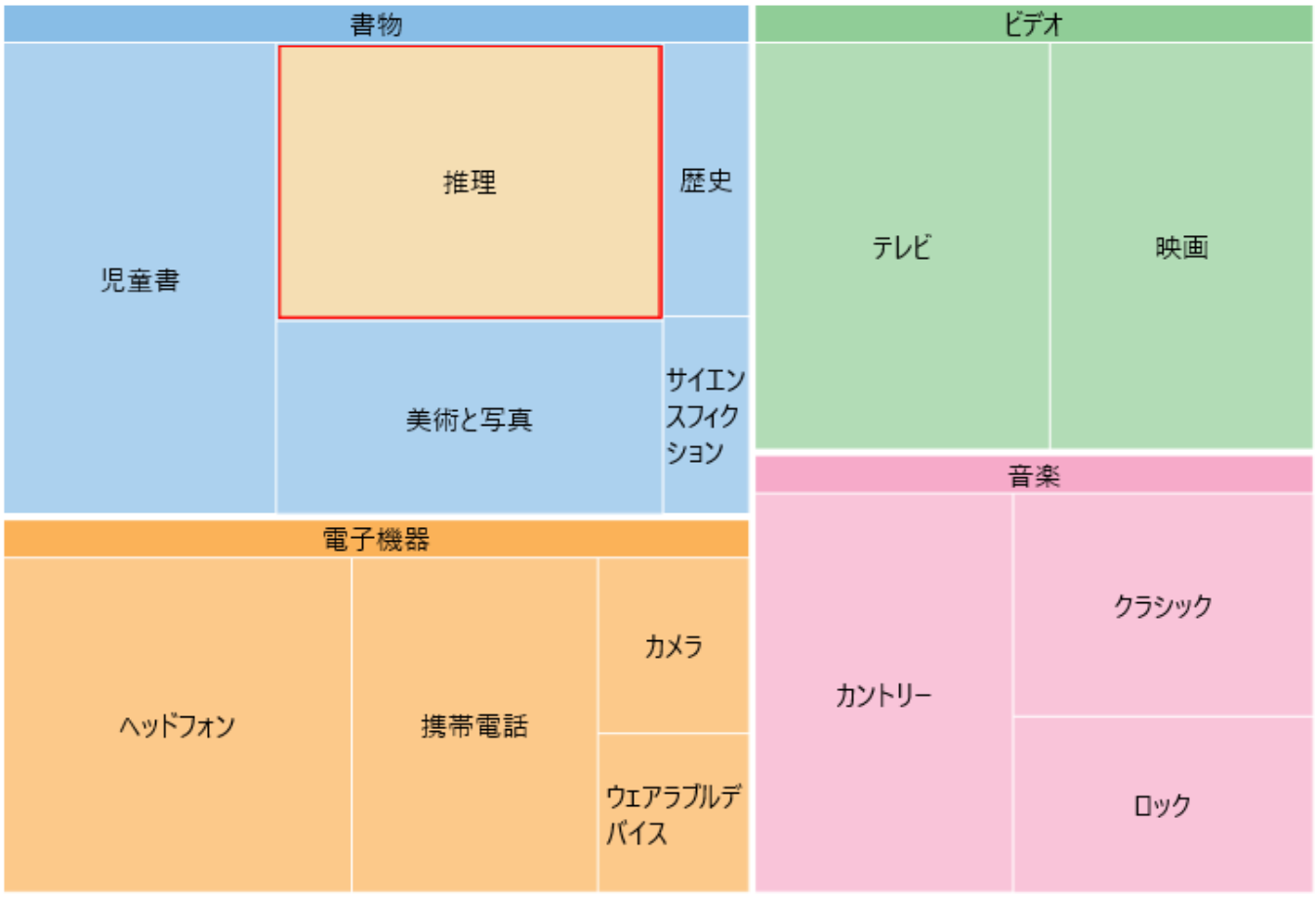
次のコードスニペットでは、ツリーマップチャートの **SelectionMode** プロパティの設定方法を示します。

XAML copyCode

```
<Chart:C1TreeMap Binding="sales"
                 MaxDepth="2"
                 BindingName="type"
                 ChildItemsPath="items"
                 ItemsSource="{Binding DataContext.Data}"
                 SelectionMode="Point">
```

カスタマイズされた TreeMap の選択

TreeMap の選択をカスタマイズするために、**SelectionStyle** プロパティを使用して、次の画像のように選択項目をスタイル設定できます。



次のコードスニペットは、**SelectionStyle** プロパティを活用して、選択されている TreeMap ノードの塗りつぶし色を変更します。

XAML copyCode

```
<Chart:C1TreeMap Binding="sales"
                 MaxDepth="2"
                 BindingName="type"
                 ChildItemsPath="items"
                 ItemsSource="{Binding DataContext.Data}"
                 SelectionMode="Point">
    <Chart:C1TreeMap.SelectionStyle>
        <Chart:ChartStyle Fill="Wheat"/>
    </Chart:C1TreeMap.SelectionStyle>
```

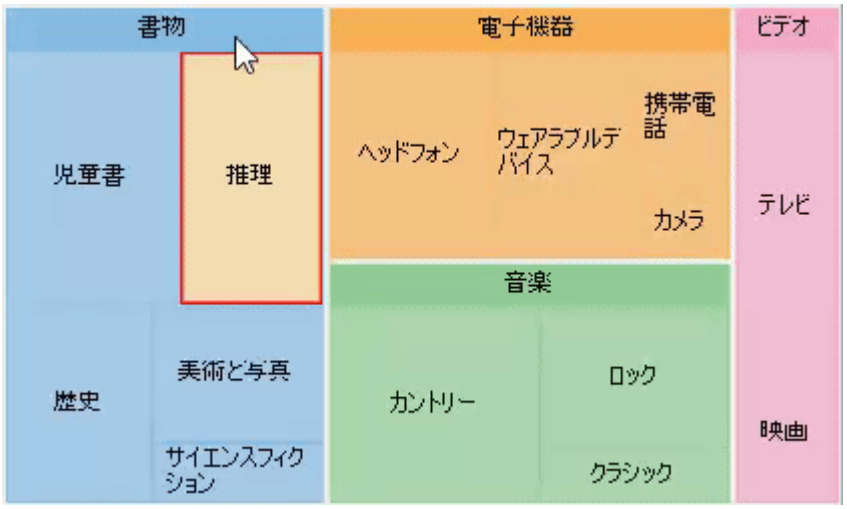
さらに、**SelectionChanged** イベントを処理することで、TreeMap の選択の動作をカスタマイズできます。また、**SelectedIndex** と **SelectedItem** のプロパティを活用して、アプリケーションで取得した情報を再利用できます。

ドリルダウン

TreeMap では、データのデータ項目をドリルダウンして詳細な分析を行うことができます。エンドユーザーは、目的のノードをクリックするだけで、データ階層内の下位レベルにアクセスできます。階層を上に戻るには、プロット領域を右クリックするだけです。

TreeMap のドリルダウン機能は、**MaxDepth** プロパティを 0 より大きな値に設定した場合にのみ有効になります。このプロパティは、TreeMap チャートの階層化データのレベルを定義します。

次の gif 画像では、ドリルダウンの例として、クリックされた TreeMap ノードのデータポイントを表示しています。



TreeMap のドリルダウン機能は、TreeMap ノードの選択が無効の場合、つまり **SelectionMode** プロパティが **None** に設定されている場合にのみ機能します。選択の詳細については、TreeMap での選択を参照してください。