FlexGrid for UWP

2019.08.19 更新

グレープシティ株式会社

FlexGrid for UWP

目次

FlexGrid for UWP

この単純かつ強力なグリッドコントロールを使用すると、表形式のデータを列と行に表示できます。**FlexGrid for UWP**は、モダンなタッチベースのエンタープライズアプリケーション用に再設計された、使い慣れた従来型のデータグリッドを提供します。

主な特長

FlexGrid for UWPには、次の主な特長があります。

- **タッチ操作**

  将来、スクロールバーはなくなります。**FlexGrid for UWP** はモバイル環境に適した UX を提供するように設計されており、ユーザーは、グリッドをスライドして列間をスクロールしたり行を移動することができます。

- **柔軟なデータ連結**

  C1FlexGrid コントロールは、**IEnumerable** を実装する任意のオブジェクトに簡単に連結したり、オブジェクトのリストに直接連結することができます。

- **非連結モード**

  データ連結のサポートに加えて、**C1FlexGrid** は、非連結モードでも十分に機能します。WinForms バージョンのコントロールから使い慣れた構文を使用して、グリッドに行と列を簡単に追加できます。非連結モードの場合、**C1FlexGrid** コントロールはデータを内部的に保存します。

- **グループ化**

  C1FlexGrid は、データ連結されると、基底の **CollectionView** に対するグループ化をサポートします。グループ化された行が作成され、タップによって展開したり折りたたむことができます。

- **セルの編集**

  C1FlexGrid は、Windows ストア用の使いやすいインタフェースによるセルの編集をサポートします。編集は、Web アプリケーションやデスクトップアプリケーションのようにインラインではなく、タッチインタフェースに最適な独立したポップアップで行われます。

- **カスタムセル**

  C1FlexGrid では、より単純なインタフェース **ICellFactory**が提供されているため、カスタムセルの定義がこれまでより簡単になりました。カスタムセルは、グリッドセルにテキスト以外のコンテンツを表示する場合や、条件付き書式設定を適用する場合に役立ちます。

- **セル結合**

  C1FlexGrid では、グリッドレベルでセル結合を行うことができます。特定の行および列に対して **AllowMerging** プロパティを設定するだけで、必要に応じてこの機能を有効にすることができます。セル結合は、中身が同じ隣接するセル同士を結合し、常にユーザーが最も読み取りやすいようにテキストを配置します。

- **列/行ヘッダー**

  FlexGrid は、列ヘッダーとして複数の固定行をサポートし、行ヘッダーとして複数の固定列をサポートします。このしくみを利用して、複数のセルを含むヘッダーから成る「バンド」を作成すると、列をグループにまとめる際に便利です。この機能を実現するためにセル結合が使用されます。

- **列と行のフリーズ**

  Rows.Frozen または **Columns.Frozen** プロパティを設定するだけで、任意の数の行および列をフリーズできます。フリーズされたペインは、スクロールしてもビューに表示されたままになります。

- **行および列の自動サイズ変更**

  C1FlexGrid コントロールには、現在表示されているデータを考慮して動作する **AutoSizeRow** メソッドと **AutoSizeColumn**メソッドが含まれます。これは、迅速かつ正確に実行されます。

- **ストリームへの保存**

  Save メソッドを使用して、データを簡単に出力して保存できます。保存形式には、プレーンテキスト、HTML、CSV があります。

- **列と行のドラッグ アンド ドロップ**

  AllowDraggingと AllowDropプロパティを設定するだけで、グリッド内に任意の行または列をドラッグ アンド ドロップしてその位置を変更できます。

- **Row Details**

  FlexGrid は、行の詳細をデータテンプレートとして使用して、テキスト、画像、バインドされたデータコントロールの助けで追加の情報を表示する柔軟性を提供します。

クイックスタート

このクイックスタートガイドは、**FlexGrid for UWP**を初めて使用するユーザーのために用意されています。このクイックスタートガイドでは、最初に Visual Studio で新しいプロジェクトを作成し、アプリケーションに **FlexGrid for UWP**コントロールを追加し、コンテンツをコントロールに追加します。

手順1:アプリケーションへの FlexGrid の追加

この手順では、最初に Visual Studio で FlexGrid for UWPを使用する UWP スタイルのアプリケーションを作成します。プロジェクトをセットアップし、C1FlexGrid コントロールをアプリケーションに追加するには、次の手順に従います。

1. Visual Studio 2012 で、[**ファイル**]→[**新規作成**]→[**プロジェクト**]を選択します。
2. [**新しいプロジェクト**]ダイアログボックスで、左ペインの言語を展開し、言語の下で[**Windows ストア**]を選択し、テンプレートリストで[**新しいアプリケーション (XAML)**]を選択します。名前を入力し、[**OK**]をクリックしてプロジェクトを作成します。
3. **MainPage.xaml** が開いていない場合は開きます。<Grid> タグと </Grid> タグの間にカーソルを置き、1回クリックします。
4. ツールボックスに移動し、[**C1FlexGrid**]アイコンをダブルクリックして、コントロールをアプリケーションに追加します。
5. **C1FlexGrid** のマークアップを次のように編集します。

   マークアップ

   ```
   <FlexGrid:C1FlexGrid x:Name="flexgrid1" AllowResizing="Both"
   AllowDragging="Both"
           AllowDrop="True" ColumnHeaderForeground="Black"/>
   ```

   このマークアップは、コントロールに名前を付け、列/行をサイズ変更、ドラッグ、およびドロップする機能を設定し、ヘッダーテキストの色を設定します。

ここまでの成果

これで、UWP スタイルのアプリケーションを作成できました。次の手順では、グリッドに表示されるデータを追加するコードをアプリケーションに追加します。

手順2:C1FlexGrid アプリケーションへのコードの追加

前の手順では、新しいUWP スタイルのプロジェクトを作成し、**C1FlexGrid** コントロールをアプリケーションに追加しましたが、現時点のグリッドにはデータが含まれていません。アプリケーションを実行しても、グリッドには何も表示されません。この手順では、引き続き、XML データソースをプロジェクトに追加し、そのデータソースにグリッドを連結します。

Visual Studio でデータソースを追加し、グリッドを連結するには、次の手順に従います。

1. ソリューションエクスプローラウィンドウで、プロジェクトを右クリックし、[**追加**]→[**新しい項目**]を選択します。
2. [**新しい項目の追加**]ダイアログボックスで、インストール済みテンプレートのリストから[**XML ファイル**]を選択します。そのファイルに「**Products.xml**」という名前を付け、[**追加**]をクリックしてダイアログボックスを閉じます。
   これで **Products.xml** ファイルがプロジェクトに含まれ、自動的に開かれます。
3. **Products.xml** ファイルにある既存のテキストを次の XML マークアップに置き換え、ファイルを保存します。

   XML でマークアップの書き方

   マークアップ

   ```
   <?xml version="1.0" encoding="utf-8" ?>
   <Products>
         <Product Name="Chai" Category="Beverages" Unit="10 boxes x 20 bags"
   ```

```
Price="18" />
      <Product Name="Chang" Category="Beverages" Unit="24 - 12 oz bottles"
Price="19" />
      <Product Name="Aniseed Syrup" Category="Condiments" Unit="12 - 550 ml
bottles" Price="10" />
      <Product Name="Chef Anton's Cajun Seasoning" Category="Condiments"
Unit="48 - 6 oz jars" Price="22" />
      <Product Name="Chef Anton's Gumbo Mix" Category="Condiments" Unit="36
boxes" Price="21.35" />
      <Product Name="Grandma's Boysenberry Spread" Category="Condiments"
Unit="12 - 8 oz jars" Price="25" />
      <Product Name="Uncle Bob's Organic Dried Pears" Category="Produce"
Unit="12 - 1 lb pkgs." Price="30" />
      <Product Name="Northwoods Cranberry Sauce" Category="Condiments" Unit="12
- 12 oz jars" Price="40" />
      <Product Name="Mishi Kobe Niku" Category="Meat/Poultry" Unit="18 - 500 g
pkgs." Price="97" />
      <Product Name="Ikura" Category="Seafood" Unit="12 - 200 ml jars"
Price="31" />
      <Product Name="Queso Cabrales" Category="Dairy Products" Unit="1 kg pkg."
Price="21" />
      <Product Name="Queso Manchego La Pastora" Category="Dairy Products"
Unit="10 - 500 g pkgs." Price="38" />
      <Product Name="Konbu" Category="Seafood" Unit="2 kg box" Price="6" />
      <Product Name="Tofu" Category="Produce" Unit="40 - 100 g pkgs."
Price="23.25" />
      <Product Name="Genen Shouyu" Category="Condiments" Unit="24 - 250 ml
bottles" Price="15.5" />
      <Product Name="Pavlova" Category="Condiments" Unit="32 - 500 g boxes"
Price="17.45" />
      <Product Name="Alice Mutton" Category="Meat/Poultry" Unit="20 - 1 kg tins"
Price="39" />
      <Product Name="Carnarvon Tigers" Category="Seafood" Unit="16 kg pkg."
Price="62.5" />
      <Product Name="Teatime Chocolate Biscuits" Category="Confections" Unit="10
boxes x 12 pieces" Price="9.2" />
      <Product Name="Sir Rodney's Marmalade" Category="Confections" Unit="30
gift boxes" Price="81" />
      <Product Name="Sir Rodney's Scones" Category="Confections" Unit="24 pkgs.
x 4 pieces" Price="10" />
      <Product Name="Gustaf's Knäckebröd" Category="Grains/Cereals" Unit="24 -
500 g pkgs." Price="21" />
      <Product Name="Tunnbröd" Category="Grains/Cereals" Unit="12 - 250 g pkgs."
Price="9" />
      <Product Name="Guaraná Fantástica" Category="Beverages" Unit="12 - 355 ml
cans" Price="4.5" />
      <Product Name="NuNuCa Nuß-Nougat-Creme" Category="Confections" Unit="20 -
450 g glasses" Price="14" />
</Products>
```

これは、標準の Microsoft Northwind データベースの Products テーブルからデータを受け取って追加します。

4. [**表示**]→[**コード**]を選択してコードビューに切り替えます。
5. 次のコードを Page コンストラクタに追加します。

Visual Basic コードの書き方

```
Visual Basic
Public Sub New()
    Me.InitializeComponent()
    LoadData()
End Sub
```

C# コードの書き方

```
C#
public MainPage()
{
    this.InitializeComponent();
    LoadData();
}
```

6. メインクラス内のページのコンストラクタの直後に次のコードを追加します。

Visual Basic コードの書き方

```
Visual Basic
  'Product クラスを作成します。
    Public Class Product
        Private _Name As String
        Public Property Name() As String
            Get
                Return _Name
            End Get
            Set(ByVal value As String)
                _Name = value
            End Set
        End Property
        Private _Category As String
        Public Property Category() As String
            Get
                Return _Category
            End Get
            Set(ByVal value As String)
                _Category = value
            End Set
        End Property
        Private _Unit As String
        Public Property Unit() As String
            Get
                Return _Unit
            End Get
            Set(ByVal value As String)
                _Unit = value
            End Set
        End Property
        Private _Price As String
```

```
        Public Property Price() As String
            Get
                Return _Price
            End Get
            Set(ByVal value As String)
                _Price = value
            End Set
        End Property
    End Class
    Private Sub LoadData()
        'XML データソースを初期化します。
        Dim ProductsDoc As XDocument = XDocument.Load("Products.xml")
        Dim data As IEnumerable(Of Product) = (From Product In
ProductsDoc.Descendants("Product") Select New Product With {.Name =
Product.Attribute("Name").Value, .Category =
Product.Attribute("Category").Value, .Unit = Product.Attribute("Unit").Value,
.Price = Product.Attribute("Price").Value}).ToList
        'C1FlexGrid コントロールをデータソースに連結します。
        flexgrid1.ItemsSource = data
    End Sub
    End Sub
```

C# コードの書き方

C#

```
// Product クラスを作成します
public class Product
{
    public string Name { get; set; }
    public string Category { get; set; }
    public string Unit { get; set; }
    public string Price { get; set; }
}
private void LoadData()
{

     // XML データソースを初期化します
     XDocument ProductsDoc = XDocument.Load("Products.xml");
     List<Product> data = ( from Product in ProductsDoc.Descendants( "Product" )
            select new Product
            {
                Name = Product.Attribute("Name").Value,
                Category = Product.Attribute("Category").Value,
                Unit = Product.Attribute("Unit").Value,
                Price = Product.Attribute("Price").Value
            }
     ).ToList();
     // C1FlexGrid コントロールをデータソースに連結します
     flexgrid1.ItemsSource = data;
}
```

手順3:C1FlexGrid アプリケーションの実行

これまでに UWP スタイルのアプリケーションを作成し、外観と動作をカスタマイズしたので、次にアプリケーションを実行します。アプリケーションを実行し、**FlexGrid for UWP**の実行時の動作を確認するには、次の手順に従います。

1. [**デバッグ**]メニューから[**デバッグ開始**]を選択し、実行時にアプリケーションがどのように表示されるかを確認します。製品データがグリッドに表示されることを確認します。

	Name	Category	Unit	Price
	バードワイン	飲料	150g×20袋	210¥
	コーヒーマイルド	飲料	250ml×20本	1,000¥
	ホワイトソルト	調味料	250ml×20本	100¥
	ブラックペッパー	調味料	250ml×20本	100¥
	ピュアシュガー	調味料	150g×20袋	1,000¥
	うまい素	調味料	500g	2,000¥
	オタル白ラベル	加工食品	500g×50パック	2,500¥
	ピュアデミグラスソース	調味料	250ml×20本	1,000¥
	ベターローストハム	肉類	50g×5本	3,500¥

2. 列ヘッダー(たとえば 種別)をクリックし、別の列ヘッダー(たとえば、価格)の前または後にドラッグして、列の位置を変更します。
3. 行の場合も同じように、行ヘッダーをクリックして別の行ヘッダーの前または後にドラッグし、行の位置を変更します。
4. 列ヘッダー(たとえば、商品名)の右端をクリックし、列の端をドラッグして列ヘッダーのサイズを変更します。
5. 同じように、行ヘッダーの上端や下端をドラッグして行ヘッダーのサイズを変更します。

おめでとうございます!

これで **FlexGrid for UWP**クイックスタートは完了です。**FlexGrid for UWP**アプリケーションを作成し、**C1FlexGrid** コントロールをカスタマイズし、アプリケーションの実行時機能をいくつか確認することができました。

XAML クイックリファレンス

開発を開始するには、ルート要素タグに **ComponentOne** 名前空間宣言を追加します。

XAML

```
xmlns:FlexGrid="using:C1.Xaml.FlexGrid"
```

コントロールを Visual Studio ツールボックスに配置し、それをダブルクリックするかタップすることで、コントロールと名前空間の両方を追加することもできます。

FlexGrid for UWPは、次の図のようになります。

	製品名	種別	単位	単価
	Chai	Beverages	10 boxes x 20 bags	18
	Chang	Beverages	24 - 12 oz bottles	19
	Aniseed Syrup	Condiments	12 - 550 ml bottles	10
	Chef Anton's Cajun Seaso	Condiments	48 - 6 oz jars	22
	Chef Anton's Gumbo Mix	Condiments	36 boxes	21.35
	Grandma's Boysenberry Sp	Condiments	12 - 8 oz jars	25
	Uncle Bob's Organic Driec	Produce	12 - 1 lb pkgs.	30
	Northwoods Cranberry Sa	Condiments	12 - 12 oz jars	40
	Mishi Kobe Niku	Meat/Poultry	18 - 500 g pkgs.	97

この図は、「クイックスタート」トピックからの抜粋です。

C1FlexGrid コントロールでは、2つの方法で列を作成することができます。つまり、コントロールに自動的に列を生成させるか、列を独自に作成します。1つのプロパティ(**AutoGenerateColumns**)を使用して、このオプションを制御します。

AutoGenerateColumns="True"

XAML に列を独自に生成する必要がない場合、または XAML に列を独自に生成しない場合は、AutoGenerateColumns プロパティを True に設定します。これで、**C1FlexGrid** コントロールによってグリッドに列が作成されます。次の例は、「**クイックスタート**」で行う設定と同様です。

XAML

```
<Page xmlns:FlexGrid="using:C1.Xaml.FlexGrid"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:FlexGridQSTest"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:Xaml="using:C1.Xaml"
    x:Class="FlexGridQSTest.MainPage"
    mc:Ignorable="d">
    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
```

```
        <RowDefinition/>
    </Grid.RowDefinitions>
   <FlexGrid:C1FlexGrid x:Name="flexgrid1" AutoGenerateColumns="True"
AllowResizing="Both" AllowDragging="Columns" AllowDrop="True"
ColumnHeaderForeground="White"  />
    </Grid>
</Page>
```

AutoGenerateColumns="False"

独自の列を定義する場合は、**AutoGenerateColumns** プロパティを False に設定します。マークアップは次の例のようになります。この例では、**Columns**を連結しています。

XAML

```
<Page xmlns:FlexGrid="using:C1.Xaml.FlexGrid"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:FlexGridGroupTest"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:Xaml="using:C1.Xaml"
    x:Class="FlexGridGroupTest.MainPage"
    mc:Ignorable="d">
    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
        </Grid.RowDefinitions>

        <FlexGrid:C1FlexGrid x:Name="c1FlexGrid1" AutoGenerateColumns="False"
ShowOutlineBar="True" BorderThickness="1" >
            <FlexGrid:C1FlexGrid.Columns>
                <FlexGrid:Column Binding="{Binding ID}" Format="g0"/>
                <FlexGrid:Column Binding="{Binding Name}" />
                <FlexGrid:Column Binding="{Binding Country}" />
                <FlexGrid:Column Binding="{Binding First}" />
                <FlexGrid:Column Binding="{Binding Last}" />
                <FlexGrid:Column Binding="{Binding Active}"/>
                <FlexGrid:Column Binding="{Binding Weight}" Format="n2"/>
                <FlexGrid:Column Binding="{Binding Hired}" Format="d"/>
                <FlexGrid:Column Binding="{Binding Father}" />
                <FlexGrid:Column Binding="{Binding Brother}" />
            </FlexGrid:C1FlexGrid.Columns>
        </FlexGrid:C1FlexGrid>
    </Grid>
</Page>
```

C1FlexGrid の使い方

データの集計

グリッドに販売データなどのデータを表示する場合に、国別または製品別の販売データなど、特定のカテゴリを集計したいことがあります。

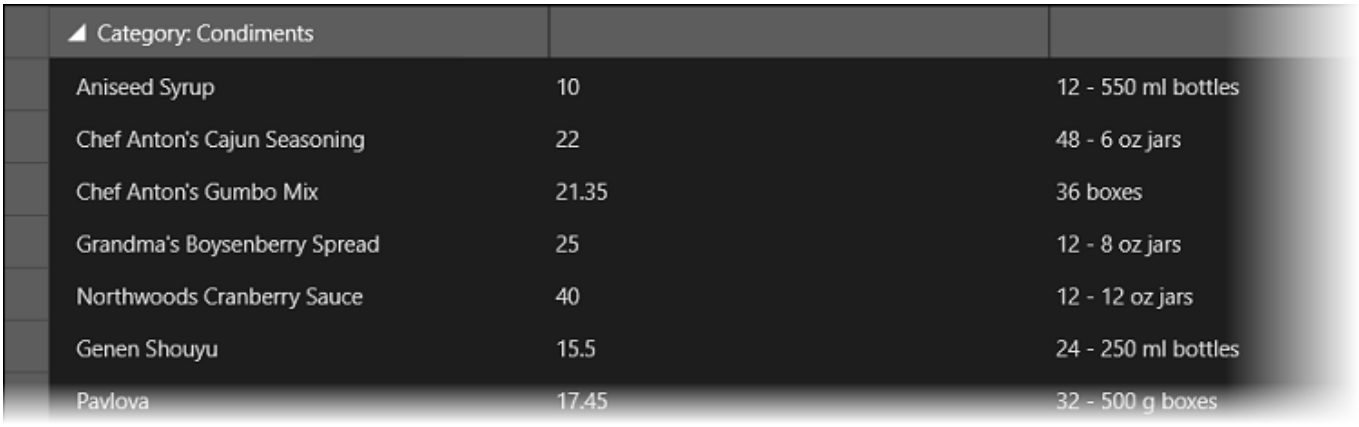
それは、**C1FlexGrid** コントロールを使用して簡単に行うことができます。XAML マークアップまたはコードのいずれかで、集計する列の GroupAggregate プロパティを設定するだけです。このプロパティを設定すると、C1FlexGrid が自動的に集計を行い、その値がグループヘッダー行に表示されます。

次のグリッド定義で説明します。

XAML

```
<c1:C1FlexGrid x:Name="c1FlexGrid1" AutoGenerateColumns="False" ShowOutlineBar="True"
BorderThickness="1">
<c1:C1FlexGrid.Columns>
<c1:Column Binding="{Binding Name}" Width="*"/>
<c1:Column Binding="{Binding Line}" />
<c1:Column Binding="{Binding Color}" />
<c1:Column Binding="{Binding Price}" Format="c2" Width="*"/>
<c1:Column Binding="{Binding Weight}" Format="n2" Width="*" />
<c1:Column Binding="{Binding Cost}" Format="c2" Width="*"/>
<c1:Column Binding="{Binding Volume}" Width="*"/>
<c1:Column Binding="{Binding Rating}" Format="n2" Width="*"/>
<c1:Column Binding="{Binding Discontinued}" Width="*"/>
</c1:C1FlexGrid.Columns>
</c1:C1FlexGrid>
```

製品種目別にデータをグループ化するように構成された C1CollectionView オブジェクトを **ItemsSource** プロパティに設定すると、C1FlexGrid で項目をグループ化できます。データを集計する前に、グループ化の方法を確認するには、このリンクを参照してください。



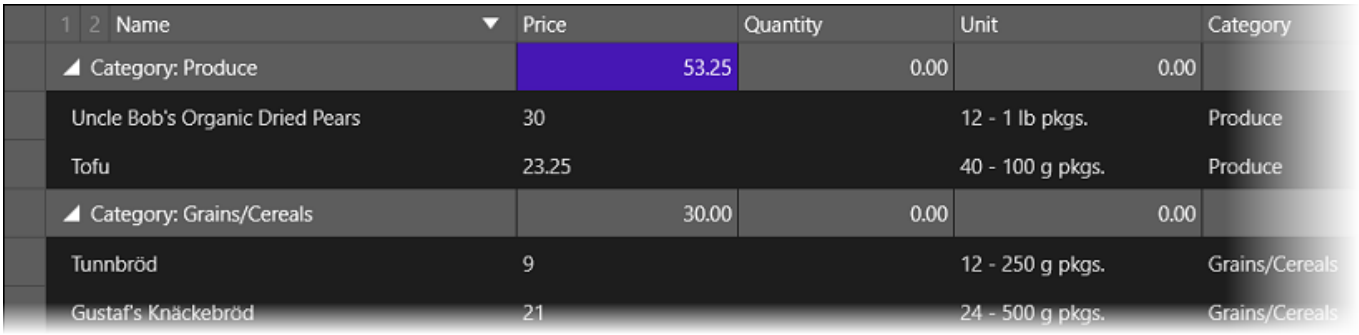
FlexGrid は、項目をグループ化できるだけではありません。[Price]、[Weight]、[Cost]、[Volume]、[Rating]の各列の合計を表示する場合、XAML マークアップは次のようになります。

XAML

```
<c1:C1FlexGrid x:Name="c1FlexGrid1" AutoGenerateColumns="False" ShowOutlineBar="True"
BorderThickness="1">
<c1:C1FlexGrid.Columns>
<c1:Column Binding="{Binding Name}" Width="*"/>
```

```
<c1:Column Binding="{Binding Line}" />
<c1:Column Binding="{Binding Color}" />
<c1:Column Binding="{Binding Price}" Format="c2" GroupAggregate="Sum" Width="*"/>
<c1:Column Binding="{Binding Weight}" Format="n2" GroupAggregate="Sum" Width="*" />
<c1:Column Binding="{Binding Cost}" Format="c2" GroupAggregate="Sum" Width="*"/>
<c1:Column Binding="{Binding Volume}" GroupAggregate="Sum" Width="*"/>
<c1:Column Binding="{Binding Rating}" Format="n2" GroupAggregate="Average"
Width="*"/>
<c1:Column Binding="{Binding Discontinued}" Width="*"/>
</c1:C1FlexGrid.Columns>
</c1:C1FlexGrid>
```

[Rating]列は、平均値を集計するように設定されています。このように変更すると、C1FlexGrid コントロールは次の図のようになります。



列ヘッダーに各列の集計値が表示されます。

セル結合

グリッドの **AllowMerging** プロパティは、グリッドレベルのセル結合を有効にします。グリッドレベルの結合を有効にしたら、**Row.AllowMerging** および **Column.AllowMerging** プロパティを使用して、結合する特定の行および列を選択します。

たとえば、次のコードは、同じ名前を含むセルを結合します。

Visual Basic

```
' スクロール可能な領域での結合を有効にします
c1FlexGrid1.AllowMerging = AllowMerging.Cells

' "Name" 列
c1FlexGrid1.Columns("Name").AllowMerging = True
```

C#

```
// スクロール可能な領域での結合を有効にします
    c1FlexGrid1.AllowMerging = AllowMerging.Cells;

// "Name" 列
    c1FlexGrid1.Columns["Name"].AllowMerging = true;
```

カスタムアイコン

FlexGrid displays various icons during its operations such as sorting, filtering etc. These icons can be changed using

various icon templates provided in the FlexGrid control. These icon templates can be accessed through following properties.

Properties	Description
SortAscendingIconTemplate	Allows you to set the template of sort icon for sorting values in ascending order.
SortDescendingIconTemplate	Allows you to set the template of sort icon for sorting values in descending order.
ExpandedAboveIconTemplate	Allows you to set the template for icon when the item group/detail is expanded above.
ExpandedBelowIconTemplate	Allows you to set the template for icon when the item group/detail is expanded below.
CollapsedIconTemplate	Allows you to set the template for group icon when the item group/detail is collapsed.
NewRowIconTemplate	Allows you to set the template of new row icon displayed in the header of a new row.

You can change the icons set by these templates either to the built-in icons provided by the FlexGrid or to your own custom image, geometric figures, font etc as an icon.

The following image displays a custom image which is set as a sort icon for sorting values in descending order.

ID	Name	CountryID	Active	First	Last
56	Zeb Myers	Italy	☐	Zeb	Myers
64	Zeb Myers	Ethiopia	☑	Zeb	Myers
87	Xavier Griswold	Congo	☑	Xavier	Griswold
11	Vic Ulam	Russia	☑	Vic	Ulam
24	Vic Stevens	Vietnam	☑	Vic	Stevens
84	Vic Richards	Egypt	☐	Vic	Richards
65	Vic Quaid	Brazil	☑	Vic	Quaid
58	Vic Krause	United Kingdoı	☐	Vic	Krause
93	Vic Jammers	Russia	☐	Vic	Jammers
71	Vic Griswold	Nigeria	☐	Vic	Griswold
32	Vic Griswold	Pakistan	☐	Vic	Griswold
0	Vic Evers	Thailand	☑	Vic	Evers
23	Ulrich Krause	China	☐	Ulrich	Krause
74	Ulrich Bishop	Mexico	☑	Ulrich	Bishop
90	Ted Trask	Myanmar	☑	Ted	Trask

FlexGrid also allows you to change the appearance of the different icons used in the control using the **C1Icon** class. The C1Icon class is an abstract class that provides a series of different objects that can be used for displaying monochromatic icons which can easily be tinted and resized.

Using built-in Icons

To set the built-in icons for the abovementioned templates, you can set the following properties of the

C1IconTemplate class.

Icon	Image
Edit	
Asterisk	
ArrowUp	
ArrowDown	
ChevronUp	
ChevronDown	
ChevronLeft	
ChevronRight	
TriangleNorth	
TriangleSouth	
TriangleEast	
TriangleWest	
TriangleSouthEast	
Star5	

For instance, to change the default sort ascending icon to a built-in icon, for example, TriangleNorth, use the following code:

C#

```
grid.SortAscendingIconTemplate = C1IconTemplate.TriangleNorth;
```

Using Custom Icons

FlexGrid also allows you to set your own custom image, font, or path as an icon through the respective classes.

Icon Type	Icon Class Name
Bitmap/Image	C1BitmapIcon class
Font character	C1FontIcon class
SVG	C1PathIcon class (child class of C1VectorIcon class)

For instance, to change the default sort descending icon to a custom image, use the following code:

C#

```
_flexGrid.SortDescendingIconTemplate = new C1IconTemplate(() => new C1BitmapIcon()
{
     Source = new BitmapImage() { UriSource = new Uri("ms-
appx:///Assets/arrow_down.png") },
     Width = 20,
     Height = 20
});
```

カスタムセル

Microsoft のデータグリッドコントロール（WinForms、Silverlight、または WPF）を使用したことがあればわかるように、大きなカスタマイズを実行するためには、コードを使用してカスタム **Column** オブジェクトを作成し、いくつかのメソッドをオーバーライドし、カスタム列をグリッドに追加する必要があります。**C1FlexGrid** コントロールは、これとは大きく異なる方法を使用します。コードでは単純なセルファクトリを使用してカスタムセルを生成し、XAMLでは標準のデータテンプレートを使用できます。

「コードでのカスタムセル」では、**ICellFactory** インタフェースを作成する方法を説明します。

「XAMLでのカスタムセル」では、**Column**オブジェクトのプロパティを使用してカスタムビジュアル要素を定義する例を示します。

コードでのカスタムセル

既に ICellFactory クラスについて理解している場合は、ここをクリックして、**C1FlexGrid CellFactory** の実装に進むことができます。

このグリッドには、グリッドに表示されるすべてのセルを作成する CellFactory クラスが含まれます。カスタムセルを作成するには、**ICellFactory** インタフェースを実装するクラスを作成し、このクラスをグリッドの **CellFactory** プロパティに割り当てる必要があります。カスタム列と同様に、カスタム **ICellFactory** クラスは、極めて特殊なアプリケーション固有のクラスである場合も、一般的で再利用可能な構成可能なクラスである場合も考えられます。通常、カスタム **ICellFactory** クラスは、セルを直接処理するため、カスタム列よりもかなり簡単になります（カスタム列は、列自体のほか、その列に含まれるセルなどの他のオブジェクトを処理する必要があります）。

ICellFactory インタフェースは極めて単純です。

C#

```
public interface ICellFactory
        {
         FrameworkElement CreateCell(
         C1FlexGrid grid,
         CellType cellType,
         CellRange range);

         FrameworkElement CreateCellEditor(
         C1FlexGrid grid,
         CellType cellType,
         CellRange range)

         void DisposeCell(
         C1FlexGrid grid,
         CellType cellType,
         FrameworkElement cell);
```

```
        }
```

ICellFactory インタフェースは、次の3つのメソッドを呼び出します。

- **CreateCell**
  このメソッドは、セルの表示に使用される FrameworkElement オブジェクトを作成します。このパラメータには、セルを所有するグリッド、作成するセルの種類、および複数のセルで表される CellRange が含まれます。
- **CreateCellEditor**
  このメソッドは、編集モードでセルを作成する以外は、最初のメソッドと同じです。
- **DisposeCell**
  このメソッドは、グリッドからセルが削除された後で呼び出されます。これにより、呼び出し元は、セルオブジェクトに関連付けられているすべてのリソースを破棄できます。

カスタムセルを使用する際は、グリッドセルが一時的であることを理解しておく必要があります。セルは、ユーザーがグリッドの範囲をスクロール、ソート、選択するたびに作成され、破棄されます。このプロセスは「仮想化」と呼ばれ、Windows Store アプリケーションではごく一般的です。仮想化を使用しないと、グリッドは通常、数千個ものビジュアル要素を同時に作成しなければならなくなり、パフォーマンスが低下します。

実装

カスタム **ICellFactory** クラスは、C1FlexGrid に含まれるデフォルトの CellFactory クラスから継承できるため、比較的簡単に実装することができます。

デフォルトの CellFactory クラスは、拡張可能なクラスになるように設計されているため、セル作成の詳細処理はすべてこのクラスに任せて、必要な部分だけをカスタマイズすることができます。次のコードは、安値を赤色、高値を緑色にする条件付き書式設定を適用します。

C#

```
c1FlexGrid1.CellFactory = new ConditionalFormattingFactory();
       class ConditionalFormattingFactory : CellFactory
            {
       // 安値と高値を示すために使用するブラシを作成します
        static Brush _brLowValue = new SolidColorBrush(Colors.Red);
        static Brush _brHighValue = new SolidColorBrush(Colors.Green);

       // セルの値に基づいてカスタムブラシを適用するためにオーバーライドされます
         public override void ApplyCellStyles(C1FlexGrid grid, CellType cellType,
             CellRange range, Border bdr)
             {
       // データセルだけを対象とします（ヘッダーは対象外）
         if (cellType == CellType.Cell)
             {
       // double 値だけを対象とします
         var col = grid.Columns[range.Column];
         if (col.DataType == typeof(double))
             {
       // セル値を取得します
         var value = (double)grid[range.Row, col];

       // 値が範囲外の場合の書式設定を適用します
         if (value < 100 || value > 1000)
             {
             var tb = bdr.Child as
            TextBlock;
             if (tb != null)
```

```
            {
            bdr.Background = value < 100 ? _brLowValue : _brHighValue;
                    }
                }
            }
        }
    }
```

XAMLでのカスタムセル

コードを記述する代わりに、XAML によってカスタムセルを作成することもできます。C1FlexGrid の **Column** オブジェクトには、**CellTemplate** および **CellEditingTemplate** プロパティが含まれます。これらのプロパティを使用して、列内のセルを表示および編集するためのビジュアル要素を指定できます。

たとえば、次の XAML コードは、列内の値を表示および編集するために使用されるカスタムビジュアル要素を定義します。この列内のセルは、緑色、太字、中央揃えのテキストで表示されます。テキストは、テキストボックスの横にある編集アイコンを使用して編集します。

マークアップ

```
<c1:C1FlexGrid x:Name="_fgTemplated">
<c1:C1FlexGrid.Columns>

<!-- テンプレート列を追加します -->
<c1:Column ColumnName="_colTemplated" Header="Template" Width="200">

<!-- 表示モードのセルのテンプレート -->
<c1:Column.CellTemplate>
<DataTemplate>
<TextBlock Text="{Binding Name}"
Foreground="Green" FontWeight="Bold"
VerticalAlignment="Center"/>
</DataTemplate>
</c1:Column.CellTemplate>

<!-- 編集モードのセルのテンプレート -->
<c1:Column.CellEditingTemplate>
<DataTemplate>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Image Source="edit_icon.png" Grid.Column="0" />
<TextBox Text="{Binding Name, Mode=TwoWay}" Grid.Column="1" />
</Grid>
</DataTemplate>
</c1:Column.CellEditingTemplate>
</c1:Column>
</c1:C1FlexGrid.Columns>
</c1:C1FlexGrid>
```

編集機能

C1FlexGrid は、Windows ストア用の使いやすいインタフェースによるセルの編集をサポートします。編集は、Web アプリケーションやデスクトップアプリケーションのようにインラインではなく、タッチインタフェースに最適な独立したポップアップで行われます。編集を開始するには、編集するセルをダブルクリックまたはタップします。セルは次の図のように表示されます。

	製品名	種別	単位	単価
	Chai	Beverages	10 boxes x 20 bags	18
	Chang	Beverages	24 - 12 oz bottles	19
	Aniseed Syrup	Condiments	12 - 550 ml bottles	10
	Chef Anton's Cajun Seaso	Condiments	48 - 6 oz jars	22
	Chef Anton's Gumbo Mix	Condiments	36 boxes	21.35
	Grandma's Boysenberry S	Condiments	12 - 8 oz jars	25
	Uncle Bob's Organic Driec	Produce	12 - 1 lb pkgs.	30
	Northwoods Cranberry Sa	Condiments	12 - 12 oz jars	40
	Mishi Kobe Niku	Meat/Poultry	18 - 500 g pkgs.	97

カスタムエディタ

C1FlexGrid は、2つの組み込みエディタを備えています。1つはブール値のチェックボックスで、もう1つは、前述したオートコンプリートとリスト選択によって標準の TextBox を拡張した **C1FlexComboBox** です。

このドキュメントで前述したカスタムセルを作成する場合と同じメカニズムを使用して、独自のエディタを作成および使用できます。

- カスタム **CellFactory** クラスを実装し、**CreateCellEditor** メソッドをオーバーライドして、エディタを作成して基底のデータ値に連結します。
- XAML を使用して、カスタムエディタを必要とする列に対して **CellEditingTemplate** を指定します。

オートコンプリートとマップされた列

オートコンプリートとマップされた列は、ColumnValueConverter と呼ばれる組み込みクラスを使用して実装されます。このクラスは、一般的な3つの結合シナリオを処理します。

オートコンプリートの排他モード(リストボックス形式の編集)

いくつかの特定の値のみを受け入れる列。たとえば、文字列型の "住所" 列と国名のリストがある場合に、ユーザーが国名のリストからのみ国を選択でき、リストにない値を入力できません。このシナリオは、2行のコードを使用して処理できます。

マークアップ

```
var c = flexgrid1.Columns["Country"];
c.ValueConverter = new ColumnValueConverter(GetCountryNames(), true);
```

ColumnValueConverter コンストラクタの最初のパラメータは、有効な値のリストです。2番目のパラメータは、ユーザーがリストにない値を入力できるようにするかどうかを設定します(この例では、入力できないようにします)。

オートコンプリートの非排他モード(コンボボックス形式の編集)

いくつかの一般的な値を含み、それ以外の値も受け入れる列。たとえば、文字列型の "住所" 列がある場合に、ユーザーが簡単に選択できるように一般的な国名のリストを提供することにします。ただし、この場合は、ユーザーがリストにない値も入力できるようにします。このシナリオも、2行のコードを使用して処理できます。

マークアップ

```
var c = flexgrid1.Columns["Country"];
    c.ValueConverter = new ColumnValueConverter(GetCountryNames(), false);
```

同様に、**ColumnValueConverter** コンストラクタの最初のパラメータは、有効な値のリストです。この場合、2番目のパラメータは、ユーザーがリストにない値を入力できるように、リストを排他的にしないように設定します。

データマップされた列

データマップされた列には、実際の値ではなく、キーが含まれます。たとえば、列の内容は国の 都道府県コード を表す整数であっても、対応する国名をユーザーが表示および編集できるようにします。

このシナリオでは、2行を超えるコードが必要になります。

C#

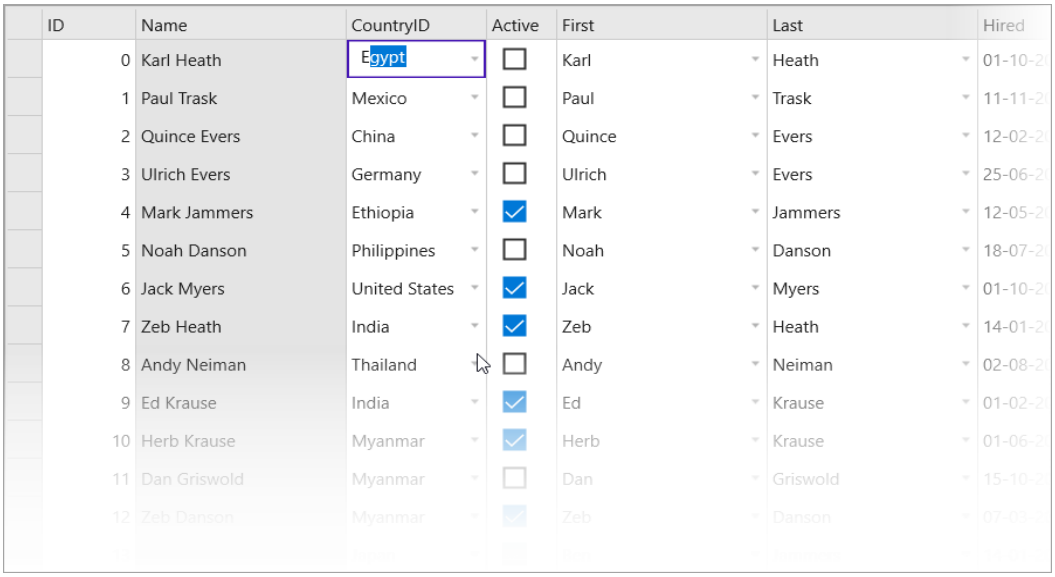
```
// IDsの代わりに国名を表示するようにCountryID列をマップします
Dictionary<int, string> dct = new Dictionary<int, string>(); foreach (var country in
Customer.GetCountries()) {     dct[dct.Count] = country; } col =
_flexEdit.Columns["CountryID"]; col.ValueConverter = new ColumnValueConverter(dct);
col.HorizontalAlignment = HorizontalAlignment.Left; col.Width = new GridLength(120)
```

このコードは、最初に、国の 都道府県コード 値（整数）を国名（文字列）にマップする辞書を構築します。

次に、その辞書を使用して ColumnValueConverter を構築し、前の例と同様に、そのコンバータを列の **ValueConverter** プロパティに割り当てます。

ユーザーは、辞書に存在するすべての国を選択でき、マップされていない値を入力できなくなります。

最後に、このコードは、列の配置を左揃えに設定します。この列の実際の内容は整数値なので、デフォルトでは右揃えで配置されます。ここでは、名前を表示するため、左揃えに設定します。



エディタの設定

組み込みエディタとカスタムエディタのどちらを使用する場合でも、**PrepareCellForEdit** イベントを利用することにより、エディタをアクティブ化する前に設定することができます。たとえば、次のコードは、選択範囲では背景が青色、文字が黄色になるようにエディタを変更します。

C#

```
// イベントハンドラを登録します
_grid.PrepareCellForEdit += _grid_PrepareCellForEdit;
// 選択範囲の外観を変更して、エディタをカスタマイズします
void _grid_PrepareCellForEdit(object sender, CellEditEventArgs e)
{
var b = e.Editor as Border;
var tb = b.Child as TextBox;
tb.SelectionBackground = new SolidColorBrush(Colors.Blue);
tb.SelectionForeground = new SolidColorBrush(Colors.Yellow);
}
```

データのフィルタ処理

C1CollectionView インタフェースは、Filter プロパティを使用したデータのフィルタ処理をサポートします。Filter プロパティは、コレクション内の各項目に対して呼び出されるメソッドを指定します。このメソッドが true を返す場合、その項目はビューに含まれます。このメソッドが false を返す場合、その項目はフィルタ処理されて非表示になります。この種類のメソッドは、述語メソッドです。

この ComponentOne のサンプルに付属する **FlexGridSamples** ソリューションには、ユーザーが検索する値を入力する **TextBox** コントロールとタイマーで構成される **SearchBox** コントロールが含まれています。このタイマーは、検索する値をユーザーが入力する際に、1文字ごとにフィルタが適用し直されることがないように、短時間の遅延を提供します。

ユーザーが入力を停止すると、このタイマーが次のコードを使用して動作し、フィルタを適用します。

C#

```
bool Filter(object item)
{
 // 検索テキストを取得します
 var srch = _txtSearch.Text;

 // テキストがない場合は、すべての項目を表示します
 if (string.IsNullOrEmpty(srch))
 {
 return true;
 }

 // 指定されたプロパティのいずれかにこのテキストが含まれている項目を表示します
 foreach (PropertyInfo pi in _propertyInfo)
 {
 var value = pi.GetValue(item, null) as string;
 if (value != null && value.IndexOf(srch, StringComparison.OrdinalIgnoreCase) > -1)
 {
 return true;
 }
 }

 // この項目を除外します...
```

```
 return false;
}
```

boolean 関数を使用して **Filter** プロパティの値を設定していることに注意してください。

この関数は、項目をパラメータとして受け取り、オブジェクトの指定されたプロパティの値を取得し、オブジェクトのいずれかのプロパティに検索対象の文字列が含まれる場合は true を返します。

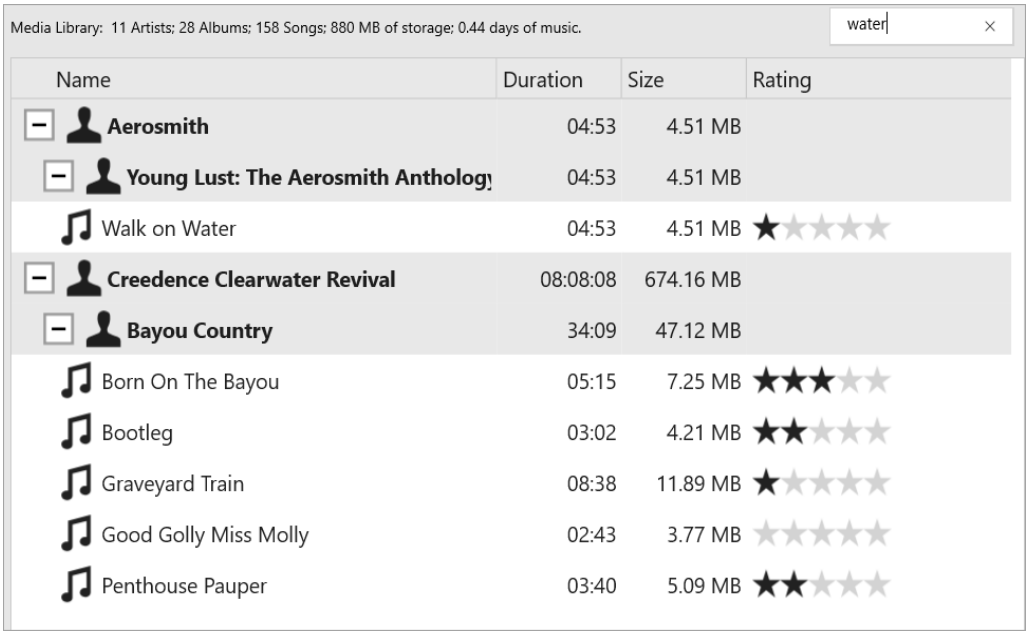
たとえば、オブジェクトの型が "Song" であり、指定されたプロパティが "Title"、"Album"、および "Artist" である場合、この関数は、曲のタイトル、アルバム、またはアーティスト内に検索対象の文字列が見つかった場合に true を返します。これは、強力で使いやすい検索メカニズムであり、Apple の iTunes アプリケーションで使用されているメカニズムに似ています。

フィルタが適用されると、グリッド（および **C1CollectionView** オブジェクトに連結されているその他すべてのコントロール）にはフィルタの結果が直ちに反映されて、フィルタによって選択された項目のみが表示されます。次のコードは、IC1CollectionViewのFilterプロパティを使用して曲をフィルタリングするために使用されます。

C#

```
var view = View as C1.Xaml.IC1CollectionView;
if (view != null)
{
  view.Filter = Filter;
  view.Refresh();
}
```

フィルタ処理とグループ化は、組み合わせて使用しても適切に機能することに注目してください。次の図に、非常に大きな曲のリストにフィルタが適用されている様子を示します。



この図は、フィルタが "Water" という単語に設定された際に取得したものです。フィルタは、すべてのフィールド（song、album、artist）で一致を検索します。このため、"Creedence Clearwater Revival" の曲はすべて自動的に含まれます。

フルテキストフィルタを使用してデータの検索

Searching a grid with huge data source can be a tedious task even in a grid that supports column filtering and sorting. To make this task easier, FlexGrid supports searching the entire data source connected to the grid through full text filtering. To apply full text filtering, you need to use **BindingPaths** property of the **FullTextFilterCondition** class, which specifies the data column(s) that will participate in the full text search filtering. For searching, you can use an editor control to act as a search box where user provides the input for obtaining the search results. You need to handle an event depending on whether you want to search as the user types in or once the input has completed. In that event, you can invoke **ApplyFullTextFilter** method of the **C1CollectionView** class which accepts input text and instance of the FullTextFilterCondition class as its parameters.

In addition, the **FullTextFilterCondition** class provides conditions to perform full text search filtering in the control. These conditions can be applied to the user input for customizing the filtering behavior using the **MatchCase**, **MatchWholeWord** and **TreatSpacesAsAndOperator** properties.

In the following example, we used a TextBox as a search box and invoked the ApplyFullTextFilter method in TextChanged event of the TextBox to implement full text search filtering in FlexGrid.

1. Add C1FlexGrid and TextBox control to the XAML designer and add five columns to the grid to display data using the following code:

XAML copyCode

```
<Grid>
    <TextBox x:Name="filterTextBox" HorizontalAlignment="Left" VerticalAlignment="Top"
             TextChanged="filterTextBox_TextChanged" Margin="0,20,0,0" Width="250"/>
    <Custom:C1FlexGrid x:Name="c1FlexGrid1" HorizontalAlignment="Left"
                       AutoGenerateColumns="false" ColumnHeaderForeground="Black"
                       Margin="0,60,0,0">
        <Custom:C1FlexGrid.Columns>
            <Custom:Column Binding="{Binding ID, Mode=TwoWay}"/>
            <Custom:Column Binding="{Binding Name, Mode=TwoWay}" Width="*"/>
            <Custom:Column Binding="{Binding Email, Mode=TwoWay}" Width="*" />
            <Custom:Column Binding="{Binding OrderDate, Mode=TwoWay}" Width="*"/>
            <Custom:Column Binding="{Binding OrderTotal, Mode=TwoWay}" Width="*"/>
        </Custom:C1FlexGrid.Columns>
    </Custom:C1FlexGrid>
</Grid>
```

2. Switch to the Code view and create a class, Customers, to add data in the FlexGrid.

Visual Basic

```
Public Class Customer
    Private _id As Integer
    Private _name, _email As String
    Private _OrderDate As DateTime
    Private _orderTotal As Double

    Shared _rnd As Random = New Random()
    Shared _firstNames As String() =
"Andy|Ben|Charlie|Dan|Ed|Fred|Herb|Elena|Stefan|Gina".Split("|"c)
    Shared _lastNames As String() =
"Ambers|Bishop|Cole|Danson|Evers|Frommer|Heath|Salvatore|Spencer|Rodriguez".Split("|"c)
    Shared _emailServers As String() = "gmail|yahoo|outlook|aol".Split("|"c)

    Public Sub New()
        Me.New(_rnd.[Next]())
    End Sub

    Public Sub New(ByVal id As Integer)
        ID = id
        Name = GetName()
        Email = String.Format("{0}@{1}.com", (Name.Substring(0, 1)).ToLower(),
GetString(_emailServers))
        OrderDate = DateTime.Today.AddDays(-_rnd.[Next](1, 365)).AddHours(_rnd.[Next](0,
24)).AddMinutes(
                                          _rnd.[Next](0, 60))
        OrderTotal = Math.Round(_rnd.NextDouble() * 10000.00, 2)
    End Sub

    Public Property ID As Integer
        Get
            Return _id
        End Get
        Set(ByVal value As Integer)

            If value <> _id Then
                _id = value
            End If
        End Set
    End Property

    Public Property Name As String
        Get
            Return _name
        End Get
        Set(ByVal value As String)

            If value <> _name Then
                _name = value
            End If
        End Set
    End Property
```

```
    Public Property Email As String
        Get
            Return _email
        End Get
        Set(ByVal value As String)

            If value <> _email Then
                _email = value
            End If
        End Set
    End Property

    Public Property OrderDate As DateTime
        Get
            Return _OrderDate
        End Get
        Set(ByVal value As DateTime)

            If value <> _OrderDate Then
                _OrderDate = value
            End If
        End Set
    End Property

    Public Property OrderTotal As Double
        Get
            Return _orderTotal
        End Get
        Set(ByVal value As Double)

            If value <> _orderTotal Then
                _orderTotal = value
            End If
        End Set
    End Property

    Private Shared Function GetString(ByVal arr As String()) As String
        Return arr(_rnd.[Next](arr.Length))
    End Function

    Private Shared Function GetName() As String
        Return String.Format("{0} {1}", GetString(_firstNames), GetString(_lastNames))
    End Function

    Public Shared Function GetCustomerList(ByVal count As Integer) As ObservableCollection(Of
Customer)
        Dim list = New ObservableCollection(Of Customer)()

        For i As Integer = 0 To count - 1
            list.Add(New Customer(i))
        Next

        Return list
    End Function
End Class
```

- **C#**

```
public class Customer
{
    //フィールド
    int _id;
    string _name, _email;
    DateTime _OrderDate;
    double _orderTotal;

    //データジェネレータ
    static Random _rnd = new Random();
    static string[] _firstNames = "Andy|Ben|Charlie|Dan|Eddy|Fred|Herb|Elena|Stefan|Gina".Split('|');
    static string[] _lastNames = "Ambers|Bishop|Cole|Danson|Evers|Frommer|Heath|Salvatore|Spencer|Rodriguez".Split('|');
    static string[] _emailServers = "gmail|yahoo|outlook|aol".Split('|');

    public Customer()
        : this(_rnd.Next())
    {
    }
    public Customer(int id)
    {
        ID = id;
        Name = GetName();
        Email = string.Format("{0}@{1}.com", (Name.Substring(0, 3)).ToLower(),
                GetString(_emailServers));
```

```
            OrderDate = DateTime.Today.AddDays(-_rnd.Next(1, 365)).AddHours(
                        _rnd.Next(0, 24)).AddMinutes(_rnd.Next(0, 60));
            OrderTotal = Math.Round(_rnd.NextDouble() * 10000.00, 2);
        }
        //オブジェクトモデル
        public int ID
        {
            get { return _id; }
            set
            {
                if (value != _id)
                {
                    _id = value;
                }
            }
        }
        public string Name
        {
            get { return _name; }
            set
            {
                if (value != _name)
                {
                    _name = value;
                }
            }
        }
        public string Email
        {
            get { return _email; }
            set
            {
                if (value != _email)
                {
                    _email = value;
                }
            }
        }
        public DateTime OrderDate
        {
            get { return _OrderDate; }
            set
            {
                if (value != _OrderDate)
                {
                    _OrderDate = value;
                }
            }
        }

        public double OrderTotal
        {
            get { return _orderTotal; }
            set
            {
                if (value != _orderTotal)
                {
                    _orderTotal = value;
                }
            }
        }

        // ** ユーティリティ
        static string GetString(string[] arr)
        {
            return arr[_rnd.Next(arr.Length)];
        }
        static string GetName()
        {
            return string.Format("{0} {1}", GetString(_firstNames), GetString(_lastNames));
        }

        public static ObservableCollection<Customer> GetCustomerList(int count)
        {
            var list = new ObservableCollection<Customer>();
            for (int i = 0; i < count; i++)
            {
                list.Add(new Customer(i));
            }
            return list;
        }
    }
```

3. Add the following code to filter data using full text filtering in the FlexGrid control.

Visual Basic

```
Public NotInheritable Partial Class MainPage
    Inherits Page

    Private cv As C1CollectionView
    Private FilterCondition As FullTextFilterCondition
```

```
    Public Sub New()
        Me.InitializeComponent()
        Dim list = New ObservableCollection(Of Customer)()

        For i As Integer = 0 To 100 - 1
            list.Add(New Customer())
        Next

        cv = New C1CollectionView(list)
        c1FlexGrid1.ItemsSource = cv
        FilterCondition = New FullTextFilterCondition()
        FilterCondition.BindingPaths = New List(Of String)() From {
            "ID",
            "Name",
            "OrderDate",
            "OrderTotal"
        }
        FilterCondition.MatchCase = True
    End Sub

    Private Sub filterTextBox_TextChanged(ByVal sender As Object, ByVal e As TextChangedEventArgs)
        cv.ApplyFullTextFilter(filterTextBox.Text, FilterCondition)
    End Sub
End Class
```

- **C#**

```
public sealed partial class MainPage : Page
{
    C1CollectionView cv;
    FullTextFilterCondition FilterCondition;
    public MainPage()
    {
        this.InitializeComponent();

        var list = new ObservableCollection<Customer>();
        for (int i = 0; i < 100; i++)
            list.Add(new Customer());

        // リストからC1CollectionViewを作成します
        cv = new C1CollectionView(list);

        c1FlexGrid1.ItemsSource = cv;

        FilterCondition = new FullTextFilterCondition();
        FilterCondition.BindingPaths = new List<string>() { "ID", "Name",
                    "OrderDate", "OrderTotal" };

        // FullTextFilterConditionを更新します
        FilterCondition.MatchCase = true;
    }
    void filterTextBox_TextChanged(object sender, TextChangedEventArgs e)
    {
        // FullTextFilterを適用します
        cv.ApplyFullTextFilter(filterTextBox.Text, FilterCondition);
    }
}
```

データのグループ化

このトピックでは、**FlexGridSamples** プロジェクトの **FlexGridDemo** ページにあるコードを使用します。このプロジェクトには、**Customer.cs** コードファイルも入っています。

C1FlexGrid コントロールでは、C1CollectionView クラスを通してデータのグループ化が行われます。たとえば、顧客リストを含むグリッドでは、C1CollectionView に少しのコードを追加して、顧客を国別にグループ化できます。

C#

```
C1.Xaml.C1CollectionView view;

    // 顧客の監視可能なリストを作成します
    var list = new System.Collections.ObjectModel.ObservableCollection();
    for (int i = 0; i < 100; i++)
        list.Add(new Customer());
```

```
    // リストから C1CollectionView を作成します
    view = new C1.Xaml.C1CollectionView(list);
    // 顧客を国別にグループ化します
    view.GroupDescriptions.Add(new C1.Xaml.PropertyGroupDescription("Country"));

    c1FlexGrid1.ItemsSource = view;
```

プロジェクトを実行すると、次の図のような FlexGrid が表示されます。

データ項目が国別にグループ化されています。ユーザーは国名の隣にあるアイコンをクリックまたはタップして、国カテゴリを折りたたんだり、展開することができます。また、列名をクリックまたはタップして、データのソートを制御することもできます。

グループ化を変更する場合は、次のコードで列名を変更するだけです。

C#

```
view.GroupDescriptions.Add(new
C1.Xaml.PropertyGroupDescription("Country"));
```

顧客の姓でソートする場合は、"Country" を "Last" に変更するだけです。

C#

```
 // 顧客の監視可能なリストを作成します

     var list = new System.Collections.ObjectModel.ObservableCollection<Customer>();
    for (int i = 0; i < 100; i++)
        list.Add(new Customer());

   // リストから C1CollectionView を作成します
   // （ソート、フィルタリング、およびグループ化をサポートしています）
   view = new C1.Xaml.C1CollectionView(list);

   // 顧客を国別にグループ化します
```

```
view.GroupDescriptions.Add(new C1.Xaml.PropertyGroupDescription("Last"));
```

グループ化を変更すると、FlexGrid コントロールは次の図のようになります。

複数セルの行および列ヘッダーの作成

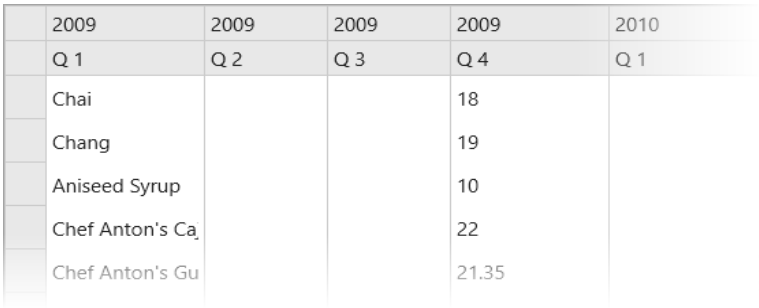
ほとんどのグリッドは、行および列ヘッダーのセルをサポートします。これにより、複数の列にまたがって1つのヘッダーを表示したり、選択範囲を含む行のステータスを示すことができます。

C1FlexGrid は、この概念をさらに広げて、複数セルのヘッダーをサポートします。たとえば、列ヘッダーを2行にして、1行に年度を表示し、もう1行に四半期を表示できます。この場合は、次のようにコードを設定します。

C#

```
// もう1つの列ヘッダー行を追加します
var ch = flexgrid1.ColumnHeaders;
ch.Rows.Add(new Row());
// ヘッダー行にデータを挿入します
for (int c = 0; c < ch.Columns.Count; c++)
{
ch[0, c] = 2009 + c / 4;
// 行0:年度
ch[1, c] = string.Format("Q {0}", c % 4 + 1);
// 行1:四半期
}
```

このコードは、次のようなグリッドを生成します。

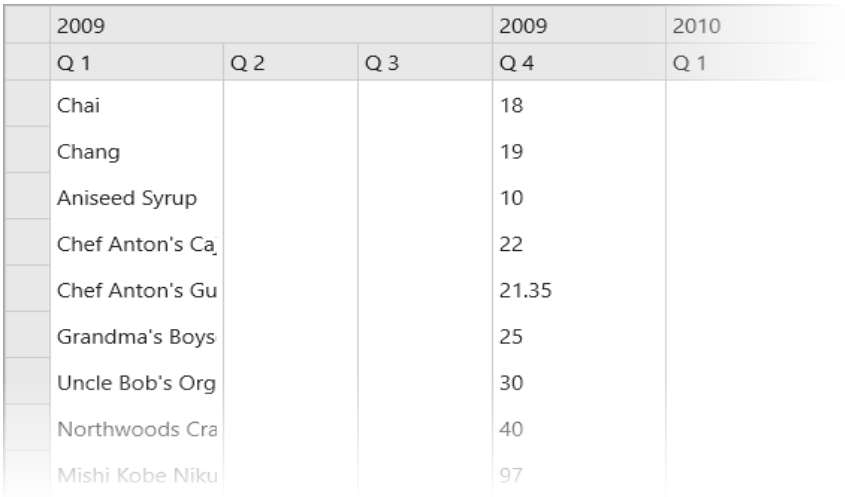


列ヘッダーを表示するために2つの行が使用されていることに注目してください。従来のグリッドでも、改行を含む列ヘッダーを使用して同様の効果を得られますが、上端の固定行に結合するようにセルを追加すると、同じ年度を参照する列が自動的に結合されるため、その違いは明らかです。このためには、コードを2行追加するだけです。

C#

```
// もう1つの列ヘッダー行を追加しま
var ch = flexgrid1.ColumnHeaders;
ch.Rows.Add(new Row());
// ヘッダー行にデータを挿入します
for (int c = 0; c < ch.Columns.Count; c++)
{
ch[0, c] = 2009 + c / 4;
// 行0:年度
ch[1, c] = string.Format("Q {0}", c % 4 + 1);
// 行1:四半期
// 上端の固定行を結合します
flexgrid1.AllowMerging = AllowMerging.All;
ch.Rows[0].AllowMerging = true;
```

この結果は、次の図のようになります。



選択モード

ほとんどのグリッドコントロールでは、表形式でデータを表示するだけでなく、マウスとキーボードを使用してデータの一部を選択できます。

C1FlexGrid は、**SelectionMode** プロパティで制御される豊富な選択モードを備えています。次の選択モードを使用できます。

- **Cell**:

選択範囲は、1つのセルに対応します。
- **CellRange**:
  選択範囲は、1つのセル範囲(隣接セルのブロック)に対応します。
- **Row**:
  選択範囲は、1つの行全体に対応します。
- **RowRange**:
  選択範囲は、隣接行のセットに対応します。
- **ListBox**:
  選択範囲は、任意の行のセットに対応します(必ずしも隣接している必要はありません)。

デフォルトの SelectionMode は **CellRange** です。これは、一般的な Excel 形式の選択動作を提供します。個々のグリッドセルではなく、データ項目全体を選択する場合には、行ベースのオプションも役立ちます。

どちらの選択モードの場合も、グリッドは Selection プロパティによって現在の選択範囲を公開します。このプロパティは、現在の選択範囲を CellRange オブジェクトとして取得または設定します。

選択範囲の監視

グリッドは、ユーザーアクションまたはコードによって選択範囲が変更されるたびに、**SelectionChanged** イベントを発生させます。これにより、新しい選択範囲に対応することができます。

たとえば、次のコードは、選択範囲を監視し、選択範囲が変更されると、コンソールに情報を出力します。

C#

```
void _flex_SelectionChanged(object sender, CellRangeEventArgs e)
{
CellRange sel = _flex.Selection;
Console.WriteLine("selection: {0},{1} - {2},{3}",
sel.Row, sel.Column, sel.Row2, sel.Column2);
Console.WriteLine("selection content: {0}",
GetClipString(_flex, sel));
}
static string GetClipString(C1FlexGrid fg, CellRange sel)
{
var sb = new System.Text.StringBuilder();
for (int r = sel.TopRow; r <= sel.BottomRow; r++)
{
for (int c = sel.LeftColumn; c <= sel.RightColumn; c++)
{
sb.AppendFormat("{0}\t", fg[r, c].ToString());
}
sb.AppendLine();
}
return sb.ToString();
}
```

このコードは、選択範囲が変更されるたびに、現在の選択範囲を表す **CellRange** の座標をリストします。また、GetClipString メソッドを使用して、選択範囲のコンテンツを出力します。このメソッドは、このドキュメントで前述したグリッドのインデクサを使用して、選択された項目をループ処理することによって選択範囲の各セルのコンテンツを取得します。

GetClipString メソッドでループする場合は、**Row**、**Row2**、**Column**、**Column2** の各プロパティの代わりに、CellRange の **TopRow**、**BottomRow**、**LeftColumn**、**RightColumn** の各プロパティを使用します。このようにする必要があるのは、ユーザーが選択する際の動作(マウスを上に向かってドラッグするか下に向かってドラッグするか)に応じて、**Row** が **Row2** より大きくなる場合と小さくなる場合があるためです。

RowCollection.GetDataItems メソッドを使用すると、**Selection** から簡単に多くの有益な情報を抽出できます。このメソッド

は、CellRange に関連付けられているデータ項目のコレクションを返します。このコレクションを取得したら、LINQ を使用して、選択された項目に関する情報を抽出および要約できます。

たとえば、Customer オブジェクトのコレクションに連結されたグリッドに対して、次の **SelectionChanged** イベントの代替実装を考えます。

C#

```
void _flex_SelectionChanged(object sender, CellRangeEventArgs e)
{
// 選択された範囲内の顧客を取得します
var customers =
_flex.Rows.GetDataItems(_flex.Selection).OfType<Customer>();

// LINQ を使用して、選択された顧客の情報を抽出します
_lblSelState.Text = string.Format(
"{0} items selected, {1} active, total weight: {2:n2}",
customers.Count(),
(from c in customers where c.Active select c).Count(),
(from c in customers select c.Weight).Sum());
}
```

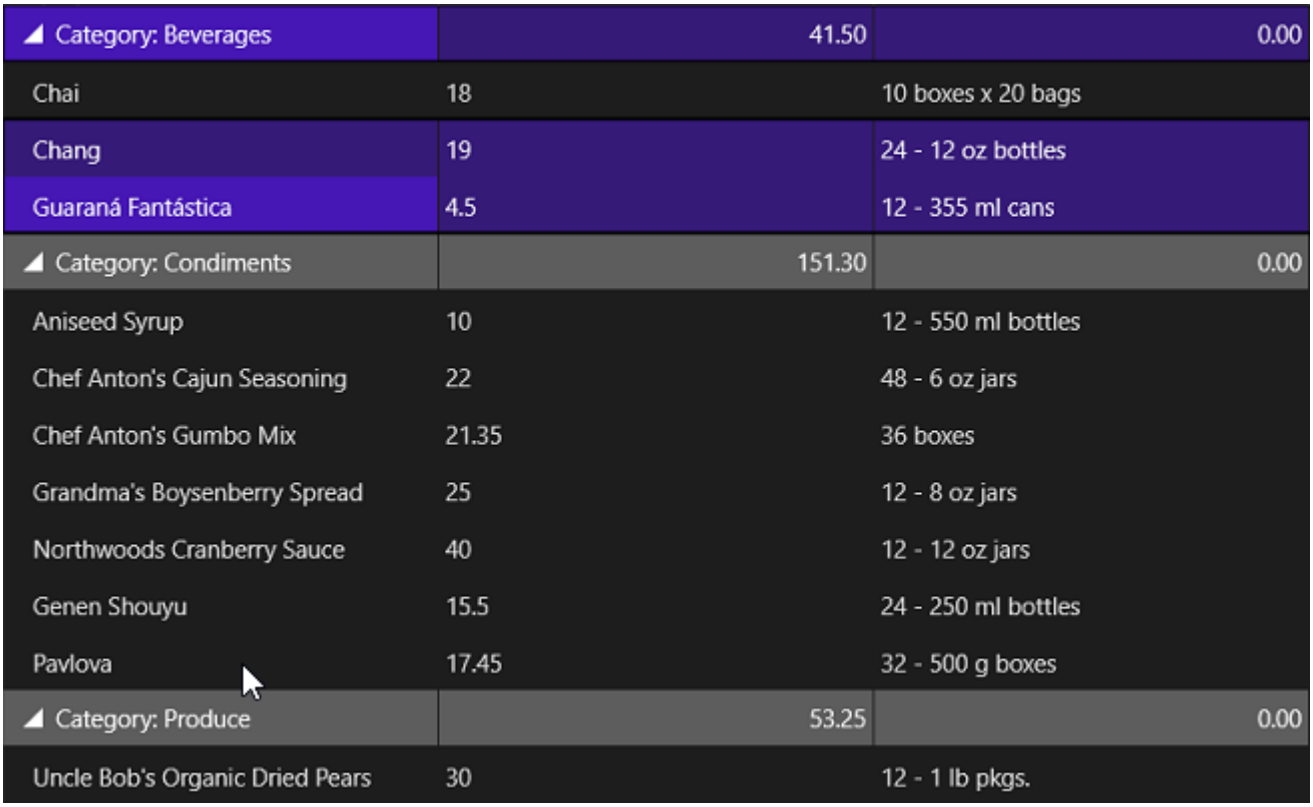
このコードでは、**OfType** 演算子を使用して、選択されたデータ項目を型 **Customer** にキャストします。この処理が完了すると、このコードは、LINQ を使用して、顧客の総数、"アクティブ" な顧客数、および選択範囲の顧客の総重量を取得します。LINQ は、このような処理に最適なツールです。柔軟性が高く表現力豊かで、コンパクトかつ効率的です。

選択範囲の表示のカスタマイズ

C1FlexGrid には、選択範囲を強調表示する方法をカスタマイズできる2つの機能が含まれています。

- **Excel 形式のマーキー**:**ShowMarquee** プロパティを true に設定すると、選択範囲を囲む四角形が自動的に描画されて、グリッドがとても見やすくなります。デフォルトでは、このマーキーは2ピクセルの濃い黒色の四角形ですが、**Marquee** プロパティを使用してカスタマイズすることができます。
- **選択されたセルのヘッダー**:グリッドの **ColumnHeaderSelectedBackground** および **RowHeaderSelectedBackground** プロパティにカスタムブラシオブジェクトを割り当てると、選択されたセルに対応するヘッダーが強調表示され、選択範囲を含む行または列を確認しやすくなります。

これらのプロパティを組み合わせることで、使い慣れた Excel のルックアンドフィールを持つグリッドを簡単に実装することができます。次の図に、例を示します。



Category: Beverages	41.50	0.00
Chai	18	10 boxes x 20 bags
Chang	19	24 - 12 oz bottles
Guaraná Fantástica	4.5	12 - 355 ml cans
Category: Condiments	151.30	0.00
Aniseed Syrup	10	12 - 550 ml bottles
Chef Anton's Cajun Seasoning	22	48 - 6 oz jars
Chef Anton's Gumbo Mix	21.35	36 boxes
Grandma's Boysenberry Spread	25	12 - 8 oz jars
Northwoods Cranberry Sauce	40	12 - 12 oz jars
Genen Shouyu	15.5	24 - 250 ml bottles
Pavlova	17.45	32 - 500 g boxes
Category: Produce	53.25	0.00
Uncle Bob's Organic Dried Pears	30	12 - 1 lb pkgs.

非連結モード

C1FlexGrid は、C1CollectionView データソースと組み合わせて使用することで、その機能を完全に活用できるように設計されています。

ただし、このグリッドは、非連結モードで使用することもできます。単にグリッドに行と列を追加する場合は、次に示すように一般的なインデックス表記を使用して、セルの値を取得または設定できます。

C#

```
        // 行/列を非連結グリッドに追加します
        for (int i = 0; i < 12; i++) // 4四半期から成る年を3年分
        {
         fg.Columns.Add(new Column());
        }
        for (int i = 0; i < 500; i++)
        {
         fg.Rows.Add(new Row());
        }

        // 非連結グリッドにデータを挿入します
        for (int r = 0; r < fg.Rows.Count; r++)
        {
         for (int c = 0; c < fg.Columns.Count; c++)
         {
         fg[r, c] = string.Format("cell [{0},{1}]", r, c);
         }
        }
```

C1FlexGrid ユーザーは、インデックス表記についても理解しておく必要があります。この表記は、このコントロールの WinForms バージョンで実装されるものと同じです。セルを指定するには、行インデックスと列インデックス、行インデックスと列名、または行インデックスと **>Column** オブジェクトを使用します。

インデックス表記は、連結モードと非連結モードで機能します。連結モードでは、データは、データソース内の項目から取得され、データソース内の項目に適用されます。非連結モードでは、データはグリッドによって内部的に保存されます。

UWP バージョンのコントロールでは、行インデックスと列インデックスに固定行と固定列が含まれません。次の図は、これを示しています。

	0,0	0,1	0,2
	1,0	1,1	1,2
	2,0	2,1	2,2

このような表記では、インデックスはデータ項目のインデックスと一致し（行0には項目0が含まれます）、列カウントは表示されるプロパティ数と一致するため、インデックス作成が容易になります。

ただし、このような表記を使用する場合、固定セルのコンテンツにアクセスするには、**RowHeader** プロパティと **ColumnHeader** プロパティが必要です。

たとえば、次のコードを使用して行ヘッダーをカスタマイズできます。

C#

```
// 非連結列ヘッダーを設定します
var ch = fg.ColumnHeaders;
ch.Rows.Add(new Row()); // 年度用のヘッダーと四半期用のヘッダー
for (int c = 0; c < ch.Columns.Count; c++)
{
ch[0, c] = 2009 + c / 4; // 年度
ch[1, c] = string.Format("Q {0}", c % 4 + 1); // 四半期
}

// 最初の固定行の結合を許可します
ch.Rows[0].AllowMerging = true;

// 非連結行ヘッダーを設定します
var rh = fg.RowHeaders;
rh.Columns.Add(new Column());
for (int c = 0; c < rh.Columns.Count; c++)
{
rh.Columns[c].Width = new GridLength(60);
for (int r = 0; r < rh.Rows.Count; r++)
{
rh[r, c] = string.Format("hdr {0},{1}", c == 0 ? r / 2 : r, c);
}
}
```

グリッドのコンテンツ領域を扱う際と同じオブジェクトモデルと手法を使用して、行および列ヘッダーをカスタマイズしてデータを挿入できます。

パフォーマンスの最適化

大規模なデータセットを **C1FlexGrid** コントロールに表示することは、一般的な要件です。UWP で大規模なデータセットを処理するために一般に使用される技術を UI の仮想化といいます。UI の仮想化では、項目が、ビューに表示されるまでは生成されません。つまり、ビューポート（ウィンドウまたはクライアント領域のサイズ）内で可視な数のセルだけが生成されます。**C1FlexGrid** コントロールは、垂直方向と水平方向のスクロールについて、UI の仮想化をデフォルトでサポートしています。

C1FlexGrid は、セルテンプレートに対してコンテナの再利用も行っています。これは、表示範囲外にスクロールされたセル要素を破棄するのではなく再利用することで、スクロールのパフォーマンスを最適化する機能です。カスタムセルテンプレートはコンテナの再利用をサポートしますが、セルファクトリはこれをサポートしていないことに注意してください。カスタムセルのパフォーマンスを向上させるには、セルファクトリではなく、セルテンプレートを簡略化された連結およびコンバータと組み合わせて使用することをお勧めします。詳細については、次のトピックを参照してください。

- セルテンプレートを使用して XAML でカスタムセルを作成する
- セルファクトリを使用してコードでカスタムセルを作成する

UI の仮想化とセルテンプレートの再利用は **C1FlexGrid** に付属し、余分な作業を行う必要なく、そのまま無償で使用できます。さらにパフォーマンスを向上させる必要がある場合は、テンプレートのインクリメンタルアップロードを検討してください。

- テンプレートのインクリメンタル更新は、ユーザーがスクロールする際に、複雑なカスタムセルを段階的にレンダリングするために使用できます。

テンプレートのインクリメンタル更新

CellContentChanging イベントによってテンプレートのインクリメンタル更新を行うと、セルテンプレートを段階的に更新できます。この方法では、一度にすべてレンダリングするのではなく、部品ごと、または段階的にレンダリングするため、ロード時間が短縮し、スクロールのパフォーマンスが向上します。

たとえば、グリッドを非常に高速でスクロールすると、セルの内容はプレーンテキストで表示されます。グリッドのスクロールが止まると、セルの内容が更新されて、複雑な計算を必要とする高負荷の内容が表示されます。このような内容は、スクロール中にはすべてのセルについてレンダリングする必要がありません。

以降の段階で、さらに高負荷のアクションを実行できます。これらのアクションは、UI スレッドが空いている場合に実行されます。これは、Microsoft GridView コントロールでの類似の機能に従ってモデル化されたものです。

> **メモ**: 実際の **CellContentChanging** イベントは、**Financial** サンプルで確認できます。Control Explorer では、**See it in action!** サンプルです。コンピュータの **Documents\ComponentOne Samples\UWP\C1.UWP.FlexGrid\CS** には **FlexGrid サンプル**がインストールされています。

CellContentChanging イベントは次のようになります。

C#

```
private void _flexFinancial_CellContentChanging(C1FlexGrid sender,
CellContentChanging
          EventArgsargs)
    {
     if (args.Phase == 0)
     {
     args.RegisterUpdateCallback(_flexFinancial_CellContentChanging);
     }
     else if (args.Phase == 1)
     {
     var factory = _flexFinancial.CellFactory as FinancialCellFactory;
     if (factory != null)
     factory.ShowLiveData(_flexFinancial, args.Range, args.Cell);
     }
    }
```

上記の **CellContentChanging** サンプルは、次の **ShowLiveData** イベントを呼び出します。

C#

```
public void ShowLiveData(C1FlexGrid grid, CellRange range, FrameworkElement cell)
    {
     var stockTicker = (cell as Border).Child as StockTicker;
     if (stockTicker != null)
     {
     var c = grid.Columns[range.Column];
     var r = grid.Rows[range.Row];
     var pi = c.PropertyInfo;
```

```
    // スパークラインを表示するには
    stockTicker.Tag = r.DataItem;
    stockTicker.BindingSource = pi.Name;

    var binding = new Binding { Path = new PropertyPath(pi.Name) };
    binding.Converter = new MyConverter();
    binding.Source = r.DataItem;
    binding.Mode = BindingMode.OneWay;
    stockTicker.SetBinding(StockTicker.ValueProperty, binding);
    }
  }
```

イベントで使用される StockTicker クラスは、スパークラインを表しています。これは、実装の中で高負荷の部分です。これを **CellContentChanging** イベントから呼び出すと、UI スレッドが空くまで **ShowLiveData** イベントは実行されなくなります。

印刷

The FlexGrid control provides the **Print** method to print the grid. This method has three overloads which takes parameters that allow you to specify the document name, page margins, scaling, maximum number of pages to print, and print parameters. The output is a faithful rendering of the grid, including all style elements, fonts, gradients, images, etc. Row and column headers are included on every page.

The following code uses Print method that opens a Print dialog directly for printing the FlexGrid:

Visual Basic

```
c1FlexGrid1.Print("FlexGrid_Print")
```

C#

```
c1FlexGrid1.Print("FlexGrid_Print");
```

行詳細テンプレート

FlexGrid では、行の詳細テンプレートを使用して各行に関する情報を柔軟に表示できます。行の詳細テンプレートは、詳細を表示するために各行に追加できるデータパネルです。テキスト、UI 要素、InputPanel などのデータ連結コントロールを行詳細テンプレートに埋め込むことができます。行ごとに、行のサマリーを表示するデータテンプレートを挿入し、グリッドのサイズに影響することなくテキストボックスなどの他のコントロールで詳細を表示/提供することができます。このテンプレートを使用して、グループ化されたデータを表示する階層グリッドを作成することもできます。

次の図に、行詳細テンプレートから表示される各行の詳細を示します。

FlexGridに行の詳細テンプレートを追加します

1. XAMLデザイナーにC1FlexGridコントロールを追加し、コントロールの名前を"grid"として設定します。
2. XAMLコードでは、<Grid>タグ内に3つの列を作成し、製品ID、製品名、注文日などのデータフィールドを表示します。

XAML copyCode

```
<FlexGrid:C1FlexGrid.Columns>
    <FlexGrid:Column Header="Product ID" Binding="{Binding ProductId}" Width="75" />
    <FlexGrid:Column Header="Product Name" Binding="{Binding ProductName}" Width="150" />
    <FlexGrid:Column Header="Order Date" Binding="{Binding OrderDate}" Width="300" />
</FlexGrid:C1FlexGrid.Columns>
```

3. StackPanel内にImageコントロールと6つのTextBlockコントロールを含めるための行の詳細テンプレートを作成します。

XAML copyCode

```
<FlexGrid:C1FlexGrid.RowDetailsTemplate>
    <DataTemplate>
        <StackPanel Background="GhostWhite">
            <Image HorizontalAlignment="Left" Name="img"
                   Source="{Binding ImgSource}"  Height="64" Margin="10" />
            <Grid Margin="0, 10">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="*" />
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                </Grid.RowDefinitions>
                <TextBlock Text="Product ID: " FontWeight="Bold" />
                <TextBlock Text="{Binding ProductId}" Grid.Column="1" />
                <TextBlock Text="Product Name: " FontWeight="Bold" Grid.Row="1" />
                <TextBlock Text="{Binding ProductName}" Grid.Column="1" Grid.Row="1" />
```

```
                <TextBlock Text="Order Date: " FontWeight="Bold" Grid.Row="2" />
                <TextBlock Text="{Binding OrderDate}" Grid.Column="1" Grid.Row="2" />
            </Grid>
        </StackPanel>
    </DataTemplate>
</FlexGrid:C1FlexGrid.RowDetailsTemplate>
```

4. プロジェクト名を右クリックし、メニューから**追加|新しいフォルダー**を選択して、**Resources**というフォルダを追加します。
5. **Resources** フォルダーにイメージを追加します。
6. **コード**ビューに切り替え、ProductDetailsというクラスを作成してFlexGridにデータを追加します。
   - **Visual Basic**

```
Public Class ProductDetails
    Public Property ProductId() As Integer
        Get
            Return m_ProductId
        End Get
        Set
            m_ProductId = Value
        End Set
    End Property
    Private m_ProductId As Integer
    Public Property ProductName() As String
        Get
            Return m_ProductName
        End Get
        Set
            m_ProductName = Value
        End Set
    End Property
    Private m_ProductName As String
    Public Property OrderDate() As DateTime
        Get
            Return m_OrderDate
        End Get
        Set
            m_OrderDate = Value
        End Set
    End Property
    Private m_OrderDate As DateTime
    Public Property ImgSource() As ImageSource
        Get
            Return m_ImgSource
        End Get
        Set
            m_ImgSource = Value
        End Set
    End Property
    Private m_ImgSource As ImageSource
End Class
```

   - **C#**

```
public class ProductDetails
{
    public int ProductId { get; set; }
    public string ProductName { get; set; }
    public DateTime OrderDate { get; set; }
    public ImageSource ImgSource { get; set; }
}
```

7. 製品のリストを作成し、そのリストにデータを追加し、**ItemsSource**プロパティを使用して、FlexGridコントロールをリストに連結します。
   - **Visual Basic**

```
'リストを作成します
Dim users As New List(Of ProductDetails)()
'リストに項目を追加します
users.Add(New ProductDetails() With {
  .ProductId = 101,
  .ProductName = "飲料",
  .OrderDate = New DateTime(1971, 7, 23, 8, 0, 20),
  .ImgSource = New BitmapImage(New Uri("ms-appx:///Resources/Beverage.png"))
```

```
})
users.Add(New ProductDetails() With {
  .ProductId = 102,
  .ProductName = "調味料",
  .OrderDate = New DateTime(1974, 1, 17, 12, 30, 10),
  .ImgSource = New BitmapImage(New Uri("ms-appx:///Resources/Condiments.png"))
})
users.Add(New ProductDetails() With {
  .ProductId = 103,
  .ProductName = "お菓子",
  .OrderDate = New DateTime(1991, 9, 2, 2, 42, 45),
  .ImgSource = New BitmapImage(New Uri("ms-appx:///Resources/Confections.png"))
})
users.Add(New ProductDetails() With {
  .ProductId = 104,
  .ProductName = "家禽",
  .OrderDate = New DateTime(1991, 10, 24, 6, 20, 49),
  .ImgSource = New BitmapImage(New Uri("ms-appx:///Resources/Poultry.png"))
})
'グリッドにデータを設定します
grid.ItemsSource = users
```

- **C#**

```
//リストを作成します
List<ProductDetails> users = new List<ProductDetails>();

//リストに項目を追加します
users.Add(new ProductDetails() { ProductId = 101, ProductName = "飲料",
  OrderDate = new DateTime(1971, 7, 23, 8, 0,20),
  ImgSource = new BitmapImage(new Uri("ms-appx:///Resources/Beverage.png"))});
users.Add(new ProductDetails() { ProductId = 102, ProductName = "調味料",
  OrderDate = new DateTime(1974, 1, 17, 12,30,10),
  ImgSource = new BitmapImage(new Uri("ms-appx:///Resources/Condiments.png"))});
users.Add(new ProductDetails() { ProductId = 103, ProductName = "お菓子",
  OrderDate = new DateTime(1991, 9, 2, 2,42,45),
  ImgSource = new BitmapImage(new Uri("ms-appx:///Resources/Confections.png"))});
users.Add(new ProductDetails() { ProductId = 104, ProductName = "家禽",
  OrderDate = new DateTime(1991, 10, 24, 6,20,49),
  ImgSource = new BitmapImage(new Uri("ms-appx:///Resources/Poultry.png"))});

//グリッドにデータを設定します
grid.ItemsSource = users;
```

チュートリアル

グリッドへのデータの挿入

C1CollectionView を実装して **C1FlexGrid** コントロールにデータを挿入する方法の詳細なウォークスルーについては、「データの挿入」を参照してください。

他の多くのグリッドと同様に、グリッドをアプリケーションに追加したら、IEnumerable インタフェースを実装するオブジェクトを受け取る ItemsSource プロパティを使用して、そのグリッドにデータを挿入します。ただし、通常はより高いレベルで作業するため、C1CollectionView インタフェースを実装するオブジェクトを使用します。

C1CollectionView クラスには、UWP ICollectionView にはないソート、フィルタ処理、グループ化、編集などの機能が用意されています。C1CollectionView は、使い慣れたオブジェクトモデルを提供する一方で、ICollectionView との差異を豊富なインタフェースで解消しています。

たとえば、**C1FlexGrid for UWP**アプリケーションに顧客オブジェクトのリストを表示するには、次のようなコードを使用します。

C#

```
List<Customer> customers = await GetCustomerData();
var view = new C1.Xaml.C1CollectionView(customers);
```

次に、これを C1FlexGrid コントロールに連結します。

C#

```
c1FlexGrid1.ItemsSource = view;
```

また、グリッドを顧客リストに直接連結することもできます。ただし、通常は C1CollectionView に連結することをお勧めします。これは、C1CollectionView がアプリケーションのデータ構成を数多く保持していること、および C1CollectionView はコントロール間で共有できるためです。

さまざまなコントロールを同じ C1CollectionView オブジェクトに連結した場合、コントロールは同じビューに表示されます。1つのコントロールで1つの項目を選択すると、他のコントロールでも同じ項目が自動的に選択されます。同じビューに連結されたすべてのコントロール間で、フィルタ処理、グループ化、またはソートを共有することもできます。

前述のコードを使用した場合、グリッドは、データソースをスキャンし、データソース項目のパブリックプロパティごとに列を自動的に生成します。自動的に作成される列は、コードを使用してカスタマイズできます。また、列の自動生成を完全に無効にしたり、コードまたは XAML を使用して独自の列を作成することもできます。

たとえば、次の XAML では、列の自動生成が無効に設定され、代わりに次のように列が指定されます。

XAML

```
<FlexGrid:C1FlexGrid x:Name="flexgrid1" AllowResizing="Both" AllowDragging="Both"
AllowDrop="True" AutoGenerateColumns="false" ColumnHeaderForeground="White" >
<FlexGrid:C1FlexGrid.Columns>
   <FlexGrid:Column Binding="{Binding Active, Mode=TwoWay}" />
   <FlexGrid:Column Binding="{Binding ID, Mode=TwoWay}" />
   <FlexGrid:Column Binding="{Binding Name, Mode=TwoWay}" Width="*"/>
   <FlexGrid:Column Binding="{Binding Country, Mode=TwoWay}" Width="*"/>
   <FlexGrid:Column Binding="{Binding Hired, Mode=TwoWay}" Format="d" Width="*" />
   <FlexGrid:Column Binding="{Binding Father, Mode=TwoWay}" Width="*"/>
   <FlexGrid:Column Binding="{Binding Weight, Mode=TwoWay}" Width="*"/>
</FlexGrid:C1FlexGrid.Columns>
</FlexGrid:C1FlexGrid>
```

これは、Microsoft DataGrid コントロールまたは元の C1DataGrid コントロールを使用して同じタスクを実行する際に使用する XAML に似ています。

XAML

```
<!-- create columns on an MSDataGrid (or C1DataGrid) -->
 <ms:DataGrid Name="_msSongs"
   AutoGenerateColumns="False" >
   <ms:DataGrid.Columns>
     <ms:DataGridTextColumn Binding="{Binding Name}" Header="Title"
         CanUserReorder="False" Width="300" />
     <ms:DataGridTextColumn Binding="{Binding Duration}" />
     <ms:DataGridTextColumn Binding="{Binding Size}" />
     <ms:DataGridTextColumn Binding="{Binding Rating}" Width="200" />
   </ms:DataGrid.Columns>
 </ms:DataGrid>
```

このように、マークアップは実質的に同じです。

連結は、グリッドの Columns コレクションへのインデクサとして使用することができます。たとえば、コードを使用して "Rating" 列の幅を 300 ピクセルに設定するには、次のように記述します。

C#

```
flexgrid1.Columns["Rating"].Width = new GridLength(300);
```

この構文は、C1FlexGrid ユーザーであれば使用経験があるはずです。これは、WPF バージョンのコントロールで作業する際に使用するコマンドと同じです。

データの挿入

このヘルプでは、別のコードファイルにあるデータを連結することでグリッドにデータを挿入する方法について説明します。また、**C1FlexGrid** コントロールのフィルタ処理およびグループ化機能についても具体的に説明します。

次の手順に従います。

1. MainPage.xaml ページをまだ開いていない場合は開きます。<Grid> タグと </Grid> タグの間をクリックし、次の XAML マークアップを挿入して、フィルタ処理とグループ化を制御する通常の ComboBox および TextBox コントロールを設定します。

   XAML でマークアップを書く場合

   XAML

   ```
   <Grid>
   <Grid.RowDefinitions>
   <RowDefinition Height="Auto"/>
   <RowDefinition Height="Auto"/>
   <RowDefinition />
   </Grid.RowDefinitions>
   <Border BorderThickness="0,1,0,1" BorderBrush="White" Margin="5">
   <StackPanel Orientation="Horizontal">
   <StackPanel.Resources>
   <Style
   TargetType="TextBlock">

   <Setter Property="FontSize" Value="14"/>
   ```

```
<Setter Property="Margin" Value="10,0,6,0"/>

<Setter Property="VerticalAlignment" Value="Center"/>
</Style>
</StackPanel.Resources>

<TextBlock Text="Group on:"
HorizontalAlignment="Right"/>
<ComboBox Name="groupComboBox"
Grid.Column="1" SelectionChanged="groupComboBox_SelectionChanged_1"

Height="30" Width="110" Margin="5"/>
<TextBlock Text="Filter on:" Grid.Row="1"
HorizontalAlignment="Right"/>
<ComboBox Name="filterComboBox" Grid.Row="1"
Grid.Column="1" SelectionChanged="filterComboBox_SelectionChanged_1"

Height="30" Width="110" Margin="5"/>
<TextBox Name="filterTextBox" Grid.Row="1"
Grid.Column="2" TextChanged="filterTextBox_TextChanged_1" Height="25"

Width="110" Margin="5"/>
</StackPanel>
</Border>
```

2. Visual Studio のツールボックスでコントロールをダブルクリックするか、次の XAML マークアップをアプリケーションに追加して、C1FlexGrid コントロールをアプリケーションに追加します。

XAML

```
<FlexGrid:C1FlexGrid x:Name="flexgrid1" AllowResizing="Both" AllowDragging="Both"
AllowDrop="True" ColumnHeaderForeground="White" />
```

3. <FlexGrid:C1FlexGrid> タグと </FlexGrid: C1FlexGrid> タグの間に次の XAML マークアップを追加して、**FlexGrid.Columns** を設定し、マークアップで連結を作成します。

XAML

```
<FlexGrid:C1FlexGrid.Columns>
<FlexGrid:Column Binding="{Binding Active, Mode=TwoWay}" />
<FlexGrid:Column Binding="{Binding ID, Mode=TwoWay}" />
<FlexGrid:Column Binding="{Binding Name, Mode=TwoWay}" Width="*"/>
<FlexGrid:Column Binding="{Binding Country, Mode=TwoWay}" Width="*"/>
<FlexGrid:Column Binding="{Binding Hired, Mode=TwoWay}" Format="d" Width="*" />
<FlexGrid:Column Binding="{Binding Father, Mode=TwoWay}" Width="*"/>
<FlexGrid:Column Binding="{Binding Weight, Mode=TwoWay}" Width="*"/>
</FlexGrid:C1FlexGrid.Columns>
```

4. アプリケーション名を右クリックし、リストから**[追加]**→**[新しい項目]**を選択します。**[新規ファイルの追加]**ダイアログボックスで**[コードファイル]**を選択し、**Customer.cs** と名前を付けます。新しいコードファイルがすぐに表示されます。
5. **Customer.cs** ファイルで次の名前空間とコードを追加して、グリッドに挿入するデータを作成します。

C# でコードを書く場合

C#

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Reflection;
```

```
namespace FlexGridSamples
{
public class Customer :
INotifyPropertyChanged,
IEditableObject
{
// ** フィールド
int _id, _countryID;
string _first, _last;
string _father, _brother, _cousin;
bool _active;
DateTime _hired;
double _weight;

// ** データジェネレータ
static Random _rnd = new Random();
static string[] _firstNames =
"Andy|Ben|Charlie|Dan|Ed|Fred|Gil|Herb|Jack|Karl|Larry|Mark|Noah|
Oprah|Paul|Quince|Rich|Steve|Ted|Ulrich|Vic|Xavier|Zeb".Split('|');
static string[] _lastNames =
"Ambers|Bishop|Cole|Danson|Evers|Frommer|Griswold|Heath|Jammers|
Krause|Lehman|Myers|Neiman|Orsted|Paulson|Quaid|Richards|Stevens|
        Trask|Ulam".Split('|');

static string[] _countries = "China|India|United
States|Indonesia|Brazil|Pakistan|Bangladesh|Nigeria|Russia|Japan|
Mexico|Philippines|Vietnam|Germany|Ethiopia|Egypt|Iran|Turkey|
        Congo|France|Thailand|United
Kingdom|Italy|Myanmar".Split('|');

// ** コンストラクタ
public Customer()
: this(_rnd.Next(10000))
{
}
public Customer(int id)
{
ID = id;

First = GetString(_firstNames);
Last = GetString(_lastNames);
CountryID = _rnd.Next() % _countries.Length;
Active = _rnd.NextDouble() >= .5;
Hired = DateTime.Today.AddDays(-_rnd.Next(1, 365));
Weight = 50 + _rnd.NextDouble() * 50;
_father = string.Format("{0} {1}", GetString(_firstNames),
Last);
_brother = string.Format("{0} {1}", GetString(_firstNames),
Last);
_cousin = GetName();
}

// ** オブジェクトモデル
public int ID
{
```

```
get { return _id; }
set
{
if (value != _id)
{
_id = value;

RaisePropertyChanged("ID");
}
}
}
public string Name
{
get { return string.Format("{0} {1}", First, Last); }
}
public string Country
{
get { return _countries[_countryID]; }
}
public int CountryID
{
get { return _countryID; }
set
{
if (value != _countryID && value
> -1 && value < _countries.Length)
{
_countryID =
value;


// OnPropertyChanged を Null パラメータで呼び出します。このプロパティを設定すると、
// "CountryID"
の値および "Country" の値が変更されるためです。

RaisePropertyChanged("");
}
}
}
public bool Active
{
get { return _active; }
set
{
if (value != _active)
{
_active =
value;

RaisePropertyChanged("Active");
}
}
}
public string First
{
get { return _first; }
```

```
set
{
if (value != _first)
{
_first =
value;


// OnPropertyChanged を Null パラメータで呼び出します。このプロパティを設定すると、
// "First" の値および
"Name" の値が変更されるためです。

RaisePropertyChanged("");
}
}
}
public string Last
{
get { return _last; }
set
{
if (value != _last)
{
_last = value;


// OnPropertyChanged を Null パラメータで呼び出します。このプロパティを設定すると、
// "First" の値および
"Name" の値が変更されるためです。

RaisePropertyChanged("");
}
}
}
public DateTime Hired
{
get { return _hired; }
set
{
if (value != _hired)
{
_hired =
value;

RaisePropertyChanged("Hired");
}
}
}
public double Weight
{
get { return _weight; }
set
{
if (value != _weight)
{
_weight =
```

```
value;

RaisePropertyChanged("Weight");
}
}
}

// 読み取り専用要素
public string Father
{
get { return _father; }
}
public string Brother
{
get { return _brother; }
}
public string Cousin
{
get { return _cousin; }
}

// ** ユーティリティ
static string GetString(string[] arr)
{
return arr[_rnd.Next(arr.Length)];
}
static string GetName()
{
return string.Format("{0} {1}", GetString(_firstNames),
GetString(_lastNames));
}

// ** 静的リストプロバイダ
public static ObservableCollection<Customer> GetCustomerList(int count)
{
var list = new ObservableCollection<Customer>();
for (int i = 0; i < count; i++)
{
list.Add(new Customer(i));
}
return list;
}

// ** 静的値プロバイダ
public static string[] GetCountries() { return _countries; }
public static string[] GetFirstNames() { return _firstNames; }
public static string[] GetLastNames() { return _lastNames; }

#region ** INotifyPropertyChanged Members

// このインタフェースは、連結コントロールがデータオブジェクトの変更に対応できるようにします。

void RaisePropertyChanged(string propertyName)
{
OnPropertyChanged(new
PropertyChangedEventArgs(propertyName));
```

```
}
public event PropertyChangedEventHandler PropertyChanged;
protected void OnPropertyChanged(PropertyChangedEventArgs e)
{
if (PropertyChanged != null)
PropertyChanged(this, e);
}

#endregion

#region IEditableObject Members

// このインタフェースは、トランザクション編集を許可します（ユーザーは[Esc]キーを押して直前の値を復元できます）。

Customer _clone;
public void BeginEdit()
{
_clone = (Customer)this.MemberwiseClone();
}
public void EndEdit()
{
_clone = null;
}
public void CancelEdit()
{
if (_clone != null)
{
foreach (var p in
this.GetType().GetRuntimeProperties())
{
if (p.CanRead
&& p.CanWrite)
{

p.SetValue(this, p.GetValue(_clone, null), null);
}
}
}
}

#endregion
}
}
```

Visual Basic でコードを書く場合

Visual Basic

```
Visual Basic
Imports System
Imports System.Collections
Imports System.Collections.Generic
Imports System.Collections.ObjectModel
Imports System.ComponentModel
Imports System.Reflection
```

```
Namespace FlexGridSamples
Public Class Customer
Implements INotifyPropertyChanged
Implements IEditableObject
' ** フィールド
Private _id As Integer, _countryID As Integer
Private _first As String, _last As String
Private _father As String, _brother As String, _cousin As String
Private _active As Boolean
Private _hired As DateTime
Private _weight As Double

' ** データジェネレータ
Shared _rnd As New Random()
Shared _firstNames As String() =
"Andy|Ben|Charlie|Dan|Ed|Fred|Gil|Herb|Jack|Karl|Larry|Mark|Noah|
        Oprah|Paul|Quince|Rich|Steve|Ted|Ulrich|Vic|Xavier|Zeb".Split("|"c)
Shared _lastNames As String() =
"Ambers|Bishop|Cole|Danson|Evers|Frommer|Griswold|Heath|Jammers|
        Krause|Lehman|Myers|Neiman|Orsted|Paulson|Quaid|Richards|
        Stevens|Trask|Ulam".Split("|"c)

Shared _countries As String() = "China|India|United
States|Indonesia|Brazil|Pakistan|Bangladesh|Nigeria|Russia|Japan|
        Mexico|Philippines|Vietnam|Germany|Ethiopia|Egypt|Iran|Turkey|
        Congo|France|Thailand|United
Kingdom|Italy|Myanmar".Split("|"c)

' ** コンストラクタ
Public Sub New()
Me.New(_rnd.[Next](10000))
End Sub
Public Sub New(id__1 As Integer)
ID = id__1

First = GetString(_firstNames)
Last = GetString(_lastNames)
CountryID = _rnd.[Next]() Mod _countries.Length
Active = _rnd.NextDouble() >= 0.5
Hired = DateTime.Today.AddDays(-_rnd.[Next](1, 365))
Weight = 50 + _rnd.NextDouble() * 50
_father = String.Format("{0} {1}", GetString(_firstNames),
Last)
_brother = String.Format("{0} {1}", GetString(_firstNames),
Last)
_cousin = GetName()
End Sub

' ** オブジェクトモデル
Public Property ID() As Integer
Get
Return _id
End Get
Set(value As Integer)
If value <> _id Then
_id = value
```

```
RaisePropertyChanged("ID")
End If
End Set
End Property
Public ReadOnly Property Name() As String
Get
Return String.Format("{0} {1}", First,
Last)
End Get
End Property
Public ReadOnly Property Country() As String
Get
Return _countries(_countryID)
End Get
End Property
Public Property CountryID() As Integer
Get
Return _countryID
End Get
Set(value As Integer)
If value <> _countryID AndAlso value
> -1 AndAlso value < _countries.Length Then
_countryID =
value

'
OnPropertyChanged を Null パラメータで呼び出します。このプロパティを設定すると、
' "CountryID"
の値および "Country" の値が変更されるためです。

RaisePropertyChanged("")
End If
End Set
End Property
Public Property Active() As Boolean
Get
Return _active
End Get
Set(value As Boolean)
If value <> _active Then
_active =
value

RaisePropertyChanged("Active")
End If
End Set
End Property
Public Property First() As String
Get
Return _first
End Get
Set(value As String)
If value <> _first Then
_first = value
```

```
'
OnPropertyChanged を Null パラメータで呼び出します。このプロパティを設定すると、
' "First" の値および
"Name" の値が変更されるためです。

RaisePropertyChanged("")
End If
End Set
End Property
Public Property Last() As String
Get
Return _last
End Get
Set(value As String)
If value <> _last Then
_last = value

'
OnPropertyChanged を Null パラメータで呼び出します。このプロパティを設定すると、
' "First" の値および
"Name" の値が変更されるためです。

RaisePropertyChanged("")
End If
End Set
End Property
Public Property Hired() As DateTime
Get
Return _hired
End Get
Set(value As DateTime)
If value <> _hired Then
_hired =
value

RaisePropertyChanged("Hired")
End If
End Set
End Property
Public Property Weight() As Double
Get
Return _weight
End Get
Set(value As Double)
If value <> _weight Then
_weight =
value

RaisePropertyChanged("Weight")
End If
End Set
End Property

' 読み取り専用要素
Public ReadOnly Property Father() As String
Get
```

```
Return _father
End Get
End Property
Public ReadOnly Property Brother() As String
Get
Return _brother
End Get
End Property
Public ReadOnly Property Cousin() As String
Get
Return _cousin
End Get
End Property

' ** ユーティリティ
Private Shared Function GetString(arr As String()) As String
Return arr(_rnd.[Next](arr.Length))
End Function
Private Shared Function GetName() As String
Return String.Format("{0} {1}", GetString(_firstNames),
GetString(_lastNames))
End Function

' ** 静的リストプロバイダ
Public Shared Function GetCustomerList(count As Integer) As
        ObservableCollection(Of
Customer)
Dim list = New ObservableCollection(Of Customer)()
For i As Integer = 0 To count - 1
list.Add(New Customer(i))
Next
Return list
End Function

' ** 静的値プロバイダ
Public Shared Function GetCountries() As String()
Return _countries
End Function
Public Shared Function GetFirstNames() As String()
Return _firstNames
End Function
Public Shared Function GetLastNames() As String()
Return _lastNames
End Function
#Region "** INotifyPropertyChanged Members"

' このインタフェースは、連結コントロールがデータオブジェクトの変更に対応できるようにします。

Private Sub RaisePropertyChanged(propertyName As String)
OnPropertyChanged(New
PropertyChangedEventArgs(propertyName))
End Sub
Public Event PropertyChanged As PropertyChangedEventHandler
Protected Sub OnPropertyChanged(e As PropertyChangedEventArgs)
RaiseEvent PropertyChanged(Me, e)
End Sub
```

```
#End Region
#Region "IEditableObject Members"

' このインタフェースは、トランザクション編集を許可します（ユーザーは[Esc]キーを押して直前の値を復元できます）。

Private _clone As Customer
Public Sub BeginEdit()
_clone = DirectCast(Me.MemberwiseClone(), Customer)
End Sub
Public Sub EndEdit()
_clone = Nothing
End Sub
Public Sub CancelEdit()
If _clone IsNot Nothing Then
For Each p In
Me.[GetType]().GetRuntimeProperties()
If p.CanRead
AndAlso p.CanWrite Then

p.SetValue(Me, p.GetValue(_clone, Nothing), Nothing)
End If
Next
End If
End Sub

#End Region
Public Sub BeginEdit1() Implements IEditableObject.BeginEdit

End Sub

Public Sub CancelEdit1() Implements IEditableObject.CancelEdit

End Sub

Public Sub EndEdit1() Implements IEditableObject.EndEdit

End Sub

Public Event PropertyChanged1(sender As Object, e As
        PropertyChangedEventArgs) Implements
INotifyPropertyChanged.PropertyChanged
End Class
End Namespace
```

6. コードビューに切り替えて、次の名前空間を追加します。

C# でコードを書く場合

C#

```
using C1.Xaml.FlexGrid;
using C1.Xaml;
using System.Collections.ObjectModel;
```

Visual Basic でコードを書く場合

```
Visual Basic
Imports C1.Xaml.FlexGrid
Imports C1.Xaml
Imports System.Collections.ObjectModel
```

7. MainPage() コンストラクタの上に次のコードを追加して、ICollectionView インタフェースの実装を記述します。

C# でコードを書く場合

```
C#
private readonly C1CollectionView _c1CollectionView;
private const string NoneItem = "(None)";
```

Visual Basic でコードを書く場合

```
Visual Basic
Dim _c1CollectionView As C1CollectionView
Const NoneItem As String = "(None)"
```

8. InitializeComponent() メソッドのすぐ下に次のコードを挿入して、リストと監視可能なコレクションを作成します。このコードは C1FlexGrid.ItemsSource プロパティも設定します。

C# でコードを書く場合

```
C#
IList<string> fieldNames = new string[]
{ "ID", "Name", "Country", "Active", "Hired", "Father", "Weight" };

List<string> groupFields = new
List<string>(fieldNames);
groupFields.Sort();
//groupFields.Remove("Active");
List<string> filterFields = new
List<string>(groupFields);
groupFields.Insert(0, NoneItem);
groupComboBox.ItemsSource = groupFields;
groupComboBox.SelectedItem = NoneItem;
filterComboBox.ItemsSource = filterFields;
filterComboBox.SelectedIndex = 0;


ObservableCollection<Customer> customers =
Customer.GetCustomerList(50);
_c1CollectionView = new C1CollectionView();
_c1CollectionView.SourceCollection = customers;

c1FlexGrid1.ItemsSource = _c1CollectionView;
}
```

Visual Basic でコードを書く場合

```
Visual Basic
Dim fieldNames As IList(Of String) = New String()
{"ID", "Name", "Country", "Active", "Hired", "Father", "Weight"}

Dim groupFields As New List(Of String)(fieldNames)
```

```
groupFields.Sort()
groupFields.Remove("Active")
Dim filterFields As New List(Of String)(groupFields)
groupFields.Insert(0, NoneItem)
groupComboBox.ItemsSource = groupFields
groupComboBox.SelectedItem = NoneItem
filterComboBox.ItemsSource = filterFields
filterComboBox.SelectedIndex = 0

Dim customers As ObservableCollection(Of Customer) = Customer.GetCustomerList(50)
_c1CollectionView = New C1CollectionView()
_c1CollectionView.SourceCollection = customers

flexgrid1.ItemsSource = _c1CollectionView
```

9. グループ化機能を追加するには、次のコードを追加します。

C# でコードを書く場合

C#

```
void UpdateGrouping()
{
if (_c1CollectionView == null)
return;
using (_c1CollectionView.DeferRefresh())
{

_c1CollectionView.GroupDescriptions.Clear();
if (groupComboBox.SelectedItem !=
NoneItem)
{

_c1CollectionView.GroupDescriptions.Add(new PropertyGroupDescription
        ((string)groupComboBox.SelectedItem));
}
}
}
```

Visual Basic でコードを書く場合

Visual Basic

```
Private Sub UpdateGrouping()
If (_c1CollectionView Is Nothing) Then
Return
End If
_c1CollectionView.DeferRefresh()
_c1CollectionView.GroupDescriptions.Clear()
If (groupComboBox.SelectedItem <> NoneItem) Then
_c1CollectionView.GroupDescriptions.Add(New
PropertyGroupDescription(CType(groupComboBox.SelectedItem, String)))
End If
End Sub
```

10. C1FlexGrid のフィルタ処理機能を設定および制御するには、次のコードを使用します。

C# でコードを書く場合

C#

```
void UpdateFiltering()
if (filterTextBox.Text.Length == 0)
_c1CollectionView.Filter = null;
else
{
if (_c1CollectionView.Filter ==
null)

_c1CollectionView.Filter = FilterFunction;
else

_c1CollectionView.Refresh();
}
}

bool FilterFunction(object customer)
{
Customer cust = customer as Customer;
if (cust == null)
return false;
object propValue = null;
switch ((string)filterComboBox.SelectedItem)
{
case "ID":
propValue =
cust.ID;
break;
case "Name":
propValue =
cust.Name;
break;
case "Country":
propValue =
cust.Country;
break;
case "Hired":
propValue =
cust.Hired;
break;
case "Father":
propValue =
cust.Father;
break;
case "Weight":
propValue =
cust.Weight;
break;
default:
return true;
}
if (propValue == null)
return false;
return propValue.ToString().StartsWith(filterTextBox.Text,
StringComparison.CurrentCultureIgnoreCase);
```

```
}
```

Visual Basic でコードを書く場合

Visual Basic

```
 Private Sub UpdateFiltering()
If filterTextBox.Text.Length = 0 Then
_c1CollectionView.Filter = Nothing
Else
If _c1CollectionView.Filter Is Nothing Then
_c1CollectionView.Filter = AddressOf
FilterFunction
Else
_c1CollectionView.Refresh()
End If
End If
End Sub

Private Function FilterFunction(ByVal customer As Object) As Boolean '
Dim cust As Customer = CType(customer, Customer)
If (cust Is Nothing) Then
Return False
End If

Dim propValue As Object = Nothing
Select Case (CType(filterComboBox.SelectedItem, String))
Case "ID"
propValue = cust.ID
Case "Name"
propValue = cust.Name
Case "Country"
propValue = cust.Country
Case "Hired"
propValue = cust.Hired
Case "Father"
propValue = cust.Father
Case "Weight"
propValue = cust.Weight
Case Else
Return True
End Select
```

11. 最後に、ComboBox と TextBox の SelectionChanged および TextChanged イベントを処理するコードを追加します。

C# でコードを書く場合

C#

```
private void groupComboBox_SelectionChanged_1(object sender,
SelectionChangedEventArgs e)
{
UpdateGrouping();
}

private void filterTextBox_TextChanged_1(object sender, TextChangedEventArgs e)
{
UpdateFiltering();
```

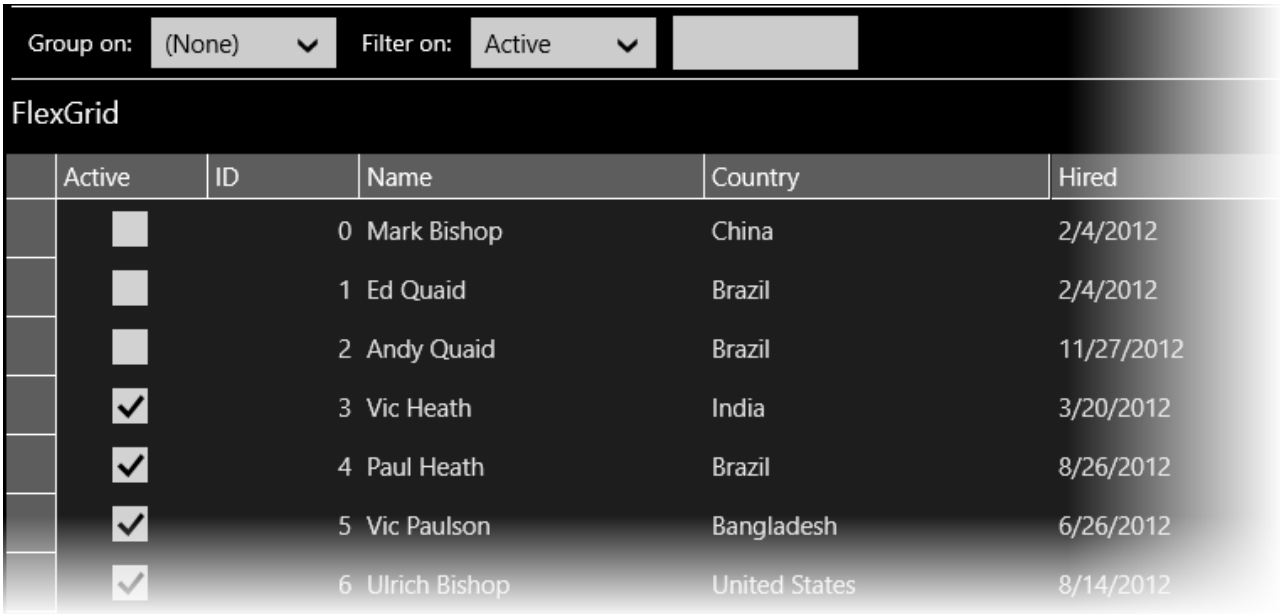
```
}

private void filterComboBox_SelectionChanged_1(object sender,
SelectionChangedEventArgs
e)
{
filterTextBox.Text = "";
}
```

Visual Basic でコードを書く場合

Visual Basic

```
If (propValue Is Nothing) Then
Return False
End If
Return propValue.ToString.StartsWith(filterTextBox.Text,
StringComparison.CurrentCultureIgnoreCase)
End Function
Private Sub groupComboBox_SelectionChanged_1(ByVal sender As Object, ByVal e As
SelectionChangedEventArgs)
UpdateGrouping()
End Sub
Private Sub filterTextBox_TextChanged_1(ByVal sender As Object, ByVal e As
TextChangedEventArgs)
UpdateFiltering()
End Sub
Private Sub filterComboBox_SelectionChanged_1(ByVal sender As Object, ByVal e As
SelectionChangedEventArgs)
filterTextBox.Text = ""
End Sub
```

12. アプリケーションを実行すると、さまざまな方法でデータをフィルタ処理したりグループ化することができます。C1FlexGrid コントロールは次の図のようになります。



タスク別ヘルプ

コードでの C1FlexGrid の作成

C1FlexGrid コントロールを作成するには、アプリケーションのコンストラクタの下に次のコードを挿入します。

Visual Basic

```
Dim flexgrid1 As C1FlexGrid
LayoutRoot.Children.Add(flexgrid1)
```

C#

```
var flexgrid1 = new C1FlexGrid();
LayoutRoot.Children.Add(flexgrid1);
```