FlexGrid for WPF

2022.04.21 更新

グレープシティ株式会社

目次

FlexGrid for WPF の概要

FlexGrid for WPF is a lightweight data grid control designed on a flexible object model. Based on the popular WinForms version, **FlexGrid** offers many unique features such as unbound mode, flexible cell merging, and multi-cell row and column headers that make it a proven solution for data management and tabulation.

Symbol	Name	Bid	Ask	Last Sale
A	Agilent Technologies	765.95 (2.9%)	808.60 (4.5%)	787.27 (3.7%)
AA	Alcoa Inc.	940.02 (-1.4%)	856.95 (3.2%)	898.49 (0.7%)
AACC	Asset Acceptance Capital Corp.	712.93 (-3.0%)	678.27 (1.7%)	695.60 (-0.8%)
AAME	Atlantic American Corporation	842.38 (4.9%)	980.37 (-2.6%)	911.38 (0.7%)
AANB	Abigail Adams National Bancorp, Inc.	784.71 (0.3%)	749.68 (-0.9%)	767.19 (-0.3%)
AAON	AAON, Inc.	187.56 (6.0%)	198.61 (5.1%)	193.08 (5.5%)
AAPL	Apple Inc.	31.14 (-2.0%)	30.88 (-1.6%)	31.01 (-1.8%)
AATI	Advanced Analogic Technologies, Inc.	136.74 (2.5%)	144.40 (2.9%)	140.57 (2.7%)
AAUK	Anglo American plc	168.06 (0.8%)	138.89 (-3.7%)	153.47 (-1.3%)
AAWW	Atlas Air Worldwide Holdings	259.02 (5.6%)	197.86 (3.6%)	228.44 (4.7%)
ABAX	ABAXIS, Inc.	758.44 (5.6%)	687.37 (-2.1%)	722.91 (1.8%)
ABBC	Abington Bancorp, Inc.	215.18 (-0.4%)	180.83 (-3.7%)	198.01 (-1.9%)

リファレンス	
.NET Framework アセンブリ	**.NET アセンブリ**

注意: FlexGridコントロールは、**.NET Framework** および**.NET** の両方と互換性があります。

主な特長

FlexGrid では、シンプルなグリッドのほかに高度なデータ可視化機能を使用できます。以下に、これらの機能を一覧にします。

- **柔軟なデータ連結**
  FlexGrid は、連結モードでも非連結モードでも使用できます。連結モードでは、データソースからデータを取得して表示します。非連結モードでは、グリッド自身がデータを管理します。

- **高度なグリッド機能**
  FlexGrid は、セル結合、データフィルタ処理、ソート、編集、集計などの高度なグリッド機能をサポートします。隣接する同じ値のセルを結合して複数のセルにまたがってデータを表示したり、セルの範囲に対して合計、平均などの統計値を計算したり、グリッドの各列にフィルタを適用することができます。

- **階層化されたスタイル**
  FlexGrid は、ツリー形式の階層化スタイルにデータをまとめます。各レコードを展開または折りたたんで、子グリッドに詳細を表示できます。

- **印刷のサポートの統合**
  FlexGrid には印刷のサポートが組み込まれ、用紙の方向、マージン、フッターテキストなどを制御できます。改ページを処理したり、繰り返しのヘッダー行を追加したり、各ページにカスタム要素を追加するなど、さまざまな印刷イベントを提供するリッチオブジェクトモデルを備えます。ダイアログを表示して、プリンタの選択や設定をユーザーに任せることもできます。

- **高度なグループ化とフィルタ処理機能**
  FlexGrid は、独立したアセンブリとして提供される独立したコントロール FlexGridGroupPanel により、UI 機能としてグ

ループ化をサポートします。同様に、このコントロールには FlexGridFilter コンポーネントが付属します。このコンポーネントはアドホックなフィルタ処理を可能にし、フットプリントを抑えるために別個に提供されます。

- **カスタムセル**
  FlexGrid は、カスタムセルによってグリッドの大幅なカスタマイズをサポートします。コントロールは、ビジュアル要素をカスタマイズするために、CellFactory クラスと組み込みの CellTemplate および CellEditingTemplate を提供します。

- **行の詳細**
  FlexGrid では、データテンプレートで柔軟に行詳細を表示できます。これを使用して、テキストや画像だけでなくデータ連結コントロールを表示できます。

- **固定とピン留め**
  実行時にマウスをドラッグして行と列を固定できます。1つ以上の列をFlexGridの左側にピン留めできます。FlexGrid の列の固定を使用すると、特定の列の順序で列をロックできます。これにより、グリッドを水平方向にスクロールしながら列を表示できます。

WPF グリッドの比較

このセクションでは、各プラットフォームで FlexGrid が提供する機能と、FlexGrid for WPF の他のグリッドコントロールの機能を表にして比較します。

WPF グリッドの比較
WPF エディションで使用できるさまざまなグリッドの比較です。

WPF グリッドの比較

C1FlexGrid、C1DataGrid、MS DataGrid などのさまざまな WPF グリッドから提供されるすべての機能を説明します。PDF 版の比較表をダウンロードできます。

データ連結

機能	C1FlexGrid	C1DataGrid	MS DataGrid
連結モード	○	○	○
非連結モード	○		

レイアウトおよび外観

機能	C1FlexGrid	C1DataGrid	MS DataGrid
テーマ	17 のテーマ	17 のテーマ	
ClearStyle		○	

表現

機能	C1FlexGrid	C1DataGrid	MS DataGrid
列の自動生成	○	○	○
テキスト列	○	○	○
CheckBox 列	○	○	○
ComboBox 列	○	○	○
ハイパーリンク列		○	○
DateTime 列		○	
数値列		○	
Image 列		○	
固定列	○	○	○
固定行	○	○	
カスタム列	○	○	○
カスタム行		○	
カスタムセル（セルファクトリ）	○		
新規行の追加	○	○	○
セルの結合	○	○	

行詳細	○	○	○
階層ビュー	○	カスタムコードの使用	

ソート

機能	C1FlexGrid	C1DataGrid	MS DataGrid
ICollectionView	○	○	○

フィルタ処理

機能	C1FlexGrid	C1DataGrid	MS DataGrid
ICollectionView	○	○	○
Excel 形式のフィルタ機能	○	○	
フィルタ行	カスタムコードの使用	○	
カスタムフィルタ		○	
全文検索		○	

グループ化

機能	C1FlexGrid	C1DataGrid	MS DataGrid
ICollectionView	○	○	○
ドラッグアンドドロップによるグループ化	○	○	
小計	○	○	

編集

機能	C1FlexGrid	C1DataGrid	MS DataGrid
セル内編集	○	○	○
検証	○	○	○
IDataErrorInfo	○	○	○
IEditableObject		○	○
ICustomTypeDescriptor	○	○	○
データ注釈	○	○	

印刷

機能	C1FlexGrid	C1DataGrid	MS DataGrid
印刷	○	○	

エクスポート

機能	C1FlexGrid	C1DataGrid	MS DataGrid

Excel	○	○	
Text	○		
HTML	○		

Ux

機能	C1FlexGrid	C1DataGrid	MS DataGrid
キーボードナビゲーション	○	○	○
RTL サポート	○	○	○
タッチサポート	○	○	○
クリップボードサポート	○	○	○
複数選択モード	○	○	○

ローカライズ

機能	C1FlexGrid	C1DataGrid	MS DataGrid
.NET ローカライズのサポート	○	○	○
内蔵の翻訳	25 の言語	25 の言語	○
地域設定（数、日付、通貨）	○	○	○

パフォーマンス

機能	C1FlexGrid	C1DataGrid	MS DataGrid
遅延スクロール	○	○	○
UI の仮想化	25 の言語	25 の言語	○
C1DataSource によるサーバー側データ仮想化	○	○	

その他

機能	C1FlexGrid	C1DataGrid	MS DataGrid
設計時サポート	○	○	○
WPF 互換性	○	○	○
UI オートメーション	○	○	○
アセンブリサイズ	301 KB	776 KB	PresentationFramework.dll の一部

FlexGrid API の比較

WPF エディションの WinForms の C1FlexGrid クラスのオブジェクトモデルついて説明します。

要素

プロパティ

FlexGrid for WinForms	FlexGrid for WPF and Silverlight	コメント
BottomRow	BottomRow	
Cols	Columns	
ExtendLastCol		最後の列の幅を "*" に設定します。
LeftCol	LeftColumn	
Rows		
RightCol		
TopRow		

メソッド

FlexGrid for WinForms	FlexGrid for WPF and Silverlight	コメント
FindRow(...)		サポートされていません（コードで簡単に実行可能）

イベント

FlexGrid for WinForms	FlexGrid for WPF and Silverlight	コメント
EnterCell	SelectionChanged	
LeaveCell	SelectionChanging	
RowValidated	OnRowEditEnded	
RowValidating	OnRowEditEnding	

コア機能

プロパティ

FlexGrid for WinForms	FlexGrid for WPF and Silverlight	コメント
AllowAddNew		データソースレベルでサポート。組み込み UI なし。
AllowDelete		サポートされていません。
AllowEditing	IsReadOnly	WPF コントロールで一貫性を持たせるために名前が変更されています。
AllowFiltering		データソースレベルでサポート。組み込み UI なし。

AllowFreezing		Rows.Frozen プロパティと Columns.Frozen プロパティを参照してください。
AllowMerging	AllowMerging	
AllowSorting	AllowSorting	
AutoSearch		サポートされていません。
AutoSearchDelay		サポートされていません。
AutoGenerateColumns	AutoGenerateColumns	
DataSource	ItemsSource	
Enabled	IsEnabled	
EditOptions		CellFactory プロパティと PrepareCellForEdit イベントを参照してください。
SelectionMode	SelectionMode	
Subtotal	Column.GroupAggregate	
SubtotalPosition		サポートされていません。

メソッド		
FlexGrid for WinForms	**FlexGrid for WPF and Silverlight**	**コメント**
BeginUpdate		
EndUpdate		
FinishEditing (bool cancel)		
LoadExcel (string)		
SaveExcel (string)		
Select(int row, int col, bool scrollIntoView)		
Sort (order, int col1, int col2)		
StartEditing (row, col)		

イベント		
FlexGrid for WinForms	**FlexGrid for WPF and Silverlight**	**コメント**
AfterSort		
AfterSubtotal		サポートされていません。

BeforeSubtotal		サポートされていません。
StartEdit	BeginningEdit	
SetUpEditor	PrepareCellForEdit	

レイアウトおよび外観

プロパティ		
FlexGrid for WinForms	**FlexGrid for WPF and Silverlight**	**コメント**
BackColor	Background	
ClipboardCopyMode	ClipboardCopyMode	
DrawMode		CellFactory プロパティを参照してください。
HighLight		サポートされていません。
NewRowWatermark		サポートされていません。
ShowButtons		サポートされていません。
ShowCellLabels		サポートされていません。
ShowCursor		サポートされていません。
ShowErrors		サポートされていません。
ShowSort	ShowSort	
FocusRect		サポートされていません。

ユーザー操作

プロパティ		
FlexGrid for WinForms	**FlexGrid for WPF and Silverlight**	**コメント**
AllowDragging	AllowDragging	
AllowResizing	AllowResizing	
AutoResize		AutoSizeRows メソッドと AutoSizeColumns メソッドを参照してください。
ScrollBars	HorizontalScrollbarVisibility VerticalScrollbarVisibility	サポートされていません。

ScrollOptions		サポートされていません。
ScrollPosition	ScrollPosition	

イベント		
FlexGrid for WinForms	**FlexGrid for WPF and Silverlight**	**コメント**
AfterScroll	ScrollPositionChanged	
BeforeScroll	ScrollPositionChanging	
AfterSelChange	SelectionChanged	
BeforeSelChange	SelectionChanging	
SelChange	SelectionChanged	

キーボード操作

プロパティ		
FlexGrid for WinForms	**FlexGrid for WPF and Silverlight**	**コメント**
AutoClipboard		ClipboardCopyMode と ClipboardPasteMode を参照してください
ClipboardCopyMode	ClipboardCopyMode	
KeyActionEnter	KeyActionEnter	
KeyActionTab	KeyActionTab	
ScrollOptions		サポートされていません。
ScrollPosition	ScrollPosition	

階層化機能

プロパティ		
FlexGrid for WinForms	**FlexGrid for WPF and Silverlight**	**コメント**
TreeIndent	Tree.Indent	
Tree.Show(level)	CollapseGroupsToLevel(level)	

クイックスタート

このクイックスタートでは、FlexGrid コントロールを追加し、データを生成します。最初に Visual Studio で WPF アプリケーションを作成し、FlexGrid コントロールを追加し、データを連結します。

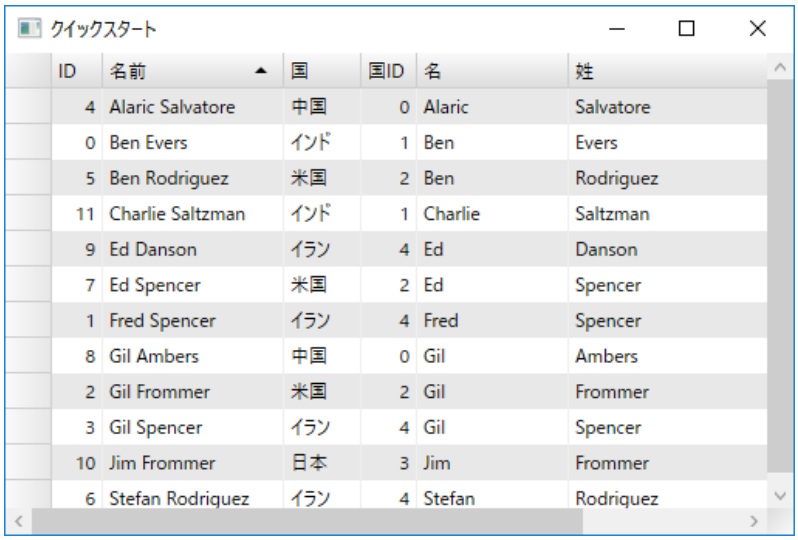
以下のセクションでは、**.NET 4.5.2**および**.NET 5**バージョンの**FlexGrid**の使用を開始します。

.NET Framework

FlexGrid コントロールにデータを追加して表示する単純な WPF アプリケーションを作成するには、次の手順に従います。

1. **FlexGrid コントロールを WPF アプリケーションに追加する**
2. **データを追加して FlexGrid に表示する**
3. **FlexGrid をデータと連結する**

次の図に、顧客のサンプルデータが挿入された FlexGrid を示します。



FlexGrid コントロールを WPF アプリケーションに追加する

XAML とコードから WPF アプリケーションに FlexGrid コントロールを追加します。

XAML の使用

1. Visual Studio で WPF プロジェクトを作成します。
2. FlexGrid コントロールを XAML デザイナ MainWindow.xaml にドラッグします。
   C1.WPF.FlexGrid.dll がプロジェクトの参照フォルダに追加されます。
3. 次のコード例に示すように、XAML コードを編集してコントロールの名前を「grid」に設定します。

```
<Window x:Class="FilterRow.MainWindow"
        ...
        xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
        ... >
    <Grid x:Name="LayoutRoot">
        <c1:C1FlexGrid Name='grid' Grid.Row="1"/>
    </Grid>
</Window>
```

コードの使用

1. Visual Studio で WPF プロジェクトを作成します。
2. C1.WPF.FlexGrid .dll ファイルをプロジェクトの **References** フォルダに追加します。
3. コードビュー（MainWindow.xaml.cs ファイル）に切り替え、次の using 文を追加します。

```
using C1.WPF.FlexGrid;
```

4. MainWindow クラスコンストラクタに次のコードを追加して、FlexGrid コントロールを追加します。

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

先頭に戻る

データを追加して FlexGrid に表示する

1. MainWindow.xaml.cs のコードビューに切り替えます。
2. クラス **Customer** を作成し、次のコードを使用して、FlexGrid に表示されるデータを追加します。

```
public class Customer
{
    //フィールド
    int _id, _countryID;
    string _first, _last;
    double _weight;

    //データの生成
    static Random _rnd = new Random();
    static string[] _firstNames =
"Andy|Ben|Charlie|Dan|Ed|Fred|Gil|Herb|Jim|Elena|Stefan|Alaric|Gina".Split('|');
    static string[] _lastNames =
"Ambers|Bishop|Cole|Danson|Evers|Frommer|Salvatore|Spencer|Saltzman|Rodriguez".Split('|');
    static string[] _countries = "中国|インド|米国|日本|イラン".Split('|');

    public Customer()
        : this(_rnd.Next())
    {
    }
    public Customer(int id)
    {
        ID = id;

        名 = GetString(_firstNames);
        姓 = GetString(_lastNames);
        国ID = _rnd.Next() % _countries.Length;
        重量 = 50 + _rnd.NextDouble() * 50;
    }
    //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 名前
    {
        get { return string.Format("{0} {1}", 名, 姓); }
    }
    public string 国
    {
        get { return _countries[_countryID]; }
    }
    public int 国ID
    {
        get { return _countryID; }
        set
        {
            if (value != _countryID && value > -1 && value < _countries.Length)
            {
                _countryID = value;
                RaisePropertyChanged(null);
            }
        }
    }
    public string 名
    {
        get { return _first; }
        set
        {
            if (value != _first)
            {
                _first = value;
                RaisePropertyChanged(null);
            }
        }
    }
    public string 姓
    {
        get { return _last; }
        set
        {
            if (value != _last)
            {
                _last = value;
```

```
                RaisePropertyChanged(null);
            }
        }
    }
    public double 重量
    {
        get { return _weight; }
        set
        {
            if (value != _weight)
            {
                _weight = value;
                RaisePropertyChanged("Weight");
            }
        }
    }
    // **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;
        }

   // このインタフェースにより、バウンドコントロールがデータオブジェクトの変更に反応することができます。
    void RaisePropertyChanged(string propertyName)
    {
        OnPropertyChanged(new PropertyChangedEventArgs(propertyName));
    }
    public event PropertyChangedEventHandler PropertyChanged;
    protected void OnPropertyChanged(PropertyChangedEventArgs e)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, e);
    }
}
```

FlexGrid をデータと連結する

1. **ItemsSource** プロパティを設定して、Customer クラスからランダムなデータをグリッドに入力します。

```
//グリッドにデータを連結します
grid.ItemsSource = Customer.GetCustomerList(12);
```

先頭に戻る

.NET

FlexGrid コントロールにデータを追加して表示する単純な WPF コアアプリケーションを作成するには、次の手順に従います。

1. **FlexGrid コントロールを WPF アプリケーションに追加する**
2. **データを追加して FlexGrid に表示する**
3. **FlexGrid をデータと連結する**

次の図に、顧客のサンプルデータが挿入された FlexGrid を示します。

ID	Active	Name	Country	Country ID	First Name	Last Name
0	☐	Andy Saltzman	Japan	3	Andy	Saltzman
1	☐	Fred Cole	Myanmar	4	Fred	Cole
2	☐	Stefan Rodriguez	Japan	3	Stefan	Rodriguez
3	☐	Fred Danson	United States	2	Fred	Danson
4	☐	Elena Saltzman	Japan	3	Elena	Saltzman
5	☐	Jim Rodriguez	India	1	Jim	Rodriguez
6	☐	Herb Rodriguez	Japan	3	Herb	Rodriguez
7	☐	Herb Ambers	China	0	Herb	Ambers
8	☐	Gina Bishop	China	0	Gina	Bishop
9	☐	Gil Evers	United States	2	Gil	Evers
10	☐	Herb Rodriguez	India	1	Herb	Rodriguez

FlexGrid コントロールを WPF アプリケーションに追加する

XAML とコードから WPF コアアプリケーションに FlexGrid コントロールを追加します。

XAML の使用

1. 5.0フレームワークを使用してVisualStudioでWPFコアプロジェクトを作成します。
2. FlexGridコントロールをXAMLデザイナ（MainWindow.xaml）にドラッグします。
   C1.WPF.Grid.dllは、プロジェクトの参照フォルダーに追加されます。
3. 次のコード例に示すようにXAMLコードを編集して、コントロールの名前を「grid_design」に設定します。

```
<Window
                                                                                ...
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                                                                                ...  >
<Grid x:Name="LayoutRoot">
<c1:FlexGrid Name='grid_design'>
</c1:FlexGrid>
</Grid>
</Window>
```

先頭に戻る

コードの使用

1. Visual Studio で WPF プロジェクトを作成します。
2. FlexGrid コントロールを XAML デザイナ MainWindow.xaml にドラッグします。
3. コードビュー（MainWindow.xaml.cs ファイル）に切り替え、次の using 文を追加します。

```
using C1.WPF.Grid;
```

4. MainWindow クラスコンストラクタに次のコードを追加して、FlexGrid コントロールを追加します。

```
//コードから新しいFlexGridコントロールを初期化します
var grid_code = new FlexGrid();
LayoutRoot.Children.Add(grid_code);
```

データを追加して FlexGrid に表示する

1. MainWindow.xaml.cs のコードビューに切り替えます。
2. クラス Customer を作成し、次のコードを使用して、FlexGrid に表示されるデータを追加します。

C#

```
public class Customer
{
    //フィールド
    int _id, _countryID;
    string _first, _last;
    double _weight;
    bool _active;

    //データの生成
    static Random _rnd = new Random();
    static string[] _firstNames =
"Andy|Ben|Charlie|Dan|Ed|Fred|Gil|Herb|Jim|Elena|Stefan|Alaric|Gina".Split('|');
    static string[] _lastNames =
"Ambers|Bishop|Cole|Danson|Evers|Frommer|Salvatore|Spencer|Saltzman|Rodriguez".Split('|');
    static string[] _countries = "China|India|United States|Japan|Myanmar".Split('|');

    public Customer()
        : this(_rnd.Next())
    {
    }
    public Customer(int id)
    {
        ID = id;
        Active = false;
        FirstName = GetString(_firstNames);
        LastName = GetString(_lastNames);
        CountryID = _rnd.Next() % _countries.Length;
        Weight = 50 + _rnd.NextDouble() * 50;
    }
    //オブジェクトモデル
    public int ID
    {
        get { return _id; }
        set
        {
            if (value != _id)
            {
                _id = value;
                RaisePropertyChanged("ID");
            }
        }
    }
    public bool Active
    {
        get { return _active; }
        set
        {
            _active = value;
            RaisePropertyChanged("Active");
        }
    }
    public string Name
    {
        get { return string.Format("{0} {1}", FirstName, LastName); }
    }
    public string Country
    {
        get { return _countries[_countryID]; }
    }
    public int CountryID
    {
        get { return _countryID; }
```

```
        set
        {
            if (value != _countryID && value > -1 && value < _countries.Length)
            {
                _countryID = value;
                RaisePropertyChanged(null);
            }
        }
    }
    public string FirstName
    {
        get { return _first; }
        set
        {
            if (value != _first)
            {
                _first = value;
                RaisePropertyChanged(null);
            }
        }
    }
    public string LastName
    {
        get { return _last; }
        set
        {
            if (value != _last)
            {
                _last = value;
                RaisePropertyChanged(null);
            }
        }
    }
    public double Weight
    {
        get { return _weight; }
        set
        {
            if (value != _weight)
            {
                _weight = value;
                RaisePropertyChanged("Weight");
            }
        }
    }
    // **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;
        }

    // このインタフェースにより、バウンドコントロールがデータオブジェクトの変更に反応することができます。
```

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

FlexGrid をデータと連結する

1. **ItemsSource** プロパティを設定して、**Customer** クラスからランダムなデータをグリッドに入力します。

C#

```
//グリッドデータを生成します
var gridData = Customer.GetCustomerList(12);
//グリッドにデータを連結します
grid_code.ItemsSource = gridData;
```

先頭に戻る

オブジェクトモデルの概要

FlexGrid には、さまざまなクラス、オブジェクト、コレクション、関連するメソッドおよびプロパティを提供するリッチオブジェクトモデルが用意されています。これらのオブジェクトの一部とその主なプロパティを次の表に一覧します。

C1FlexGrid
プロパティ: AllowAddNew, **AllowDragging**, **AllowMerging**, **AllowResizing**, **AllowSorting**, **AutoComplete**, **AutoGenerateColumns**, **CellFactory**, **Cells**, **Columns**, **FrozenRows**, **ItemsSource**, **Rows**, **SelectedIndex**, **SelectionMode**, **ShowErrors**, **ShowMarquee, TreeIndent** **メソッド: AutoSizeColumn**, **AutoSizeColumns**, **AutoSizeRow**, **AutoSizeRows**, **GetAggregate**, **Copy**, **HitTest**
Column
プロパティ: AllowSorting, **AutoGenerated**, **Binding**, **CellEditingTemplate**, **CellTemplate**, **Format**, **GroupAggregate**, **Header**
CellStyle
プロパティ: Background, **CornerRadius**, **FontSize**, **HorizontalAlignment**, **Tag**, **TextWrapping**, **VerticalAlignment**
C1FlexGridFilter
プロパティ: Editor, **FilterDefinition**, **NullValueString**, **Owner**, **UseCollectionView** **メソッド: GetColumnFilter**, **LoadFilterDefinition**, **SaveFilterDefinition**, **ShowFilterEditor**
C1FlexGridGroupPanel
プロパティ: DragMarkerColor, **FlexGrid**, **HideGroupedColumns**, **MaxGroups**, **Watermark**, **WatermarkText** **メソッド: OnApplyTemplate**
Row
プロパティ: ActualHeight, **Bottom**, **DataItem**, **Grid**, **GridPanel**, **Index**, **Selected**, **Top** **メソッド: GetDataFormatted**, **GetDataRaw**, **GetErrors**
RowCol
プロパティ: AllowDragging, **AllowMerging**, **AllowResizing**, **CellStyle**, **CellStyle**, **GridPanel**, **Foreground**, **HeaderTextWrapping**

XAML クイックレファレンス

このトピックでは、**C1FlexGrid** コントロールの作成に使用される XAML の概要を提供します。

開発を開始するには、ルート要素タグに **c1** 名前空間宣言を追加します。

xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"

次に、**FilterRow** サンプルから抜粋した **C1FlexGrid** の例を示します。

このサンプルの XAML は次のようになります。

XAML

```
<Window x:Class="FilterRow.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
Title="C1FlexGrid: FilterRow" Height="350" Width="700"
WindowStartupLocation="CenterScreen" >
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition/>
</Grid.RowDefinitions>
<CheckBox Content="'%' をワイルドカードとして使用" Margin="6" Click="CheckBox_Click" />
<c1:C1FlexGrid Name="_flex" KeyActionTab="MoveAcross" Grid.Row="1"/>
</Grid>
</Window>
```

C1FlexGrid の追加

次のように、FlexGrid コントロールは、WPF アプリケーションに XAML デザイナから追加することも、コードから追加することもできます。

XAML デザイナから FlexGrid を追加するには

1. Visual Studio で WPF プロジェクトを作成します。
2. C1.WPF.FlexGrid .dll ファイルをプロジェクトの **References** フォルダに追加します。
3. XAML マークアップに次のコードを追加して、FlexGrid コントロールを作成します。

WPF

```
<Window x:Class="FilterRow.MainWindow"
        ...
        xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
        ... >
    <Grid x:Name="LayoutRoot">
        <c1:C1FlexGrid/>
    </Grid>
</Window>
```

コードから FlexGrid を追加するには

1. Visual Studio で WPF プロジェクトを作成します。
2. C1.WPF.FlexGrid.dll ファイルをプロジェクトの **References** フォルダに追加します。
3. コードビュー（MainWindow.xaml.cs）に切り替え、次の参照文を追加します。

WPF

```
using C1.WPF.FlexGrid;
```

4. MainWindow クラスコンストラクタに次のコードを追加して、FlexGrid コントロールを追加します。

WPF

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

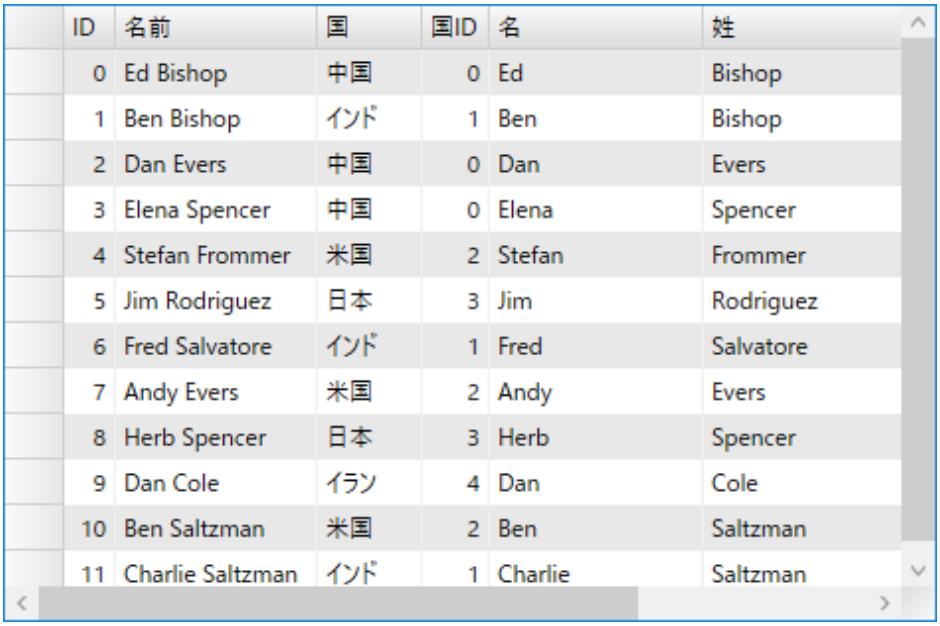
グリッドへの挿入

他のデータ視覚化コントロールと同様に、FlexGrid は連結モードサポートし、さまざまなデータソースのデータを挿入できます。
以下のセクションでは、.NET 4.5.2および.NET 5 バージョンのバインドされたFlexGridについて説明します。

.NET Framework

C1FlexGrid クラスの **ItemsSource** プロパティを使用して、グリッドにデータを挿入できます。WPF では、ItemsSource プロパティは FlexGrid を **IEnumerable** インタフェースまたは ICollectionView インタフェースに連結します。

次の図は、連結 FlexGrid を示しています。



	ID	名前	国	国ID	名	姓
	0	Ed Bishop	中国	0	Ed	Bishop
	1	Ben Bishop	インド	1	Ben	Bishop
	2	Dan Evers	中国	0	Dan	Evers
	3	Elena Spencer	中国	0	Elena	Spencer
	4	Stefan Frommer	米国	2	Stefan	Frommer
	5	Jim Rodriguez	日本	3	Jim	Rodriguez
	6	Fred Salvatore	インド	1	Fred	Salvatore
	7	Andy Evers	米国	2	Andy	Evers
	8	Herb Spencer	日本	3	Herb	Spencer
	9	Dan Cole	イラン	4	Dan	Cole
	10	Ben Saltzman	米国	2	Ben	Saltzman
	11	Charlie Saltzman	インド	1	Charlie	Saltzman

次のコード例は、FlexGrid コントロールにカスタマーオブジェクトのリストを連結する方法を示します。以下のコードを実行すると、グリッドによってデータソースがスキャンされ、データソース内の項目の各パブリックプロパティに対応する列が自動的に生成されます。自動的に作成されたコードをコードでカスタマイズすることも、列の自動生成を完全に無効にし、コードまたは XAML からカスタム列を作成することもできます。

```
grid.ItemsSource = Customer.GetCustomerList(12);
```

FlexGrid は、リスト(顧客リスト)に直接連結できます。ただし、通常は **DataCollection** に連結することをお勧めします。それは、ICollectionView がアプリケーションに代わって多くのデータ構成を保持し、これをコントロール間で共有できるためです。多くのコントロールを同じ **DataCollection** オブジェクトに連結した場合、それらはすべて同じビューを表示します。あるコントロールで 1 つの項目を選択すると、他の連結コントロールでも同じ項目が自動的に選択されます。フィルタ処理、グループ化、ソート処理も、同じビューに連結されたすべてのコントロールで共有されます。

たとえば、次のコードでは、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

```
<!-- C1FlexGridに列を作成します -->
<grid:C1FlexGrid x:Name="_flexiTunes"
     AutoGenerateColumns="False" >
   <grid:C1FlexGrid.Columns>
     <grid:Column Binding="{Binding Name}" Header="Title"
         AllowDragging="False" Width="300"/>
     <grid:Column Binding="{Binding Duration}"
         HorizontalAlignment="Right" />
     <grid:Column Binding="{Binding Size}"
         HorizontalAlignment="Right" />
     <grid:Column Binding="{Binding Rating}" Width="200"
         HorizontalAlignment ="Center" />
   </grid:C1FlexGrid.Columns>
```

```
</grid:C1FlexGrid>
```

連結をグリッドの **Columns** コレクションのインデクサとして使用できます。たとえば、「Rating」列の幅を 300 ピクセルに設定する場合は、次のコードを使用します。

C#

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

.NET

FlexGrid クラスの ItemsSource プロパティを使用して、グリッドにデータを挿入できます。WPF では、ItemsSource プロパティは FlexGrid を IEnumerable インタフェースまたは ICollectionView インタフェースに連結します。

次の図は、連結 FlexGrid を示しています。

	Title	Active	Name	Country
	0	☐	Ben Rodriguez	Myanmar
	1	☐	Alaric Spencer	China
	2	☐	Elena Saltzman	China
	3	☐	Elena Spencer	Myanmar
	4	☐	Ed Spencer	India
	5	☐	Ben Danson	Myanmar
	6	☐	Herb Evers	India
	7	☐	Gina Bishop	Japan
	8	☐	Jim Spencer	Japan
	9	☐	Charlie Saltzman	India

次のコード例は、FlexGrid コントロールにカスタマーオブジェクトのリストを連結する方法を示します。以下のコードを実行すると、グリッドによってデータソースがスキャンされ、データソース内の項目の各パブリックプロパティに対応する列が自動的に生成されます。自動的に作成されたコードをコードでカスタマイズすることも、列の自動生成を完全に無効にし、コードまたは XAML からカスタム列を作成することもできます。

```
//FlexGridにデータソースを設定
grid.ItemsSource = Customer.GetCustomerList(10);
```

FlexGrid は、リスト(顧客リスト)に直接連結できます。ただし、通常は **DataCollection** に連結することをお勧めします。それは、ICollectionView がアプリケーションに代わって多くのデータ構成を保持し、これをコントロール間で共有できるためです。多くのコントロールを同じ **DataCollection** オブジェクトに連結した場合、それらはすべて同じビューを表示します。あるコントロールで 1 つの項目を選択すると、他の連結コントロールでも同じ項目が自動的に選択されます。フィルタ処理、グループ化、ソート処理も、同じビューに連結されたすべてのコントロールで共有されます。

たとえば、次のコードでは、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

```
<!-- C1FlexGridに列を作成します -->
<c1:FlexGrid x:Name="grid" AutoGenerateColumns="False" Style="{StaticResource
excelBlue}" MinColumnWidth="10" MaxColumnWidth="300"
HorizontalAlignment="Center" HeadersVisibility="All" Height="700" Width="1000">
  <c1:FlexGrid.Columns>
     <c1:GridColumn Binding="Id" Header="Title" AllowDragging="False" Width="300"/>
```

```
    <c1:GridColumn Binding="Active" HorizontalAlignment="Right" />
    <c1:GridColumn Binding="Name" HorizontalAlignment="Right" />
    <c1:GridColumn Binding="Country" Width="200" HorizontalAlignment ="Center" />
  </c1:FlexGrid.Columns>
</c1:FlexGrid>
```

連結をグリッドの **Columns** コレクションのインデクサとして使用できます。たとえば、「Rating」列の幅を 300 ピクセルに設定する場合は、次のコードを使用します。

C#

```
//インデックスとして列の連結を使用します
grid.Columns["Country"].Width = new GridLength(300);
```

アンバウンドモード

他のデータ視覚化コントロールと同様に、FlexGrid は連結モードサポートし、さまざまなデータソースのデータを挿入できます。
以下のセクションでは、.NET 4.5.2および.NET 5 バージョンのアンバウンドのFlexGridについて説明します。

.NET Framework

FlexGrid コントロールは、ObservableCollection や DataCollection などのさまざまなデータソースやコレクションと連携して機能を最大限に活用できるように設計されています。ただし、データソースには制限されず、非連結モードでも使用できます。

以下の画像は、セルインデックス表記によって生成される非連結グリッドを示します。

	[0,0]	[0,1]	[0,2]	[0,3]	[0,4]
	[1,0]	[1,1]	[1,2]	[1,3]	[1,4]
	[2,0]	[2,1]	[2,2]	[2,3]	[2,4]
	[3,0]	[3,1]	[3,2]	[3,3]	[3,4]
	[4,0]	[4,1]	[4,2]	[4,3]	[4,4]
	[5,0]	[5,1]	[5,2]	[5,3]	[5,4]
	[6,0]	[6,1]	[6,2]	[6,3]	[6,4]
	[7,0]	[7,1]	[7,2]	[7,3]	[7,4]
	[8,0]	[8,1]	[8,2]	[8,3]	[8,4]
	[9,0]	[9,1]	[9,2]	[9,3]	[9,4]
	[10,0]	[10,1]	[10,2]	[10,3]	[10,4]

非連結グリッドを作成するには、Add メソッドを使用して、グリッドに行と列を追加します。次のコードでは、グリッドにいくつかの行と列を追加し、対応する行と列に基づいてセルを指定するインデックス表記を使用してグリッドにデータを挿入します。

C#

```
for (int i = 0; i < 20; i++)
    {
       grid.Columns.Add(new Column());
    }
for (int i = 0; i < 500; i++)
    {
       grid.Rows.Add(new Row());
     }

 // 非連結グリッドにインデックスを入力します
for (int r = 0; r < grid.Rows.Count; r++)
    {
         for (int c = 0; c < grid.Columns.Count; c++)
            {
                grid[r, c] = string.Format("[{0},{1}]", r, c);
             }
     }
```

グリッドのインデックス表記は、行と列のインデックスでセルを指定します。行インデックスと列名、または行名と列インデックスに基づいてセルを指定することもできます。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。

グリッドに表示される新しいインデックス表記には、0 行目の項目が含まれません。この表記では、インデックスがデータ項目のインデックスと一致し、列のカウントが表示されるプロパティの数と一致するため、インデックス化が容易です。この表記の唯一の欠点は、固定されたセルのコンテンツにアクセスするために新しいメソッドが必要なことです。新しいメソッドは、**RowHeaders** および **ColumnHeaders**という追加プロパティで構成されます。これらは GridPanel型のオブジェクトを返します。これは、独自の行/列セットを持つサブグリッドと考えることができます。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します。

C#

```
// グリッドの行ヘッダーを取得します
GridPanel rowHeader = grid.RowHeaders;

// グリッドに新しい固定列を追加します
rowHeader.Columns.Add(new Column());

// 行ヘッダーの幅およびコンテンツを設定します
for (int c = 0; c < rowHeader.Columns.Count; c++)
{
    // 行ヘッダーの幅を設定します
    rowHeader.Columns[c].Width = new GridLength(80);
    for (int r = 0; r < rowHeader.Rows.Count; r++)
    {
        // セルのコンテンツを設定します
        rowHeader[r, c] = string.Format("hdr {0},{1}", r, c);
    }
}
```

GridPanelクラスは、メインのグリッドと同様に **Rows** および **Columns** コレクションを公開し、同じインデックス表記をサポートします。行ヘッダーと列ヘッダーはカスタマイズでき、グリッドのコンテンツ領域（スクロール可能な部分）で使用したものと同じオブジェクトモデルや技術を使用してデータを挿入できます。

.NET

FlexGrid コントロールは、ObservableCollection や DataCollection などのさまざまなデータソースやコレクションと連携して機能を最大限に活用できるように設計されています。ただし、データソースには制限されず、非連結モードでも使用できます。

以下の画像は、セルインデックス表記によって生成される非連結グリッドを示します。

	[0,0]	[0,1]	[0,2]	[0,3]	[0,4]
	[1,0]	[1,1]	[1,2]	[1,3]	[1,4]
	[2,0]	[2,1]	[2,2]	[2,3]	[2,4]
	[3,0]	[3,1]	[3,2]	[3,3]	[3,4]
	[4,0]	[4,1]	[4,2]	[4,3]	[4,4]
	[5,0]	[5,1]	[5,2]	[5,3]	[5,4]
	[6,0]	[6,1]	[6,2]	[6,3]	[6,4]
	[7,0]	[7,1]	[7,2]	[7,3]	[7,4]
	[8,0]	[8,1]	[8,2]	[8,3]	[8,4]
	[9,0]	[9,1]	[9,2]	[9,3]	[9,4]
	[10,0]	[10,1]	[10,2]	[10,3]	[10,4]

非連結グリッドを作成するには、Add メソッドを使用して、グリッドに行と列を追加します。次のコードでは、グリッドにいくつかの行と列を追加し、対応する行と列に基づいてセルを指定するインデックス表記を使用してグリッドにデータを挿入します。

C#

```
for (int i = 0; i < 20; i++)
{
    grid.Columns.Add(new GridColumn());
}
for (int i = 0; i < 500; i++)
{
    grid.Rows.Add(new GridRow());
}

// 非連結グリッドにインデックスを入力します
for (int r = 0; r < grid.Rows.Count; r++)
{
    for (int c = 0; c < grid.Columns.Count; c++)
    {
        grid[r, c] = string.Format("[{0},{1}]", r, c);
    }
```

グリッドのインデックス表記は、行と列のインデックスでセルを指定します。行インデックスと列名、または行名と列インデックスに基づいてセルを指定することもできます。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。

グリッドに表示される新しいインデックス表記には、0 行目の項目が含まれません。この表記では、インデックスがデータ項目のインデックスと一致し、列のカウントが表示されるプロパティの数と一致するため、インデックス化が容易です。この表記の唯一の欠点は、固定されたセルのコンテンツにアクセスするために新しいメソッドが必要なことです。新しいメソッドは、**RowHeaders**および **ColumnHeaders**という追加プロパティで構成されます。これらは GridPanel型のオブジェクトを返

します。これは、独自の行/列セットを持つサブグリッドと考えることができます。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します。

C#

```
// グリッドの行ヘッダーを取得します
GridPanel rowHeader = grid.RowHeaders;

// グリッドに新しい固定列を追加します
rowHeader.Columns.Add(new GridColumn());

// 行ヘッダーの幅およびコンテンツを設定します
for (int c = 0; c < rowHeader.Columns.Count; c++)
{
    for (int r = 0; r < rowHeader.Rows.Count; r++)
    {
        // セルのコンテンツを設定します
        rowHeader[r, c] = string.Format("hdr {0},{1}", r, c);
    }
```

GridPanelクラスは、メインのグリッドと同様に **Rows** および **Columns** コレクションを公開し、同じインデックス表記をサポートします。行ヘッダーと列ヘッダーはカスタマイズでき、グリッドのコンテンツ領域（スクロール可能な部分）で使用したものと同じオブジェクトモデルや技術を使用してデータを挿入できます。

基本操作

C1FlexGrid コントロールは、連結グリッドや非連結グリッドを作成したり、セルの結合、列の集計、カスタムセルなど、ビジネスデータに必要なあらゆる拡張機能を提供しています。また、FlexGrid for WPF は Windows Forms 版と類似したオブジェクトモデルで構成されています。行と列のインデックスを作成する方法も類似しています。

このセクションでは、**C1FlexGrid** コントロールの基本操作について説明します。

列

列のデータ型を設定する

列単位でデータ型を設定するには、**DataType** プロパティを使用します。デフォルトでは、列の **DataType** プロパティは **System.Object** に設定されていて、列に任意のデータ値を格納できます。

次の例では、いくつかの列に対してデータ型を設定してその結果を表示します。

【実行例】

マークアップ

```
<c1:C1FlexGrid.Columns>
    <c1:Column Header="列1"/>
    <!--Data型を設定します-->
    <c1:Column Header="DateTime型" DataType="{x:Type DatePicker}"/>
    <c1:Column Header="Boolean型" DataType="{x:Type sys:Boolean}"/>
    <c1:Column Header="Decimal型" />
    <c1:Column Header="列5"/>
</c1:C1FlexGrid.Columns>
```

必要に応じて、**DataType** プロパティをコードで設定することもできます。

Visual Basic

```
Public Sub New()
    InitializeComponent()
    '列のDataTypeプロパティを設定します
    c1FlexGrid1.Columns(1).DataType = GetType(DateTime)
    c1FlexGrid1.Columns(2).DataType = GetType([Boolean])
    c1FlexGrid1.Columns(3).DataType = GetType([Decimal])
End Sub
```

C#

```
//列のDataTypeプロパティを設定します
c1FlexGrid1.Columns[1].DataType = typeof (DateTime);
c1FlexGrid1.Columns[2].DataType = typeof(Boolean);
c1FlexGrid1.Columns[3].DataType = typeof(Decimal);
```

注意:

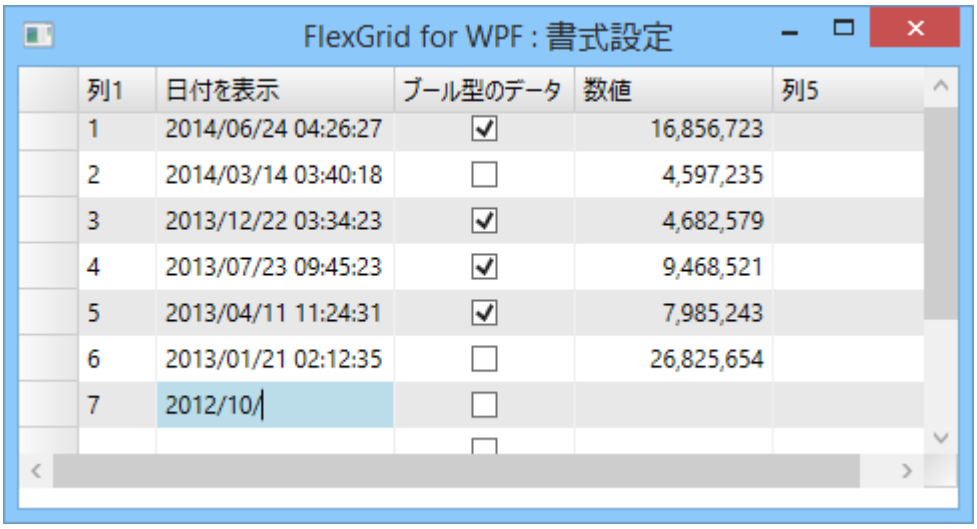
- **GetType**メソッドの詳細は、MSDNを参照してください。
- **DataType**プロパティは設計時に設定することも可能です。
- 行、列両方のデータ型が設定されている場合、列の設定が優先されます。
- 連結（バウンド）モードのグリッドにおける列のデータ型は、C1FlexGrid と接続しているデータソースに依存します。

列の書式を設定する

Format プロパティは列内のデータを書式設定して表示する方法を示します。列単位で書式を設定するには、**Format** プロパティを使用します。

注意:Format プロパティを設定した列の DataType プロパティが、Int32やDouble等の数値型や日付型に設定されていることを確認してください。Format プロパティは数値や日付データの書式を行うための機能ですので、DataType プロパティが Object や String 等に設定されている場合には機能しません。

【実行例】



マークアップ

```
<c1:C1FlexGrid.Columns>
    <c1:Column Header="列1"/>
    <!--列の書式を設定します-->
    <c1:Column DataType="{x:Type DatePicker}" Header="日付を表示" Format="yyyy/MM/dd hh:mm:ss" />
    <c1:Column DataType="{x:Type sys:Boolean}" Header="ブール型のデータ"/>
    <c1:Column Header="数値" Format="#,##0.###"/>
    <c1:Column Header="列5"/>
</c1:C1FlexGrid.Columns>
```

必要に応じて、Formatをコードで設定することもできます。

Visual Basic

```
Public Sub New()
    InitializeComponent()
    ' 列の Format property を設定します
    c1FlexGrid1.Columns(1).DataType = GetType(DateTime)
    c1FlexGrid1.Columns(2).DataType = GetType([Boolean])
    c1FlexGrid1.Columns(3).DataType = GetType([Double])
    ' 列のFormatプロパティを設定します
    c1FlexGrid1.Columns(1).Format = "yyyy/MM/dd hh:mm:ss"
    c1FlexGrid1.Columns(3).Format = "#,##0.###"
End Sub
```

C#

```
    public SetFormat()
{
    InitializeComponent();

 c1FlexGrid1.Columns[1].DataType = typeof(DateTime);
   c1FlexGrid1.Columns[2].DataType = typeof(Boolean);
   c1FlexGrid1.Columns[3].DataType = typeof(Double);
   // 列のFormatプロパティを設定します
   c1FlexGrid1.Columns[1].Format = "yyyy/MM/dd hh:mm:ss";
   c1FlexGrid1.Columns[3].Format = "#,##0.###";
```

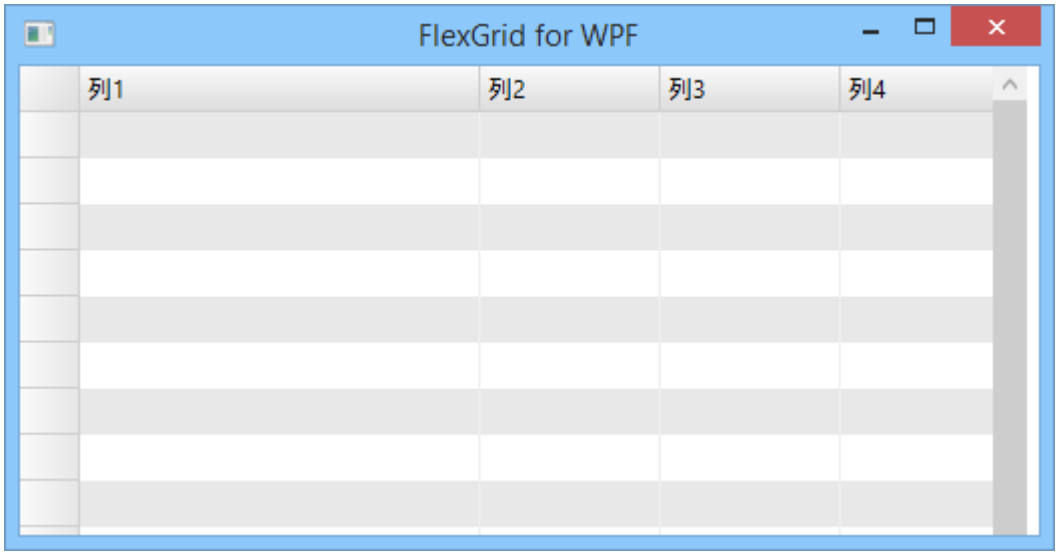
注意:

- GetType メソッドの詳細は、MSDNを参照してください。
- 数値型、日付型の書式指定についての詳細は、MSDNを参照してください。
- DataType プロパティは設計時に設定することも可能です。
- 行、列両方のデータ型・書式が設定されている場合、列の設定が優先されます。
- 連結(バウンド)モードのグリッドにおける列のデータ型は、C1FlexGrid と接続しているデータソースに依存します。

列幅を設定する

Width プロパティを使用して、各列の幅をピクセル単位で設定できます。デフォルト値は GridLength.Auto です。

【実行例】



マークアップ

```
<c1:C1FlexGrid.Columns>
    <c1:Column Width="200" Header="列1" />
    <c1:Column Header="列2"/>
    <c1:Column Header="列3"/>
    <c1:Column Header="列4"/>
    <c1:Column Header="列5"/>
</c1:C1FlexGrid.Columns>
```

必要に応じて、Widthをコードで追加することもできます。

Visual Basic

```
Public Sub New()
    InitializeComponent()
    '列の幅を設定します
    c1FlexGrid1.Columns(1).Width = New GridLength(200)
End Sub
```

C#

```
//列の幅を設定します
c1FlexGrid1.Columns[0].Width = new GridLength(200);
```

列幅の変更を許可する

AllowResizing プロパティを変更して、列幅をマウスで変更できるかどうかを指定できます。列ヘッダの端をダブルクリックすると、最大幅のエントリに合わせて列が自動的にサイズ変更されます。また、各列の**AllowResizing** プロパティを **False** に設定すると、特定列のサイズ変更を禁止できます。

さらに、C1FlexGrid.AllowResizing に AllowResize.Columns フラグ値を含めなければ、Column.AllowResizingの値にかかわらずすべての列幅の変更を禁止できます。

【実行例】

マークアップ

```
<c1:C1FlexGrid Name="c1FlexGrid1" AllowResizing="None">
    <c1:C1FlexGrid.Columns>
        <c1:Column AllowResizing="True" />
        <c1:Column AllowResizing="True" />
        <c1:Column AllowResizing="True" />
        <c1:Column/>
        <c1:Column/>
    </c1:C1FlexGrid.Columns>
</c1:C1FlexGrid>
```

必要に応じて、AllowResizing をコードで設定することもできます。

Visual Basic

```
Public Sub New()
    InitializeComponent()
    '列目の幅の変更を禁止します()
    c1FlexGrid1.Columns(0).AllowResizing = False
End Sub
```

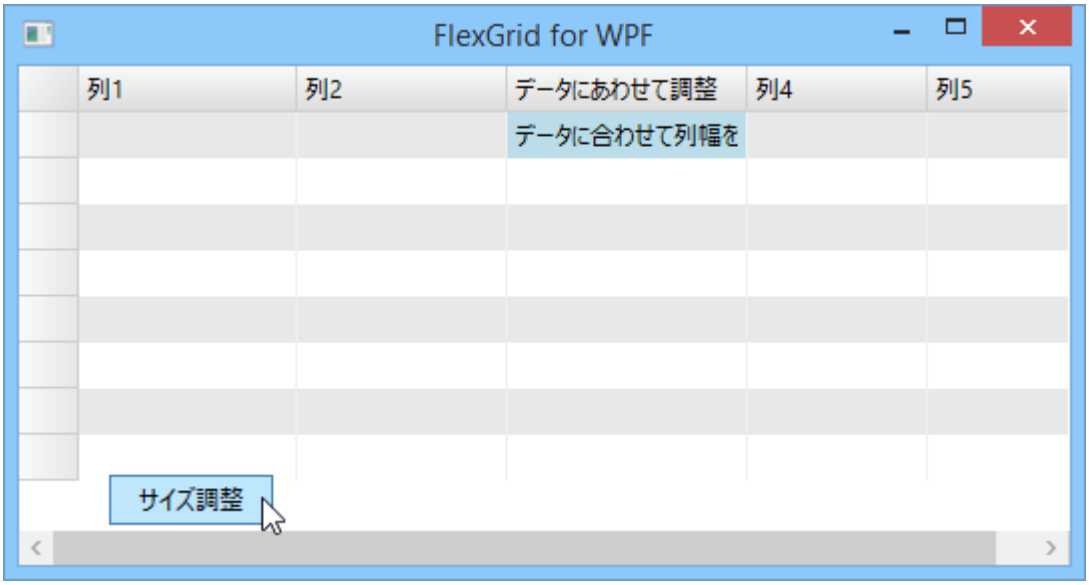
C#

```
//1列目の幅の変更を禁止します
c1FlexGrid1.Columns[0].AllowResizing = false;
```

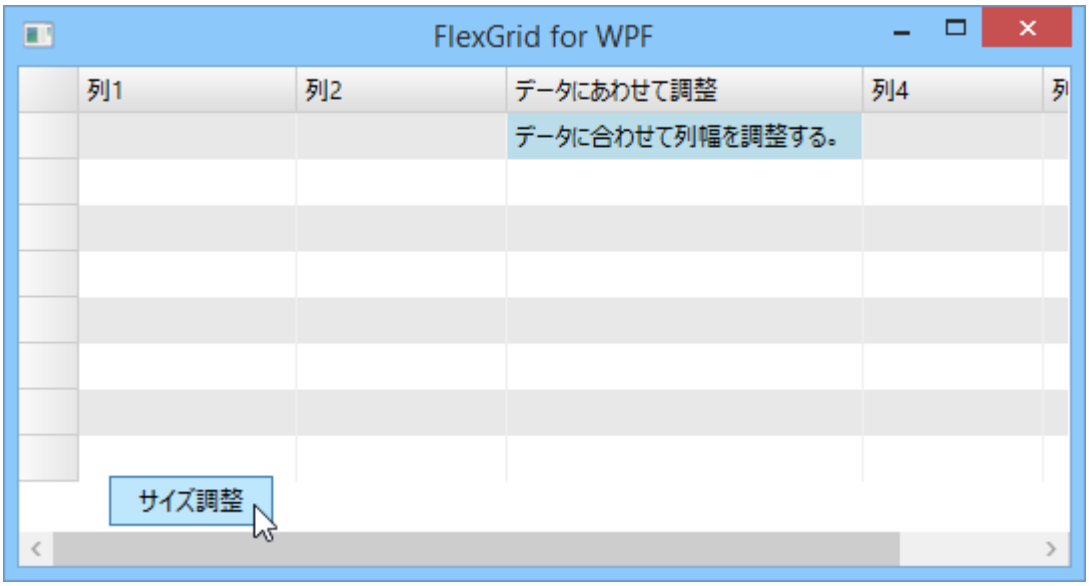
列幅をデータにあわせて調整する

AutoSizeColumn/AutoSizeColumnsメソッドで列のサイズをデータにあわせて調整できます。また、列のサイズ変更が許可されている場合、ユーザーは行／列の境界線をダブルクリックすることで表示されているデータにあわせてサイズを調整できます。

【実行例】



サイズ調整後イメージ



マークアップ

```
<Button Content="サイズ調整" Height="25" HorizontalAlignment="Left" Margin="45,204,0,0"
Name="button1" VerticalAlignment="Top" Width="82" Click="button1_Click" />
```

Visual Basic

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
    '3列目の列幅を調整
    c1FlexGrid1.AutoSizeColumn(2, 5)
End Sub
```

C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //3列目の列幅を調整
    c1FlexGrid1.AutoSizeColumn(2, 5);
}
```

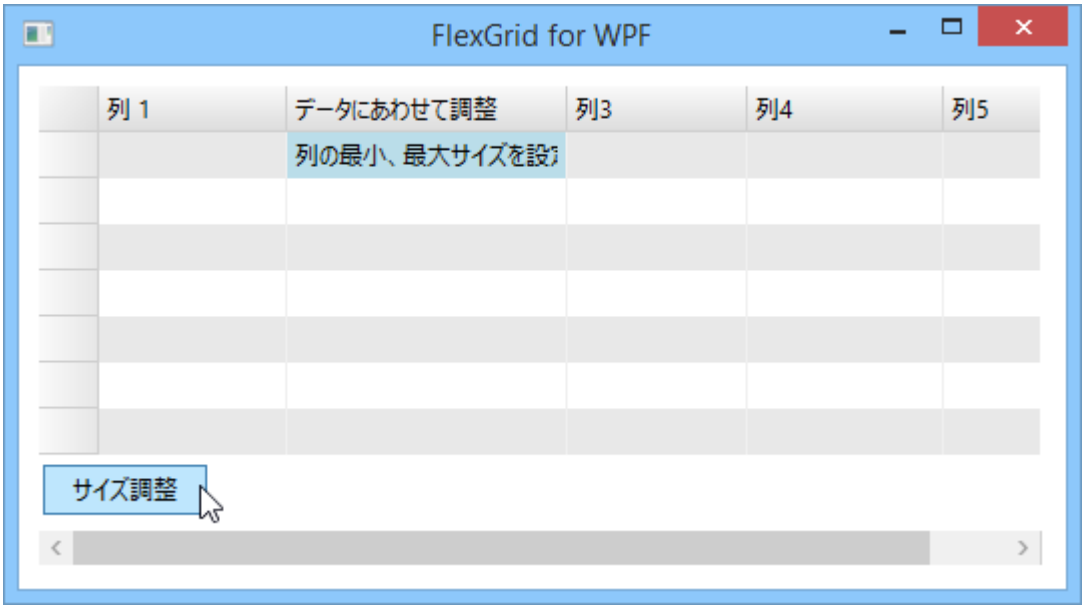
注意:**AutoSizeColumn/AutoSizeColumns**メソッドは列のサイズ変更を禁止している場合でも使用できます。

列幅の最小値／最大値を設定する

MinWidth プロパティで列の最小サイズを、**MaxWidth** プロパティで最大サイズを設定します。この設定は、ユーザーによるサイズ変更時、コードでのサイズ変更時の両方で有効です。

以下の例では、幅の自動調整を行っても列幅は最大サイズ以上にはなりません。

【実行例】



マークアップ

```
<c1:C1FlexGrid.Columns>
    <c1:Column Header="列 1" MaxWidth="Infinity" />
    <!--列幅の最小サイズを50、最大サイズを140にします-->
    <c1:Column Header="データにあわせて調整" MaxWidth="140" MinWidth="50" Width="100" />
    <c1:Column Header="列3"/>
    <c1:Column Header="列4"/>
    <c1:Column Header="列5"/>
</c1:C1FlexGrid.Columns>
```

Visual Basic

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
    '列幅の自動調整を行っても最大サイズ以上にはなりません
    c1FlexGrid1.AutoSizeColumn(1, 10)
End Sub
```

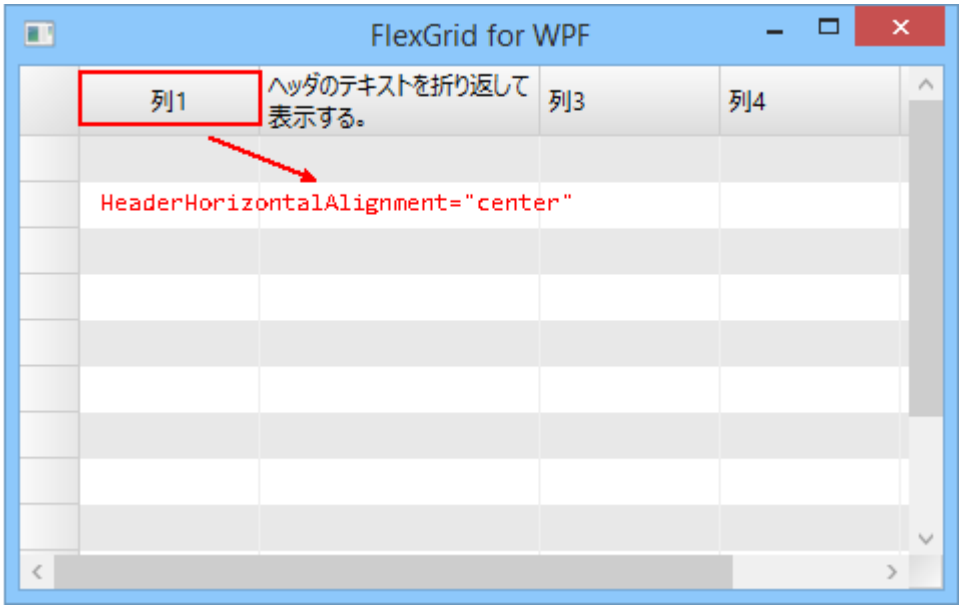
C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //列幅の自動調整を行っても最大サイズ以上にはなりません
    c1FlexGrid1.AutoSizeColumn(1, 10);
}
```

列ヘッダを設定する

列のヘッダセルにキャプションとしてテキストを追加する場合は、**Header** プロパティを使用します。そして、**HeaderHorizontalAlignment**、**HeaderTextWrapping** などのプロパティを使用してヘッダセルを独自にカスタマイズできます。

【実行例】



マークアップ

```
<c1:C1FlexGrid Name="c1FlexGrid1">
    <c1:C1FlexGrid.Rows>
        <c1:Row/>
        <c1:Row/>
        <c1:Row/>
    </c1:C1FlexGrid.Rows>
    <c1:C1FlexGrid.Columns>
        <c1:Column AllowResizing="False" Header="列1"
HeaderHorizontalAlignment="center" TextWrapping="True"/>
        <c1:Column Header="ヘッダのテキストを折り返して表示する。" Width="140"
HeaderTextWrapping="True"/>
        <c1:Column Header="列3"/>
        <c1:Column Header="列4"/>
        <c1:Column Header="列5"/>
    </c1:C1FlexGrid.Columns>
</c1:C1FlexGrid>
```

必要に応じて、Header をコードで設定することもできます。

Visual Basic

```
Public Sub New()
    InitializeComponent()
    c1FlexGrid1.Columns(0).Header = "列1"
    c1FlexGrid1.Columns(0).HeaderHorizontalAlignment = HorizontalAlignment.Center
    c1FlexGrid1.Columns(0).AllowResizing = False
    c1FlexGrid1.Columns(0).TextWrapping = True
    c1FlexGrid1.Columns(1).Header = "ヘッダのテキストを折り返して表示する。"
    c1FlexGrid1.Columns(1).HeaderTextWrapping = True
    c1FlexGrid1.Columns(2).Header = "列3"
    c1FlexGrid1.Columns(3).Header = "列4"
    c1FlexGrid1.Columns(4).Header = "列5"
End Sub
```

C#

```
c1FlexGrid1.Columns[0].Header = "列1";
c1FlexGrid1.Columns[0].HeaderHorizontalAlignment = HorizontalAlignment.Center;
c1FlexGrid1.Columns[0].AllowResizing = false;
c1FlexGrid1.Columns[0].TextWrapping = true;
c1FlexGrid1.Columns[1].Header = "ヘッダのテキストを折り返して表示する。";
c1FlexGrid1.Columns[1].HeaderTextWrapping = true;
c1FlexGrid1.Columns[2].Header = "列3";
c1FlexGrid1.Columns[3].Header = "列4";
c1FlexGrid1.Columns[4].Header = "列5";
```

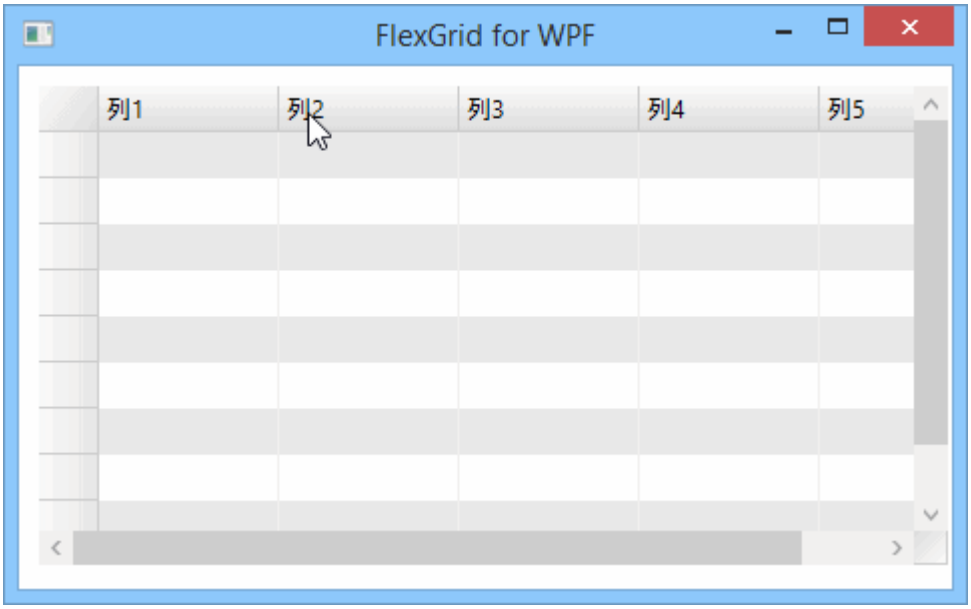
注意:

- 直接セルに値を設定する方法については、「セルの値を設定する」を参照してください。
- C1FlexGrid を連結モードで使用した場合、デフォルトでは Header プロパティ、Name プロパティにはデータソースの列名が設定されます。

列の移動を許可する

AllowDragging プロパティを変更して、列をマウスで新しい位置に移動します。また、列の **AllowDragging** プロパティを **False** に設定すると、特定列のドラッグを禁止できます。

以下の例では、1列目の移動を禁止し、2列目をドラッグします。



マークアップ

```
<c1:C1FlexGrid.Columns>
    <c1:Column AllowDragging="False" Header="列1" />
    <c1:Column Header="列2" />
    <c1:Column Header="列3" />
    <c1:Column Header="列4" />
    <c1:Column Header="列5" />
</c1:C1FlexGrid.Columns>
```

必要に応じて、**AllowDragging** をコードで設定することもできます。

Visual Basic

```
Public Sub New()
    InitializeComponent()
    '1列目の移動を禁止します
    c1FlexGrid1.Columns(0).AllowDragging = False
End Sub
```

C#

```
InitializeComponent();
//1列目の移動を禁止します
c1FlexGrid1.Columns[0].AllowDragging = false;
```

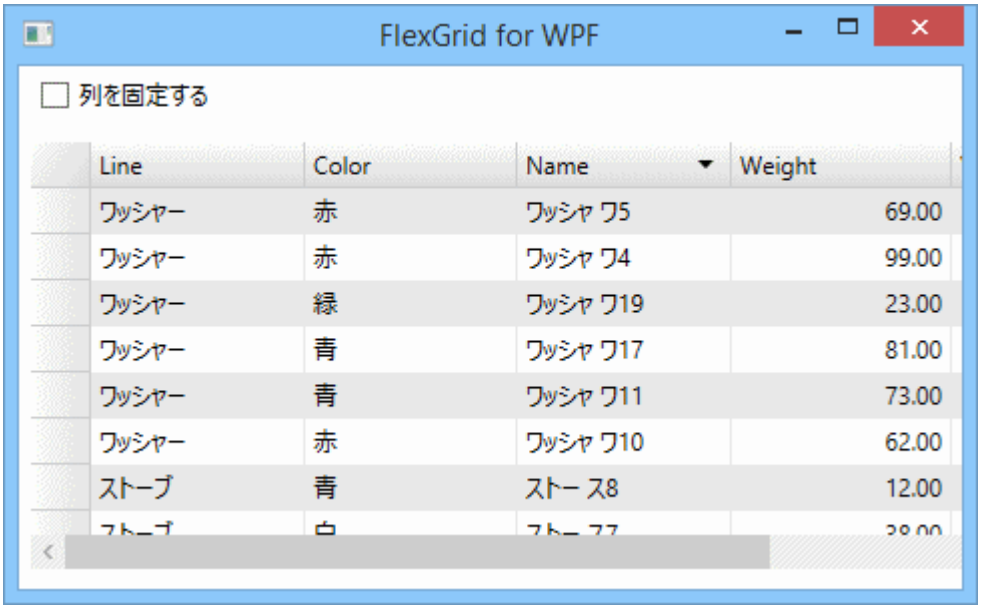
列を固定する

テーブルに多くのデータが表示されている場合、最初のいくつかの行を固定すると、グリッドをスクロールした際にそれらの列が表示されたままになるので便利です。

本動作を **Columns.Frozen** プロパティを設定して簡単に実現できます。**C1FlexGrid** は、Excel と同様にグリッドの固定領域とスクロール可能領域の間に黒い線が表示されています。また、**FrozenLinesBrush** プロパティを使用して、この分割線を削除したり、色を変更することができます。

次のコードは、Excel と同じような FreezePanes コマンドを実装する方法を示します。

【実行例】



ユーザーが _chkFreezePanes チェックボックスをオンにすると、イベントハンドラは **Columns.Frozen** プロパティを設定して、現在の選択範囲の左側に列が固定され、常に表示されるようにします。次の図に、この効果を示します。

Visual Basic

```
Private Sub _chkFreezePanes_Click(sender As Object, e As RoutedEventArgs) Handles
_chkFreezePanes.Click
    If _checkfreezepanes.IsChecked.Value Then
        C1FlexGrid.Columns.Frozen = C1FlexGrid.Selection.Column
    Else
        C1FlexGrid.Columns.Frozen = 0
    End If
End Sub
```

C#

```
//列を固定するか固定を解除をします。
private void _chkFreezePanes_Click(object sender, RoutedEventArgs e)
{
    if (_chkFreezePanes.IsChecked.Value)
    {
        C1FlexGrid.Columns.Frozen = C1FlexGrid.Selection.Column;
    }
    else
    {
        C1FlexGrid.Columns.Frozen = 0;
    }
}
```

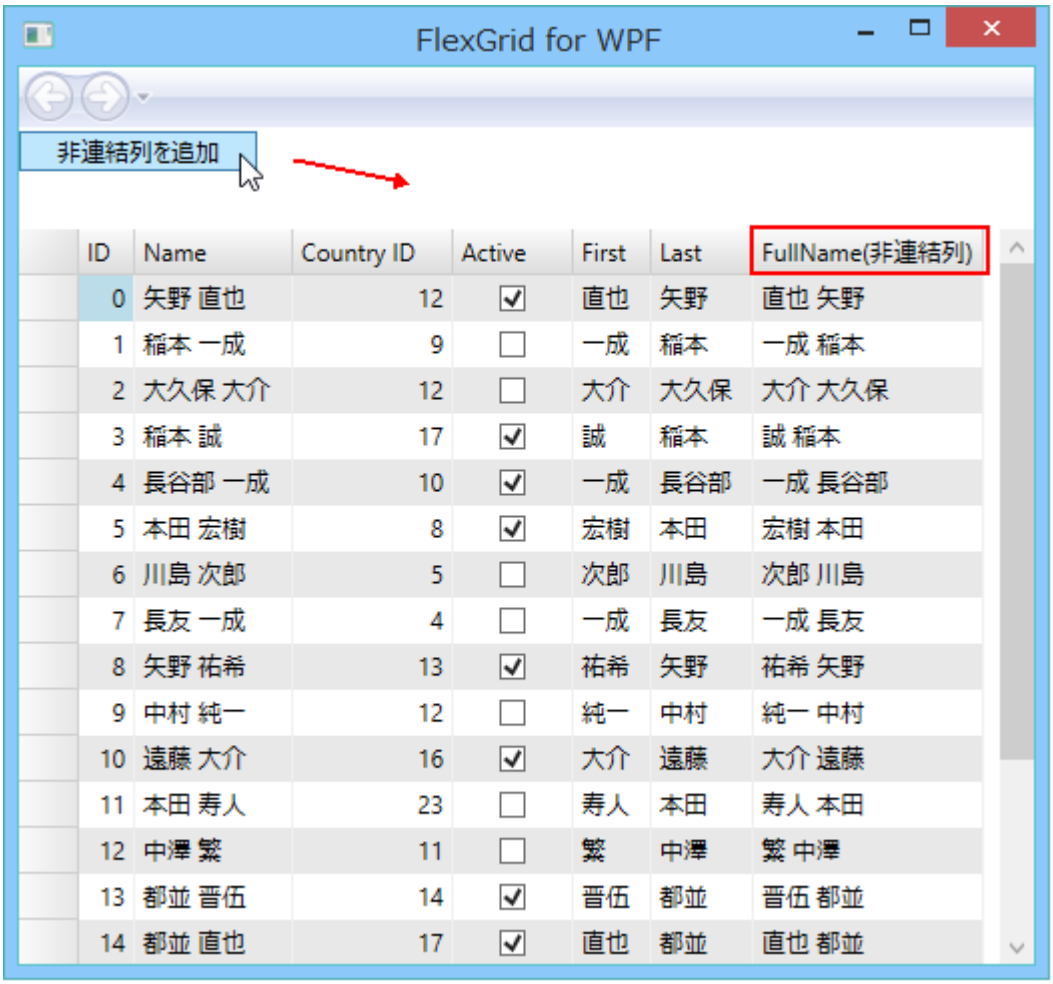
非連結列を追加する

C1FlexGrid を連結モードで使用していると、連結しているデータソースのデータがグリッドに表示されます。このとき、データソースに存在しない列を使ってデータを表示したい場合は、非連結列を使用します。非連結列は、グリッドにデータソースを連結した後に、デザイン時またはコードにより追加できます。

FlexGrid for WPF では、カスタムセルを作成するには、ICellFactory インターフェイスを実装するクラスを作成し、そのクラスのインスタンスを CellFactory プロパティに割り当てる必要があります。即ち、次のコードのように、**CellFactory** クラスを継承してカスタムセルファクトリークラスを作成し、その **CreateCellContent** メソッドでアンバウンド列の表示を制御します。

> **注意**:WPF版ではGetUnboundValueやOwnerDrawCellイベントが提供されません。代わりに、充実したCellFactoryクラスを使用します。

【実行例】



次のコードは、**CellFactory** クラスを継承して、追加する非連結列のセルの内容をカスタマイズする方法を示します。

Visual Basic

```
'CellFactoryクラスを継承します
Class CustomCellFactory
    Inherits CellFactory
    Public Overrides Sub CreateCellContent(grid As C1FlexGrid, bdr As Border, rng
As CellRange)
        ' 既定の処理を実行します
        MyBase.CreateCellContent(grid, bdr, rng)
        ' アンバウンド列の場合は、セルの内容をカスタマイズします
        If grid.Columns(rng.Column).Header = "FullName(非連結列)" Then
            Dim first As String = grid(rng.Row, "First").ToString()
            Dim last As String = grid(rng.Row, "Last").ToString()
            bdr.Child = New TextBlock() With { _
               Text = Convert.ToString(first & Convert.ToString(" ")) & last, _
               VerticalAlignment = VerticalAlignment.Center _
            }
        End If
    End Sub
End Class
```

C#

Example Title

```
//CellFactoryクラスを継承します
class CustomCellFactory : CellFactory
{
    public override void CreateCellContent(C1FlexGrid grid, Border bdr, CellRange
rng)
    {
        // 既定の処理を実行します
        base.CreateCellContent(grid, bdr, rng);

        // アンバウンド列の場合は、セルの内容をカスタマイズします
        if (grid.Columns[rng.Column].Header == "FullName(非連結列)")
        {
            string first = grid[rng.Row, "First"].ToString();
            string last = grid[rng.Row, "Last"].ToString();
            bdr.Child = new TextBlock()
            {
                Text = first + " " + last,
                VerticalAlignment = VerticalAlignment.Center
            };
        }
    }
}
```

これで、ボタンクリックで非連結列を作成し、グリッドの最終列に追加します。

Visual Basic

Example Title

```
Private Sub Button_Click_1(sender As Object, e As RoutedEventArgs)
    ' 非連結列を作成し、グリッドの最終列に追加します。
    Dim col = New Column()
    col.Header = "FullName(非連結列)"
    C1FlexGrid1.Columns.Add(col)
    col.IsReadOnly = True
    ' 非連結列の幅を調整します。
    Me.C1FlexGrid1.AutoSizeColumns(col.Index, col.Index, 0, True, True)
End Sub
```

C#

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    // 非連結列を作成し、グリッドの最終列に追加します。
    var col = new Column();
    col.Header = "FullName(非連結列)";
    C1FlexGrid1.Columns.Add(col);
    col.IsReadOnly = true;

    // 非連結列の幅を調整します。
    this.C1FlexGrid1.AutoSizeColumns(col.Index, col.Index, 0, true, true);
}
```

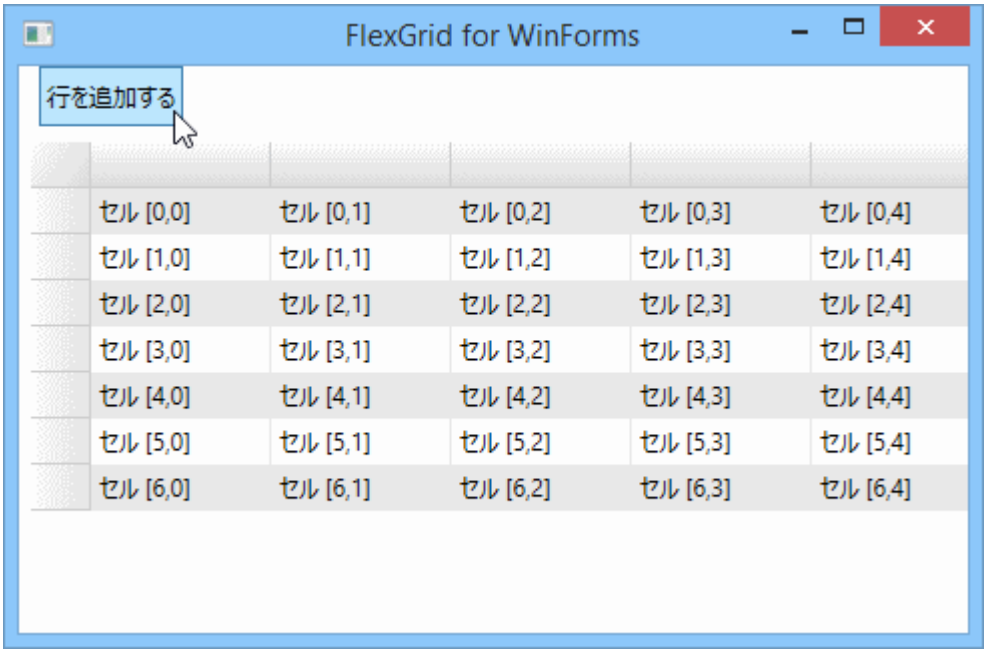
行

行を追加する

グリッドで行を追加するには、**RowCollection** クラスの Add メソッドを使用します。たとえば、次のコードでは、**C1FlexGrid** を非連結モードで作成していることを前提として、ボタンクリックで行を追加する方法を示します。

> **注意**:FlexGrid for WPF では Count プロパティは読み取り専用です。また、AddItem メソッドは提供されていません。

【実行例】



Visual Basic

```
Private Sub Button_Click_1(sender As Object, e As RoutedEventArgs)
    ' 行を追加します。C1FlexGridのRowsプロパティはRowCollection型です。
    Dim i As Integer = 0
    While i > 1
        C1FlexGrid.Rows.Add(New Row())
        i += 1
    End While
    Dim r As New Row()
    C1FlexGrid.Rows.Add(r)

    For j As Integer = 0 To C1FlexGrid.Rows.Count - 1
        For c As Integer = 0 To C1FlexGrid.Columns.Count - 1
            C1FlexGrid(j, c) = String.Format("セル [{0},{1}]", j, c)
        Next
    Next
End Sub
```

C#

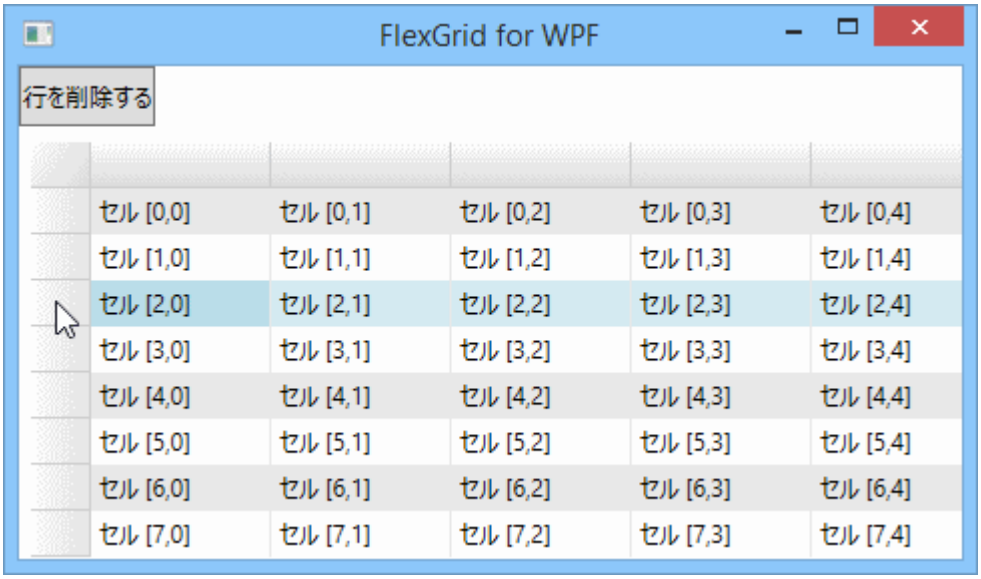
```
private void Button_Click(object sender, RoutedEventArgs e)
{
    // 行を追加します。C1FlexGridのRowsプロパティはRowCollection型です。
    for (int i = 0; i > 1; i++)
    {
        C1FlexGrid.Rows.Add(new Row());
    }
    Row r = new Row();
    C1FlexGrid.Rows.Add(r);

    for (int j = 0;j < C1FlexGrid.Rows.Count; j++)
    {
        for (int c = 0; c < C1FlexGrid.Columns.Count; c++)
        {
            C1FlexGrid[j, c] = string.Format("セル [{0},{1}]", j, c);
        }
    }
}
```

行を削除する

グリッドで行を削除するには、**RowCollection** クラスの Remove メソッドを使用します。たとえば、次のコードは、**FlexGrid** を非連結モードで作成していることを前提として、ボタンクリックで行を削除する方法を示します。

> **注意:**FlexGrid for WPF では Count プロパティは読み取り専用です。また、RemoveItem メソッドは提供されていません。

[実行例]



Visual Basic

```
Private Sub Button_Click_1(sender As Object, e As RoutedEventArgs)
    Dim lastRow = C1FlexGrid.Rows.LastOrDefault()
    If lastRow IsNot Nothing Then
        '選択した行を削除します。C1FlexGridのRowsプロパティはRowCollection型です。
        C1FlexGrid.Rows.RemoveAt(C1FlexGrid.Selection.Row)
    End If
End Sub
```

C#

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    var lastRow = C1FlexGrid.Rows.LastOrDefault();
    if (lastRow != null)
    {
        //選択した行を削除します。C1FlexGridのRowsプロパティはRowCollection型です。
        C1FlexGrid.Rows.RemoveAt(C1FlexGrid.Selection.Row);
    }
}
```

新規追加行を表示する

FlexGrid for WPF で新規行の追加を許可するには、**AllowAddNew** プロパティを **True** に変更します。行の追加を許可すると、最終行に新規追加のテンプレート行が表示されます。また、**CellFactory** クラスを継承したカスタムセルファクトリを使用して新規追加のテンプレート行に任意の文字列を表示し、ユーザーに新規行への入力テキストをプロンプトすることも可能です。

次のコードでは、グリッドをデータソースと連結していることを前提として、**AllowAddNew** プロパティおよび **CellFactory** クラスの使用方法を示します。

> **注意**:FlexGrid for WPF の場合は NewRowWatermark プロパティの代わりに CellFactory クラスを使用する方法が提供されています。

【実行例】

Visual Basic

```
Public Sub New()
    InitializeComponent()
    Dim p = Product.GetProducts(10)
    C1FlexGrid.ItemsSource = p
    'グリッドのAllowAddNewプロパティを設定します
    C1FlexGrid.AllowAddNew = True
    'カスタムセルファクトリを作成します
    C1FlexGrid.CellFactory = New CustomCellFactory()
End Sub
```

C#

```
//グリッドのAllowAddNewプロパティを設定します
C1FlexGrid.AllowAddNew = true;

//カスタムセルファクトリを作成します
C1FlexGrid.CellFactory = new MyCellFactory();
```

次のように、カスタムセルファクトリを使用して新しい行への入力をプロンプトするマスクを設定します。

Visual Basic

```
Private Class CustomCellFactory
    Inherits CellFactory
    Public Overrides Sub CreateCellContent(grid As C1FlexGrid, bdr As Border, rng
As CellRange)
        '新しい行を追加する場合、以下のマスクを設定します
        MyBase.CreateCellContent(grid, bdr, rng)
        If rng.Column = 0 AndAlso rng.Row = grid.Rows.Count - 1 Then
            Dim txt = TryCast(bdr.Child, TextBlock)
            If txt IsNot Nothing Then
                txt.Text = "新しいレコードを追加してください..."
                txt.Opacity = 0.65
                txt.FontStyle = FontStyles.Italic
                txt.IsHitTestVisible = False

            End If
        End If
    End Sub
End Class
```

C#

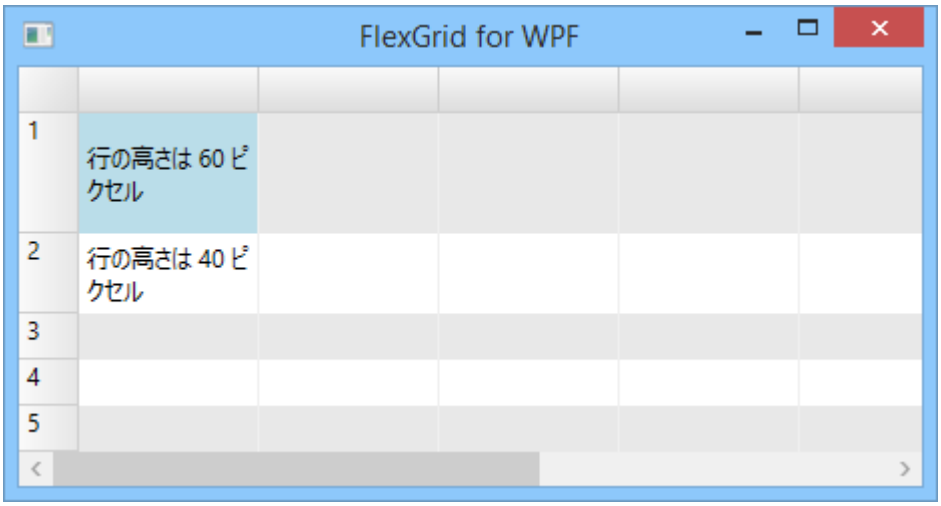
```
class MyCellFactory : CellFactory
{
    public override void CreateCellContent(C1FlexGrid grid, Border bdr, CellRange
rng)
    {
        //新しい行を追加する場合、以下のマスクを設定します
        base.CreateCellContent(grid, bdr, rng);
        if (rng.Column == 0 && rng.Row == grid.Rows.Count - 1)
        {
            var txt = bdr.Child as TextBlock;
            if (txt != null)
            {
                txt.Text = "新しいレコードを追加してください...";
                txt.Opacity = 0.65;
                txt.FontStyle = FontStyles.Italic;
                txt.IsHitTestVisible = false;
            }
        }
    }
}
```

行の高さを設定する

Height プロパティを使用して、各行の高さをピクセル単位で設定できます。デフォルトの高さを使用する場合は「-1」を設定します。

また、**Row** クラスから継承される **BoundRow** クラスと **GroupRow** クラスのHeight プロパティも同じように設定できます。

【実行例】



マークアップ

```
<c1:C1FlexGrid.Rows>
    <c1:Row Height="60" TextWrapping="true"/>
    <c1:Row Height="40" TextWrapping="true"/>
    <c1:Row/>
    <c1:Row/>
    <c1:Row/>
</c1:C1FlexGrid.Rows>
```

必要に応じて、Height をコードで追加することもできます。

Visual Basic

```
Public Sub New()
    InitializeComponent()
    '行の高さを設定します
    C1FlexGrid.Rows(0).Height = 60
    C1FlexGrid.Rows(0).Height = 40
    Me.C1FlexGrid(0, 0) = "行の高さは 60 ピクセル"
    Me.C1FlexGrid(1, 0) = "行の高さは 40 ピクセル"
End Sub
```

C#

Example Title

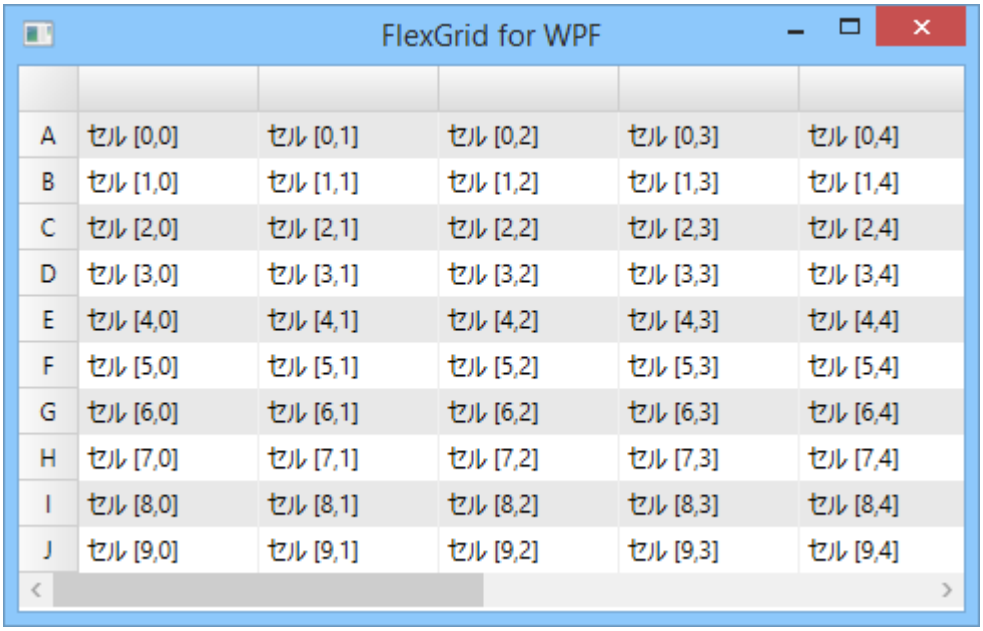
```
//行の高さを設定します
C1FlexGrid.Rows[0].Height = 60;
C1FlexGrid.Rows[0].Height = 40;
this.C1FlexGrid[0, 0] = "行の高さは 60 ピクセル";
this.C1FlexGrid[1, 0] = "行の高さは 40 ピクセル";
```

行ヘッダを設定する

行ヘッダセルにテキストをキャプションとして追加するために、**CellFactory** クラスを継承して、ヘッダセル上にカスタムテキストを表示できます。

次の例では、非連結グリッドを作成していることを前提として、カスタムセルファクトリを使用する方法を示します。

【実行例】



Visual Basic

```
'CellFactoryクラスを継承してカスタムセルファクトリを使用します
Class Custom1CellFactory
    Inherits CellFactory
    Public Overrides Sub CreateRowHeaderContent(grid As C1FlexGrid, bdr As Border,
range As CellRange)
        Dim tb = New TextBlock()
        tb.HorizontalAlignment = HorizontalAlignment.Center
        tb.VerticalAlignment = VerticalAlignment.Center
        tb.Text = GetAlphaRowHeader(range.Row)
        bdr.Child = tb
    End Sub
    'A,B..などスタイルの行ヘッダを作成します
    Private Function GetAlphaRowHeader(p As Integer) As String
        Return GetAlphaRowHeader(p, String.Empty)
    End Function
    Private Function GetAlphaRowHeader(p As Integer, s As String) As String
        Dim [rem] = p Mod 26
        s = ChrW(CInt(Asc("A")) + [rem]) & s
        p = p / 26 - 1
        Return If(p < 0, s, GetAlphaRowHeader(p, s))
    End Function
End Class
```

C#

```
//CellFactoryクラスを継承してカスタムセルファクトリを使用します
class Custom1CellFactory : CellFactory
  {
     public override void CreateRowHeaderContent(C1FlexGrid grid, Border bdr,
CellRange range)
       {
          var tb = new TextBlock();
          tb.HorizontalAlignment = HorizontalAlignment.Center;
          tb.VerticalAlignment = VerticalAlignment.Center;
          tb.Text = GetAlphaRowHeader(range.Row);
          bdr.Child = tb;
       }
          //A,B..などスタイルの行ヘッダを作成します
         string GetAlphaRowHeader(int p)
          {
            return GetAlphaRowHeader(p, string.Empty);
          }
         string GetAlphaRowHeader(int p, string s)
          {
            var rem = p % 26;
            s = (char)((int)'A' + rem) + s;
            p = p / 26 - 1;
            return p < 0 ? s : GetAlphaRowHeader(p, s);
          }
       }
```

また、行ヘッダ内の各セルにアクセスするため、**RowHeaders** という追加のプロパティが提供されています。たとえば、次のコードでは一番上の行ヘッダをアクセスします。

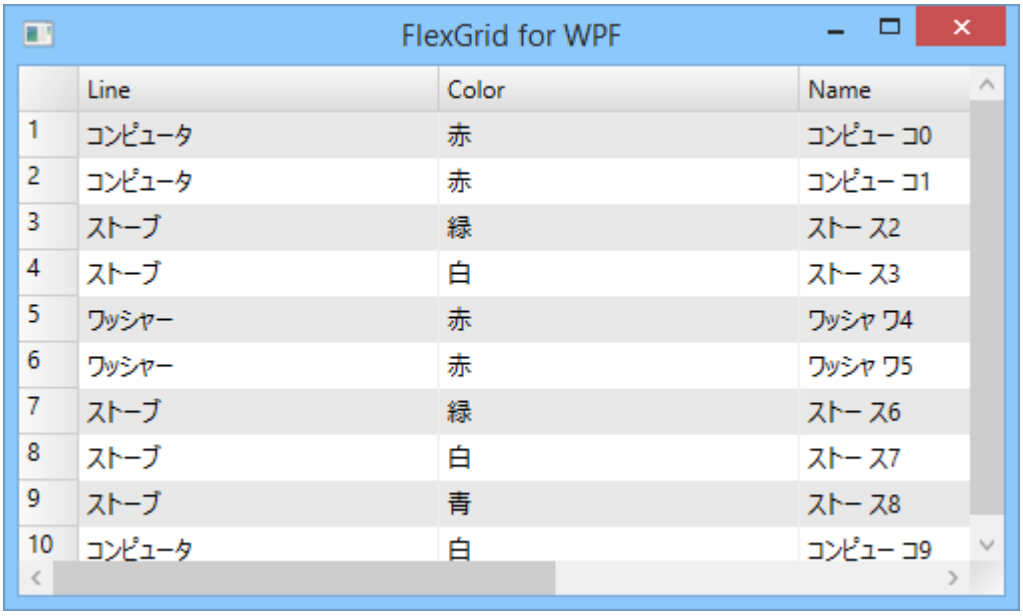
```
int row = 0;
C1FlexGrid.RowHeaders[row, 0] = "行ヘッダ";
```

行番号を表示する

FlexGrid for WPF で **CellFactory** クラスを使用することでオートナンバーと同様の処理を単純に実現できます。また、**CreateRowHeaderContent** メソッドをオーバーライドして固定行の場合も行番号を表することが可能です。

次のコードでは、グリッドをデータソースと連結していることを前提として、**CellFactory** クラスを継承したカスタムセルファクトリに行番号を設定する処理を追加します。

【実行例】



Visual Basic

```
'CellFactoryクラスを継承して、カスタムセルファクトリを使用します
Private Sub ProcessFlexGrid(C1FlexGrid As C1.WPF.FlexGrid.C1FlexGrid)
    C1FlexGrid.CellFactory = New Custom1CellFactory()
End Sub
Private Class Custom1CellFactory
    Inherits CellFactory
    Public Overrides Sub CreateRowHeaderContent(grid As C1FlexGrid, bdr As Border,
range As CellRange)
        Dim row = grid.Rows(range.Row)
        Dim textblock__1 = New TextBlock()
        Dim binding = New Binding() With { _
            Key .Source = row, _
            Key .Path = New PropertyPath("Index"), _
            Key .Converter = New IndexValueConverter() _
        }
        textblock__1.SetBinding(TextBlock.TextProperty, binding)
        bdr.Child = textblock__1
    End Sub
End Class
'行インデックスを1から表示するように設定します
Public Class IndexValueConverter
    Implements IValueConverter
    Private Function Convert(value As Object, targetType As Type, parameter As
Object, culture As CultureInfo) As Object Implements IValueConverter.Convert
        Dim index As Integer = CInt(value)
        Return index + 1
End Function
    Private Function ConvertBack(value As Object, targetType As Type, parameter As
Object, culture As CultureInfo) As Object Implements IValueConverter.ConvertBack
        Throw New NotImplementedException()
    End Function
End Class
```

C#

```
//CellFactoryクラスを継承して、カスタムセルファクトリを使用します
private void ProcessFlexGrid(C1.WPF.FlexGrid.C1FlexGrid C1FlexGrid)
{
    C1FlexGrid.CellFactory = new Custom1CellFactory();
}
class Custom1CellFactory : CellFactory
{
    public override void CreateRowHeaderContent(C1FlexGrid grid, Border bdr,
CellRange range)
    {
        var row = grid.Rows[range.Row];
        var textblock = new TextBlock();
        var binding = new Binding() { Source = row, Path = new
PropertyPath("Index"), Converter = new IndexValueConverter() };
        textblock.SetBinding(TextBlock.TextProperty, binding);
        bdr.Child = textblock;
    }
}
//行インデックスを1から表示するように設定します
public class IndexValueConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
    {
        int index = (int)value;
        return index + 1;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

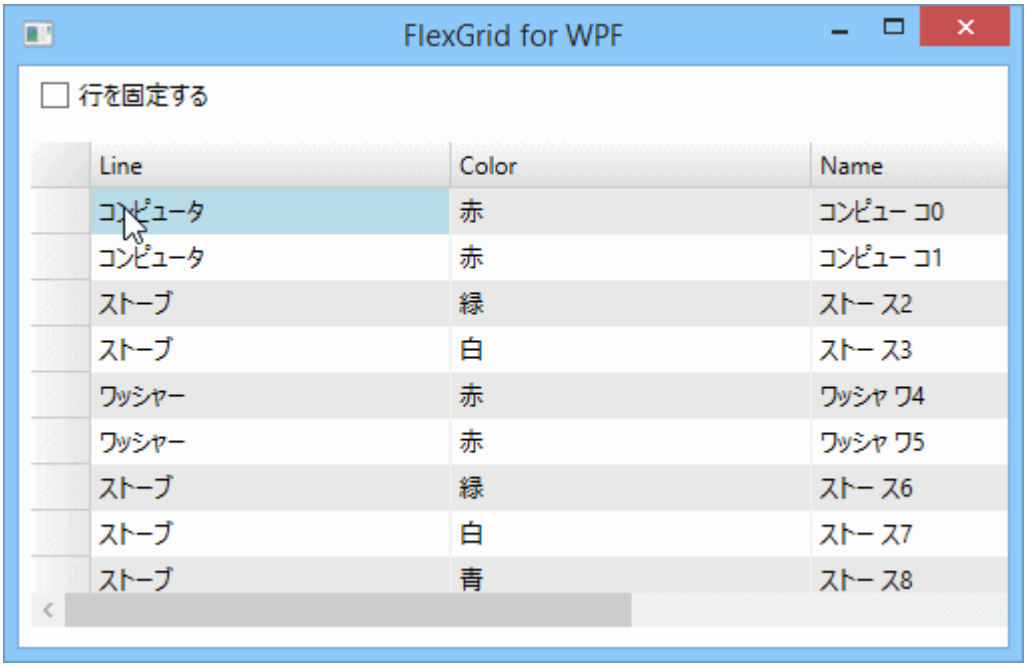
行を固定する

テーブルに多くのデータが表示されている場合、最初のいくつかの行を固定すると、グリッドをスクロールした際にそれらの行が表示されたままになるので便利です。

本動作を **Rows.Frozen** プロパティを設定して簡単に実現できます。**C1FlexGrid** では、Excel と同様にグリッドの固定領域とスクロール可能領域の間に黒い線が表示されています。また、**FrozenLinesBrush** プロパティを使用して、この分割線を削除したり、色を変更することができます。

次のコードは、Excel と同じような FreezePanes コマンドを実装する方法を示します。

【実行例】



ユーザーが _chkFreezePanes チェックボックスをオンにすると、イベントハンドラは **Rows.Frozen** プロパティを設定して、現在の選択範囲の上側に行が固定され、常に表示されるようにします。次の図に、この効果を示します。

Visual Basic

Visual Basic

```
'選択した行を固定します
Private Sub _chkFreezePanes_Click(sender As Object, e As RoutedEventArgs)
    If _chkFreezePanes.IsChecked.Value Then
        C1FlexGrid.Rows.Frozen = C1FlexGrid.Selection.Row
    Else
        C1FlexGrid.Rows.Frozen = 0
    End If
End Sub
```

C#

C#

```
//選択した行を固定します
private void _chkFreezePanes_Click(object sender, RoutedEventArgs e)
{
    if (_chkFreezePanes.IsChecked.Value)
    {
        C1FlexGrid.Rows.Frozen = C1FlexGrid.Selection.Row;
    }
    else
    {
        C1FlexGrid.Rows.Frozen = 0;
    }
}
```

行詳細テンプレート

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。FlexGrid では、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。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。この例では、行詳細テンプレートを使用して、製品関連の情報を FlexGrid に表示します。

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。

次のセクションでは、FlexGrid .NET 4.5.2および.NET 5 バージョンで行の詳細を実装する方法を学習します。

.NET Framework

FlexGrid で行詳細テンプレートを追加するには

1. WPF アプリケーションで、XAML デザイナにグリッドコントロールをドラッグします。
2. グリッドに、Product ID、Product Name、Order Date データフィールドを表示する 3 つの列を作成します。

XAML

```
<c1:C1FlexGrid.Columns>
    <c1:Column Header="Product ID" Binding="{Binding ProductId}" Width="75" />
    <c1:Column Header="Product Name" Binding="{Binding ProductName}"
Width="150" />
    <c1:Column Header="Order Date" Binding="{Binding OrderDate}" Width="300" />
</c1:C1FlexGrid.Columns>
```

3. XAML で、DockPanel に 1 つの画像コントロールと 6 つのテキストブロックコントロールが埋め込まれた行詳細テンプレートを作成します。

XAML

```
<c1:C1FlexGrid.RowDetailsTemplate>
      <DataTemplate>
          <DockPanel Background="GhostWhite">
              <Image DockPanel.Dock="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, StringFormat=d}"
                             Grid.Column="1" Grid.Row="2" />
              </Grid>
          </DockPanel>
      </DataTemplate>
  </c1:C1FlexGrid.RowDetailsTemplate>
```

4. MainWindow.xaml.cs ファイルで、グリッドにデータを追加するためのクラス User を作成します。

```
Public Class User
    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

    public class User
    {
        public int ProductId { get; set; }

        public string ProductName { get; set; }

        public DateTime OrderDate { get; set; }

        public ImageSource ImgSource{ get; set; }

    }
```

5. リストを作成してグリッドにデータを挿入し、**ItemsSource** プロパティを使用してグリッドをリストに連結します。

```
    Public Sub New()
        InitializeComponent()

        'Create a list
        Dim users As New List(Of User)()

        'Add items to the list
        users.Add(New User() With {
        .ProductId = 101,
```

```
    .ProductName = "Beverages",
    .OrderDate = New DateTime(1971, 7, 23),
    .ImgSource = New BitmapImage(New
Uri("pack://application:,,,/Resources/Beverage.png"))
})
    users.Add(New User() With {
    .ProductId = 102,
    .ProductName = "Condiments",
    .OrderDate = New DateTime(1974, 1, 17),
    .ImgSource = New BitmapImage(New
Uri("pack://application:,,,/Resources/Condiments.png"))
})
    users.Add(New User() With {
    .ProductId = 103,
    .ProductName = "Confections",
    .OrderDate = New DateTime(1991, 9, 2),
    .ImgSource = New BitmapImage(New
Uri("pack://application:,,,/Resources/Confections.png"))
})
    users.Add(New User() With {
    .ProductId = 104,
    .ProductName = "Poultry",
    .OrderDate = New DateTime(1991, 10, 24),
    .ImgSource = New BitmapImage(New
Uri("pack://application:,,,/Resources/Poultry.png"))
})

    'Populate the grid
    grid.ItemsSource = users
End Sub

public MainWindow()
{
    InitializeComponent();

    //Create a list
    List<User> users = new List<User>();

    //Add items to the list
    users.Add(new User() { ProductId = 101, ProductName = "Beverages",
OrderDate = new DateTime(1971, 7, 23),
        ImgSource = new BitmapImage(new
Uri(@"pack://application:,,,/Resources/Beverage.png")) });
    users.Add(new User() { ProductId = 102, ProductName = "Condiments",
OrderDate = new DateTime(1974, 1, 17),
        ImgSource = new BitmapImage(new
Uri(@"pack://application:,,,/Resources/Condiments.png")) });
    users.Add(new User() { ProductId = 103, ProductName = "Confections",
OrderDate = new DateTime(1991, 9, 2),
        ImgSource = new BitmapImage(new
Uri(@"pack://application:,,,/Resources/Confections.png")) });
    users.Add(new User() { ProductId = 104, ProductName = "Poultry", OrderDate
= new DateTime(1991, 10, 24),
        ImgSource = new BitmapImage(new
Uri(@"pack://application:,,,/Resources/Poultry.png")) });

    //Populate the grid
    grid.ItemsSource = users;
}
```

.NET

FlexGrid で行詳細テンプレートを追加するには

1. WPF コアアプリケーションで、XAML デザイナにグリッドコントロールをドラッグします。
2. グリッドに、**ProductID**、**Product Name**、**Order Date** データフィールドを表示する 3 つの列を作成します。

XAML

```
<c1:FlexGrid.Columns>
    <c1:GridColumn Header="ProductID" Binding="ProductId" Width="75" />
    <c1:GridColumn Header="Product Name" Binding="ProductName" Width="150" />
    <c1:GridColumn Header="Order Date" Binding="OrderDate" Width="300" />
</c1:FlexGrid.Columns>
```

3. XAML で、DockPanel に 1 つの画像コントロールと 6 つのテキストブロックコントロールが埋め込まれた行詳細テンプレートを作成します。

XAML

```
<DataTemplate>
      <DockPanel Background="GhostWhite">
          <Image DockPanel.Dock="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, StringFormat=d}"
           Grid.Column="1" Grid.Row="2" />
          </Grid>
      </DockPanel>
  </DataTemplate>
```

4. MainWindow.xaml.cs ファイルで、グリッドにデータを追加するためのクラス User を作成します。

C#

```
public class User
{
    public int ProductId { get; set; }
    public string ProductName { get; set; }
    public DateTime OrderDate { get; set; }
```

```
    public ImageSource ImgSource { get; set; }

}
```

5. リストを作成してグリッドにデータを挿入し、**ItemsSource** プロパティを使用してグリッドをリストに連結します。

C#

```
InitializeComponent();

//Create a list
List<User> users = new List<User>();

//Add items to the list
users.Add(new User()
{
    ProductId = 101,
    ProductName = "Beverages",
    OrderDate = new DateTime(1971, 7, 23),
    ImgSource = new BitmapImage(new
Uri(@"pack://application:,,,/Resources/Beverage.png"))
});
users.Add(new User()
{
    ProductId = 102,
    ProductName = "Condiments",
    OrderDate = new DateTime(1974, 1, 17),
    ImgSource = new BitmapImage(new
Uri(@"pack://application:,,,/Resources/Condiments.png"))
});
users.Add(new User()
{
    ProductId = 103,
    ProductName = "Confections",
    OrderDate = new DateTime(1991, 9, 2),
    ImgSource = new BitmapImage(new
Uri(@"pack://application:,,,/Resources/Confections.png"))
});
users.Add(new User()
{
    ProductId = 104,
    ProductName = "Poultry",
    OrderDate = new DateTime(1991, 10, 24),
    ImgSource = new BitmapImage(new
Uri(@"pack://application:,,,/Resources/Poultry.png"))
});

//Populate the grid
grid.ItemsSource = users;
```

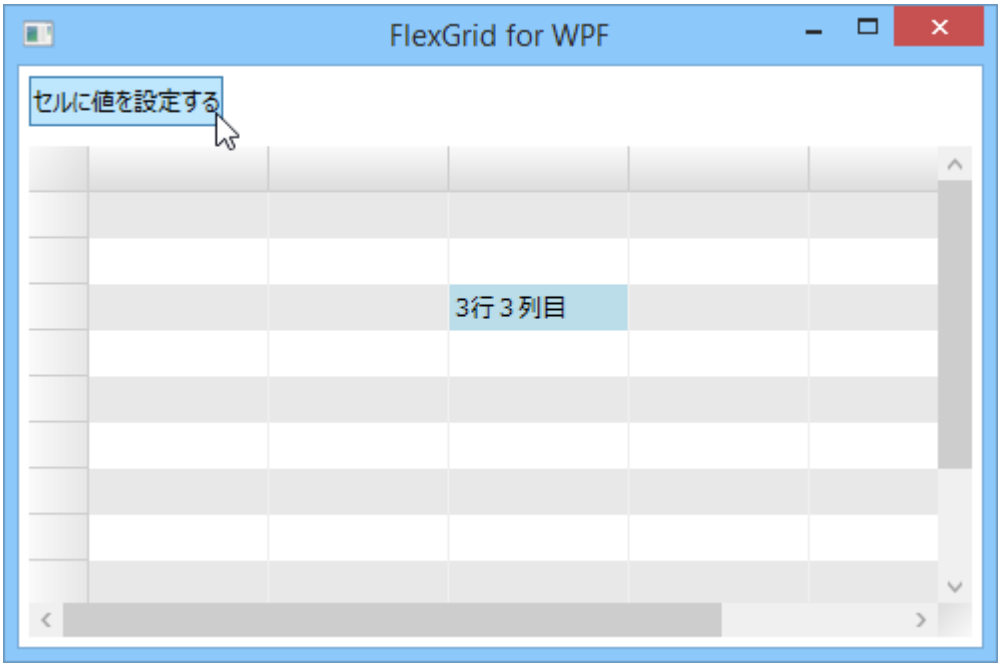
セル

セルの値を設定する

C1FlexGrid 内の各セルに最も簡単にアクセスするために、セルのインデクサを使用します。たとえば、次のサンプルコードは、グリッドの3行3列目のセルに値を設定します。

> **注意**:FlexGrid for WPF では WinForms のような SetDataメソッドは機能しませんので、ご注意ください。

【実行例】



Visual Basic

```
Private Sub Button_click(sender As Object, e As RoutedEventArgs)
    ' 3行3列目のセルに値を設定します。(インデックスは0からです)
    C1FlexGrid(3, 3) = "3行3列目"
    'セルの移動(選択)を設定します
    C1FlexGrid.Select(3, 3)
End Sub
```

C#

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    //3行3列目のセルに値を設定します。(インデックスは0からです)
    C1FlexGrid[3, 3] = "3行3列目";

    //セルの移動(選択)を設定します
    C1FlexGrid.Select(3, 3);
}
```

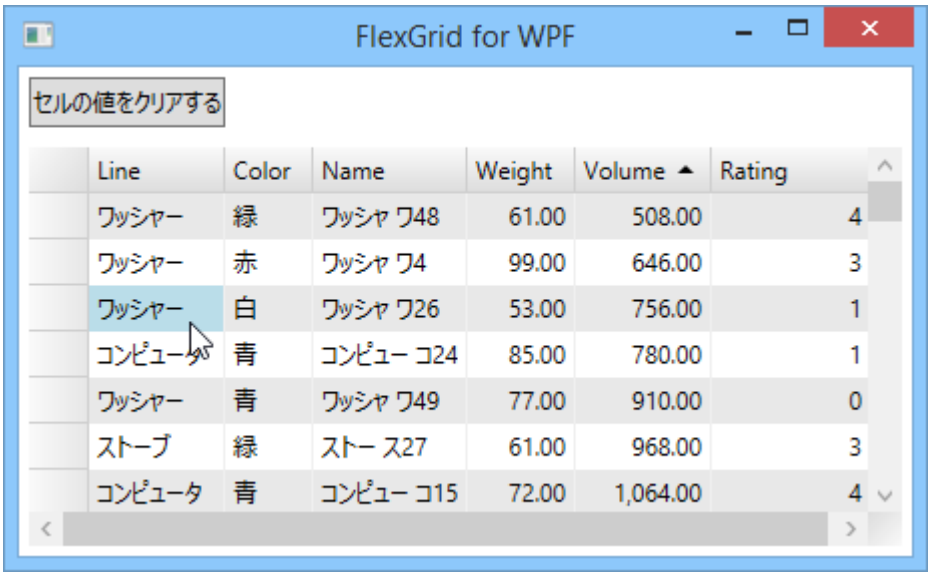
Deleteキーでセルの値を削除する

ユーザーがグリッドにある値を編集できる動作は **IsReadOnly** プロパティで制御されています。**IsReadOnly** プロパティはデフォルトでFalseに設定されていますので、カレントセルに DBNull.Value を設定してDeleteキーでセルの値を作成できます。

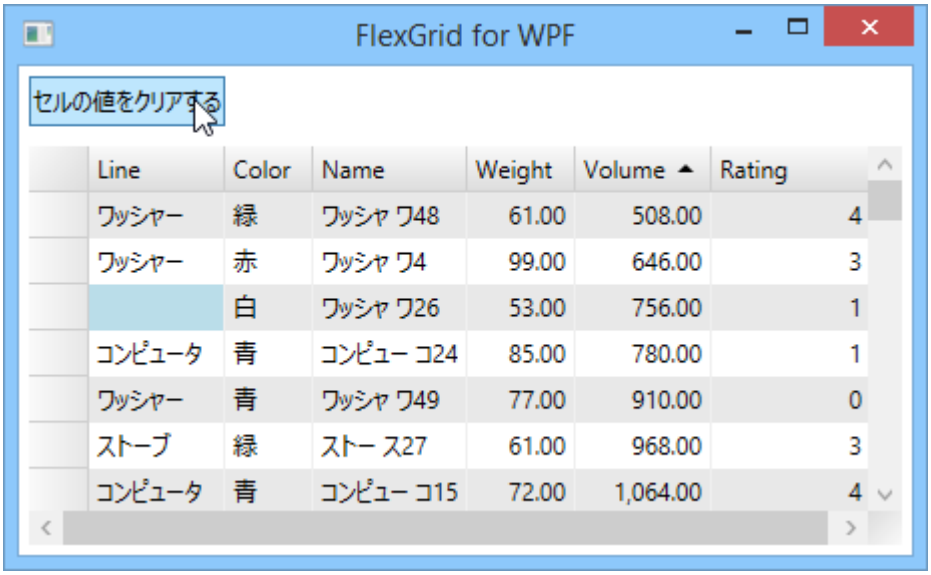
> **注意**:WinForms の AllowEditing プロパティは WPF の他のほとんどのコントロールとの整合性のために、「**IsReadOnly**」に名前が変更されています。

次の例では、グリッドはデータソースと連結されていることを前提として、ボタンクリックおよび Deleteキーでセルの値を作成する方法を示します。

【実行例】



クリア後イメージ



Visual Basic

```
Public Sub New()
    InitializeComponent()
    C1FlexGrid.ItemsSource = Product.GetProducts(50)
    AddHandler C1FlexGrid.KeyDown, AddressOf C1FlexGrid_KeyDown
    AddHandler delBtn.Click, AddressOf delBtn_Click
End Sub
'Deleteキーを押してセルのデータを削除することができます
Private Sub C1FlexGrid_KeyDown(sender As Object, e As KeyEventArgs)
    If e.Key = Key.Delete Then
        DeleteItem()
    End If
End Sub
Private Sub delBtn_Click(sender As Object, e As RoutedEventArgs)
    DeleteItem()
End Sub
Private Sub DeleteItem()
    If C1FlexGrid.SelectedItem IsNot Nothing Then
        C1FlexGrid(C1FlexGrid.Selection.Row, C1FlexGrid.Selection.Column) =
System.DBNull.Value
    End If
End Sub
```

C#

```
public DeleteCellValue()
{
    InitializeComponent();
    C1FlexGrid.ItemsSource = Product.GetProducts(50);
    C1FlexGrid.KeyDown += C1FlexGrid_KeyDown;
    delBtn.Click += delBtn_Click;
}
//Deleteキーを押してセルのデータを削除することができます
private void C1FlexGrid_KeyDown(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Delete)
        DeleteItem();
}

private void delBtn_Click(object sender, RoutedEventArgs e)
{
    DeleteItem();
}

private void DeleteItem()
{
    if (C1FlexGrid.SelectedItem != null)
    {
        C1FlexGrid[C1FlexGrid.Selection.Row, C1FlexGrid.Selection.Column] =
System.DBNull.Value;
    }
}
```

チェックボックスを表示する

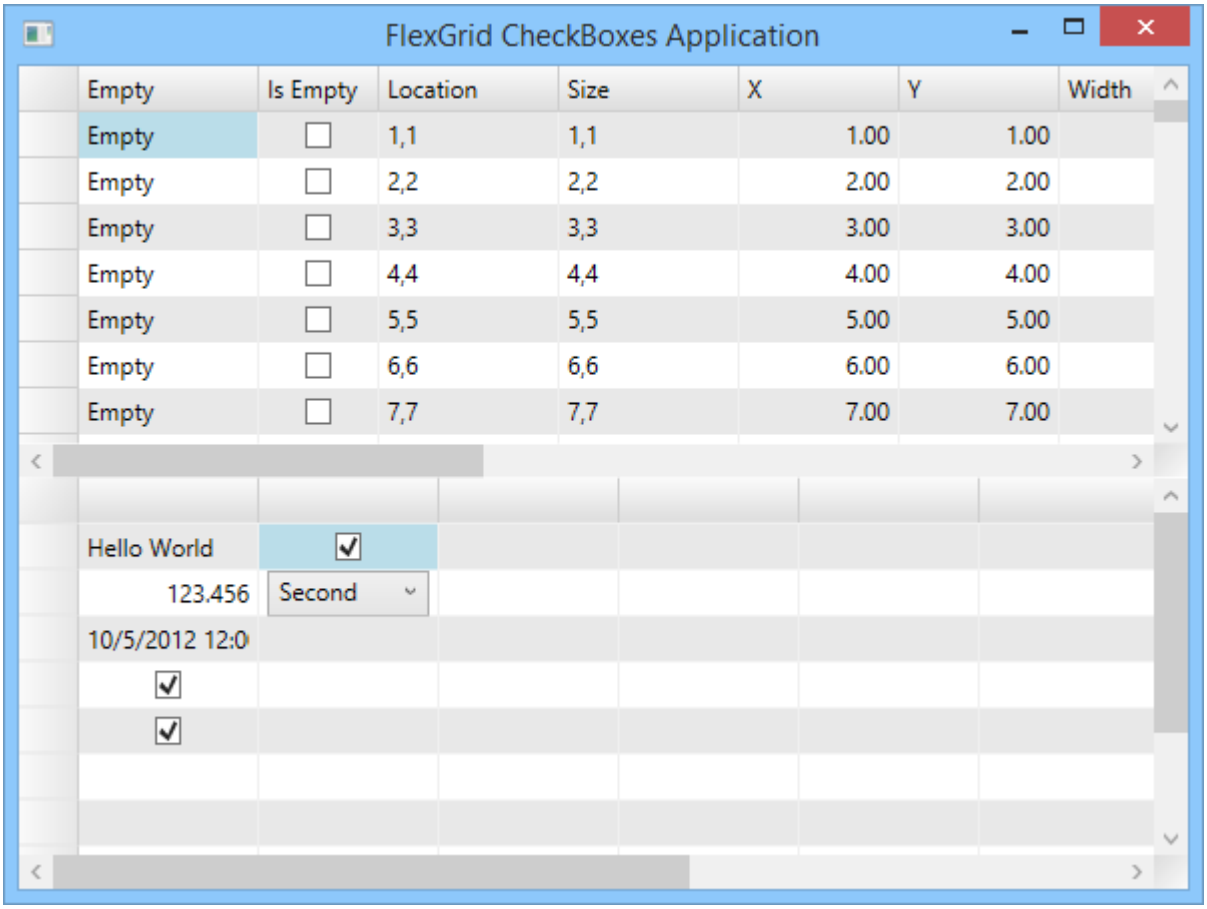
グリッドを連結モードで使用している場合は、行や列のデータ型（DataType）がブール型であれば、自動的にチェックボックスが表示されます。しかし、非連結モードの場合は、グリッドはセルの値を直接セルに格納し、Object型のデータを文字列に変換しています。それで、ブール型のデータを Excel のように True/False として解釈して表示しています。

この場合、**C1FlexGrid** でカスタムセルファクトリを使用してコントロールのデフォルト動作をオーバーライドする方法が用意されています。
この強力な機能は、WinForms版の SetCellChecked メソッドを使用する方法に比べて少し複雑ですが、より柔軟な方法です。たとえば、ブール型データを自動的にチェックボックスとして表示するようなカスタムセルファクトリを作成するか、チャックボックスを直接セルに格納してカスタムセルファクトリがそのセル内のコントロールを簡単に表示するような方法は実現可能になります。

次のコードでは、下記のシナリオを実現するカスタムセルファクトリを使用してセルにチェックボックスを表示します：

1. セルにブール型のデータがある場合、ファクトリがそのセルに対してチェックボックスを作成します
2. セルにコントロールオブジェクトがある場合、ファクトリがそのセルにコントロールを表示します

【実行例】



Visual Basic

```
Public Class MyCellFactory
    Inherits CellFactory
    Public Overrides Sub CreateCellContent(grid As C1FlexGrid, bdr As
System.Windows.Controls.Border, rng As CellRange)
        If TypeOf grid.Cells(rng.Row, rng.Column) Is Boolean Then
            ' 値を表示/編集するために、チェックボックスを作成します
            Dim chk As New CheckBox()
            chk.IsChecked = DirectCast(grid.Cells(rng.Row, rng.Column),
System.Nullable(Of Boolean))
            chk.VerticalAlignment = VerticalAlignment.Center
            chk.HorizontalAlignment = HorizontalAlignment.Center
            ToolTipService.SetToolTip(chk, "このチェックボックスはグリッドセルに格納するブール値
を表します")
            ' チェックボックスをセル要素に割り当てます
            bdr.Child = chk
            ' グリッドセルの内容が更新されるように、チェックボックスを接続します
            chk.Tag = grid

            chk.[AddHandler](CheckBox.ClickEvent, New RoutedEventHandler(AddressOf
chk_Click))
        ElseIf TypeOf grid.Cells(rng.Row, rng.Column) Is Control Then
            ' コントロールをセルに追加します(親がない場合)
            Dim ctl As Control = DirectCast(grid.Cells(rng.Row, rng.Column),
Control)
            If ctl.Parent Is Nothing Then
                bdr.Child = DirectCast(grid.Cells(rng.Row, rng.Column),
System.Windows.Controls.Control)

            End If
        Else
            ' ブール値ではない場合、デフォルト動作を実現
            MyBase.CreateCellContent(grid, bdr, rng)
        End If
    End Sub
    ' チェックボックスがクリックされた場合、値を更新します
    Private Sub chk_Click(sender As Object, e As EventArgs)
        ' クリックされたチェックボックスを取得します
        Dim chk As CheckBox = DirectCast(sender, CheckBox)
        ' チェックボックスを保持するグリッドを取得します
        Dim flex As C1FlexGrid = DirectCast(chk.Tag, C1FlexGrid)
        ' チェックボックスを持つセルを取得します
        Dim bdr As Border = DirectCast(chk.Parent, Border)
        'int row = bdr.GetValue(Grid.RowProperty.GlobalIndex);
        Dim row As Integer = CInt(bdr.GetValue(Grid.RowProperty))
        Dim col As Integer = CInt(bdr.GetValue(Grid.ColumnProperty))
        ' セルに新しい値を割り当てます
        flex(row, col) = chk.IsChecked
    End Sub
    ' セルが廃却された場合、セルの境界のコンテンツも削除します
```

```
    Public Overrides Sub DisposeCell(grid As C1.WPF.FlexGrid.C1FlexGrid, cellType
As C1.WPF.FlexGrid.CellType, cell As System.Windows.FrameworkElement)
        Dim bdr As Border = DirectCast(cell, Border)
        bdr.Child = Nothing
    End Sub
End Class
```

C#

```
public class MyCellFactory : CellFactory
{
    public override void CreateCellContent(C1FlexGrid grid,
System.Windows.Controls.Border bdr, CellRange rng)
    {

        if (grid.Cells[rng.Row, rng.Column] is bool)
        {

            // 値を表示/編集するために、チェックボックスを作成します
            CheckBox chk = new CheckBox();
            chk.IsChecked = (bool?)grid.Cells[rng.Row, rng.Column];
            chk.VerticalAlignment = VerticalAlignment.Center;
            chk.HorizontalAlignment = HorizontalAlignment.Center;
            ToolTipService.SetToolTip(chk, "このチェックボックスはグリッドセルに格納するブール値
を表します");

            // チェックボックスをセル要素に割り当てます
            bdr.Child = chk;

            // グリッドセルの内容が更新されるように、チェックボックスを接続します
            chk.Tag = grid;
            chk.AddHandler(CheckBox.ClickEvent, new RoutedEventHandler(chk_Click));


        }
        else if (grid.Cells[rng.Row, rng.Column] is Control)
        {
            // コントロールをセルに追加します(親がない場合)
            Control ctl = (Control)grid.Cells[rng.Row, rng.Column];
            if (ctl.Parent == null)
            {
                  bdr.Child =(System.Windows.Controls.Control) grid.Cells[rng.Row,
rng.Column];
            }


        }
        else
        {
            // ブール値ではない場合、デフォルト動作を実現
            base.CreateCellContent(grid, bdr, rng);

        }

    }

    // チェックボックスがクリックされた場合、値を更新します
```

```
    private void chk_Click(object sender, EventArgs e)
    {
        // クリックされたチェックボックスを取得します
        CheckBox chk = (CheckBox)sender;

        // チェックボックスを保持するグリッドを取得します
        C1FlexGrid flex = (C1FlexGrid)chk.Tag;

        // チェックボックスを持つセルを取得します
        Border bdr = (Border)chk.Parent;

        //int row = bdr.GetValue(Grid.RowProperty.GlobalIndex);
        int row =(int) bdr.GetValue(Grid.RowProperty);
        int col = (int)bdr.GetValue(Grid.ColumnProperty);

        //// セルに新しい値を割り当てます
        flex[row, col] = chk.IsChecked;

    }
    // セルが廃却された場合、セルの境界のコンテンツも削除します
    public override void DisposeCell(C1.WPF.FlexGrid.C1FlexGrid grid,
C1.WPF.FlexGrid.CellType cellType, System.Windows.FrameworkElement cell)
    {
        Border bdr = (Border)cell;
        bdr.Child = null;
    }
}
```

次は、セルにコントロールを追加する方法を示します。

Visual Basic

```
' コントロールを追加します
Dim chk As New CheckBox()
ToolTipService.SetToolTip(chk, "このチェックボックスはグリッドセルに格納しています")
chk.VerticalAlignment = VerticalAlignment.Center
chk.HorizontalAlignment = HorizontalAlignment.Center
_fgUnbound(0, 1) = chk
Dim cmb As New ComboBox()
cmb.Items.Add("First")
cmb.Items.Add("Second")
cmb.Items.Add("Third")
ToolTipService.SetToolTip(chk, "このコンボボックスはグリッドセルに格納しています")
_fgUnbound(1, 1) = cmb
' カスタムセルファクトリに結びます
' （ブール値をチェックボックスとして表示する）
_fgUnbound.CellFactory = New MyCellFactory()
```

C#

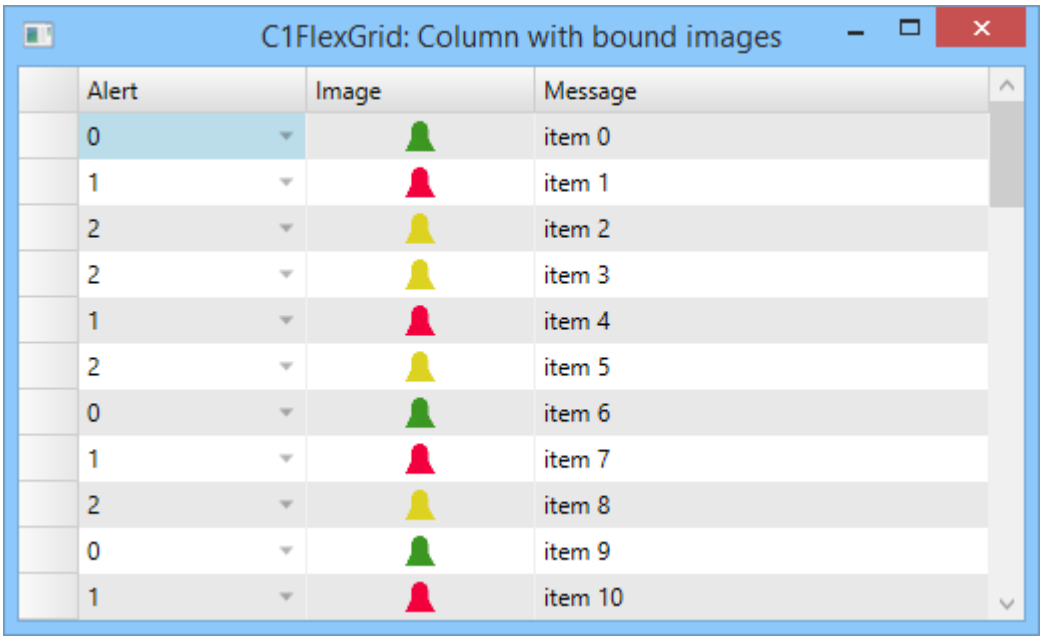
```
// コントロールを追加します
CheckBox chk = new CheckBox();
ToolTipService.SetToolTip(chk, "このチェックボックスはグリッドセルに格納しています");
chk.VerticalAlignment = VerticalAlignment.Center;
chk.HorizontalAlignment = HorizontalAlignment.Center;
_fgUnbound[0, 1] = chk;

ComboBox cmb = new ComboBox();
cmb.Items.Add("First");
cmb.Items.Add("Second");
cmb.Items.Add("Third");
ToolTipService.SetToolTip(chk, "このコンボボックスはグリッドセルに格納しています");
_fgUnbound[1, 1] = cmb;

// カスタムセルファクトリに結びます
// （ブール値をチェックボックスとして表示する）
_fgUnbound.CellFactory = new MyCellFactory();
```

セルに画像を表示する

連結グリッドでセルに画像を表示するには、列の**CellTemplate** プロパティを使用して画像要素を含むデータテンプレートに連結する方法があります。この方法では、数値プロパティにバインドされている数値を自動的に画像に変える**ValueConverter**を使用します。

たとえば、次の例をご覧ください。

【実行例】



マークアップ

```
<c1:C1FlexGrid Name="_flex" AutoGenerateColumns="False" >
```

```
    <c1:C1FlexGrid.Columns>
        <c1:Column Header="Alert" Binding="{Binding Alert}" Width="*" />
        <c1:Column Header="Image" Width="*" >
            <c1:Column.CellTemplate>
                <DataTemplate>
                    <Image
                        Margin="4"
                        Source="{Binding Path=Alert, Converter={StaticResource
AlertImageConverter}}"/>
                </DataTemplate>
            </c1:Column.CellTemplate>
        </c1:Column>
        <c1:Column Header="Message" Binding="{Binding Message}" Width="2*" />
    </c1:C1FlexGrid.Columns>
</c1:C1FlexGrid>
```

バインディングのコンバータープロパティは「**AlertImageConverter**」という静的リソースに設定されています。このパブリッククラスをメインアプリケーションに定義して、アプリケーションリソースに次のように追加されます。

```
<Window.Resources>
  <local:AlertImageConverter x:Key="AlertImageConverter" />
</Window.Resources>
```

最終的に、**AlertImageConverter** クラスを次のように実装します。

Visual Basic

```
Public Class AlertImageConverter
    Implements IValueConverter

    ' アプリケーションのリソースから静的イメージをロードします
    Private Shared _bmpRed As BitmapImage
    Private Shared _bmpYellow As BitmapImage
    Private Shared _bmpGreen As BitmapImage

    Shared Sub New()
        _bmpRed = New BitmapImage(New Uri("/Resources/redBell.png",
UriKind.Relative))
        _bmpYellow = New BitmapImage(New Uri("/Resources/yellowBell.png",
UriKind.Relative))
        _bmpGreen = New BitmapImage(New Uri("/Resources/greenBell.png",
UriKind.Relative))
    End Sub

    ' 「Alert」int型を当該するイメージに変換します
    Public Function Convert(ByVal value As Object, ByVal targetType As Type, ByVal
parameter As Object,
        ByVal culture As System.Globalization.CultureInfo) As Object Implements
IValueConverter.Convert
        Select Case (CType(value, Integer))
            Case 1
                Return _bmpRed
            Case 2
                Return _bmpYellow
        End Select

        Return _bmpGreen
    End Function

    ' 一方変換のみ(CovertBackは使用されません)
    Public Function ConvertBack(ByVal value As Object, ByVal targetType As Type,
ByVal parameter As Object,
        ByVal culture As System.Globalization.CultureInfo) As Object Implements
IValueConverter.ConvertBack
        Throw New NotImplementedException
    End Function
End Class
```

C#

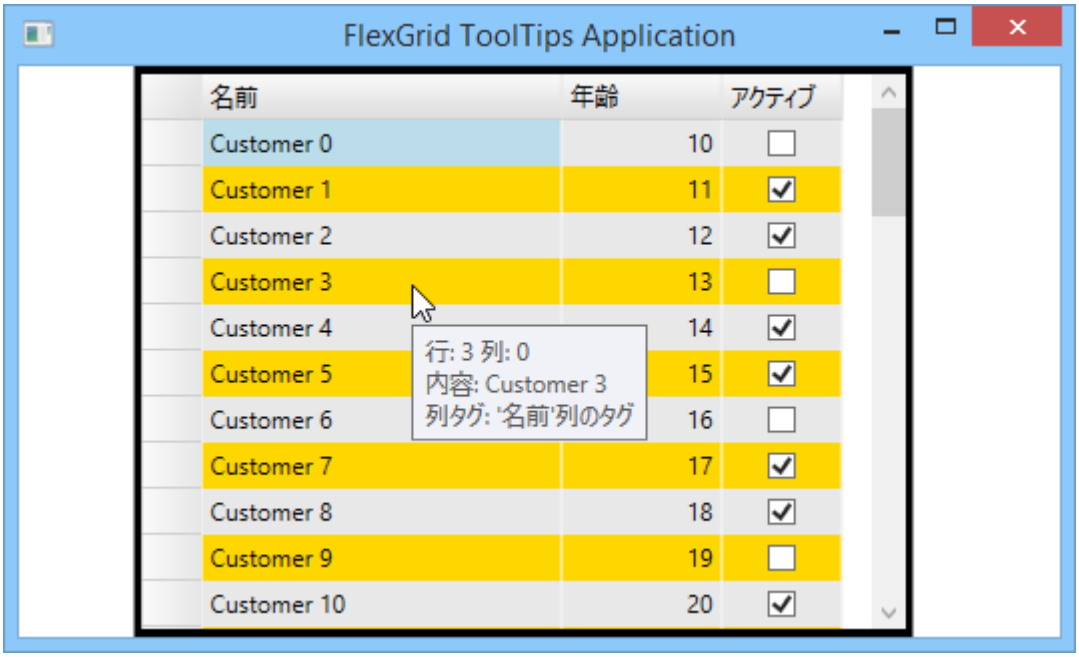
```
public class AlertImageConverter : IValueConverter
{

    // アプリケーションのリソースから静的イメージをロードします
    static BitmapImage _bmpRed, _bmpYellow, _bmpGreen;
    static AlertImageConverter()
    {
        _bmpRed = new BitmapImage(new Uri("/Resources/redBell.png",
UriKind.Relative));
        _bmpYellow = new BitmapImage(new Uri("/Resources/yellowBell.png",
UriKind.Relative));
        _bmpGreen = new BitmapImage(new Uri("/Resources/greenBell.png",
UriKind.Relative));
    }

    // 「Alert」int型を当該するイメージに変換します
    public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
    {
        switch ((int)value)
        {
            case 1: return _bmpRed;
            case 2: return _bmpYellow;
        }
        return _bmpGreen;
    }

    // 一方変換のみ(CovertBackは使用されません)
    public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }

}
```

セルのユーザーデータをツールチップに表示

ユーザーがセルデータをツールチップに表示する場合は、WPF版の標準の **ToolipService** を使用してグリッドで使用できるツールチップを作成できます。そして、カスタムセルファクトリを使用することで、ツールチップをセルに付けて表示できます。

たとえば、次の例をご覧ください。

【実行例】



Visual Basic

Visual Basic

```
Partial Public Class MainWindow
Inherits Window
Public Sub New()
    InitializeComponent()
    _grid.CellFactory = New ToolTipCellFactory()
    Dim list As New List(Of Customer)()
    For i As Integer = 0 To 49
        Dim cus     As Customer = New Customer
        cus.Name = "Customer " + i.ToString()
        cus.Age = 10 + i
        cus.Active = i Mod 3 <> 0
        list.Add(cus)
    Next i
    _grid.ItemsSource = list
    For Each col As Column In _grid.Columns
        col.Tag = String.Format("tag For column'{0}'", col.ColumnName)
    Next
End Sub
' イベントハンドラー
Private Sub _grid_PrepareCellForEdit(sender As Object, e As CellEditEventArgs)
Handles _grid.PrepareCellForEdit
    ' 選択の外観を変更してエディタをカスタマイズします
    Dim b As Border = DirectCast(e.Editor, Border)
    Dim tb As TextBox = DirectCast(b.Child, TextBox)
    If tb IsNot Nothing Then
        tb.Background = New SolidColorBrush(Colors.Black)
        tb.Foreground = New SolidColorBrush(Colors.White)
    End If
End Sub
End Class
```

C#

C#

```
public partial class MainWindow : Window
{
    public MainWindow()

    {
        InitializeComponent();
        _grid.CellFactory = new ToolTipCellFactory();
        var list = new List<Customer>();
        for (int i = 0; i < 50; i++)
        {
            list.Add(new Customer()
            {
                名前 = "Customer " + i.ToString(),
                年齢 = 10 + i,
                アクティブ = i % 3 != 0
            });
        }
        _grid.ItemsSource = list;

        foreach (Column col in _grid.Columns)
        {
            col.Tag = string.Format("'{0}'列のタグ", col.ColumnName);
        }

        // イベントハンドラー
        _grid.PrepareCellForEdit += _grid_PrepareCellForEdit;
    }

    // 選択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;
        if (tb != null)
        {
            tb.Background = new SolidColorBrush(Colors.Black);
            tb.Foreground = new SolidColorBrush(Colors.White);
        }
    }

}
```

セルにツールチップを追加するカスタムセルファクトリは、次のようになります。

Visual Basic

Visual Basic

```
Public Class ToolTipCellFactory
    Inherits CellFactory
Public Overrides Sub CreateCellContent(ByVal grid As C1FlexGrid, ByVal bdr As
Border, ByVal rng As CellRange)
        MyBase.CreateCellContent(grid, bdr, rng)
        Dim tip = String.Format("行: {0} 列: {1}" & vbCrLf & "内容: {2}" & vbCrLf &
"列タグ: {3}", rng.Row,
            rng.Column, grid(rng.Row, rng.Column), grid.Columns(rng.Column).Tag)
        ToolTipService.SetToolTip(bdr, tip)
    End Sub
End Class
```

C#

C#

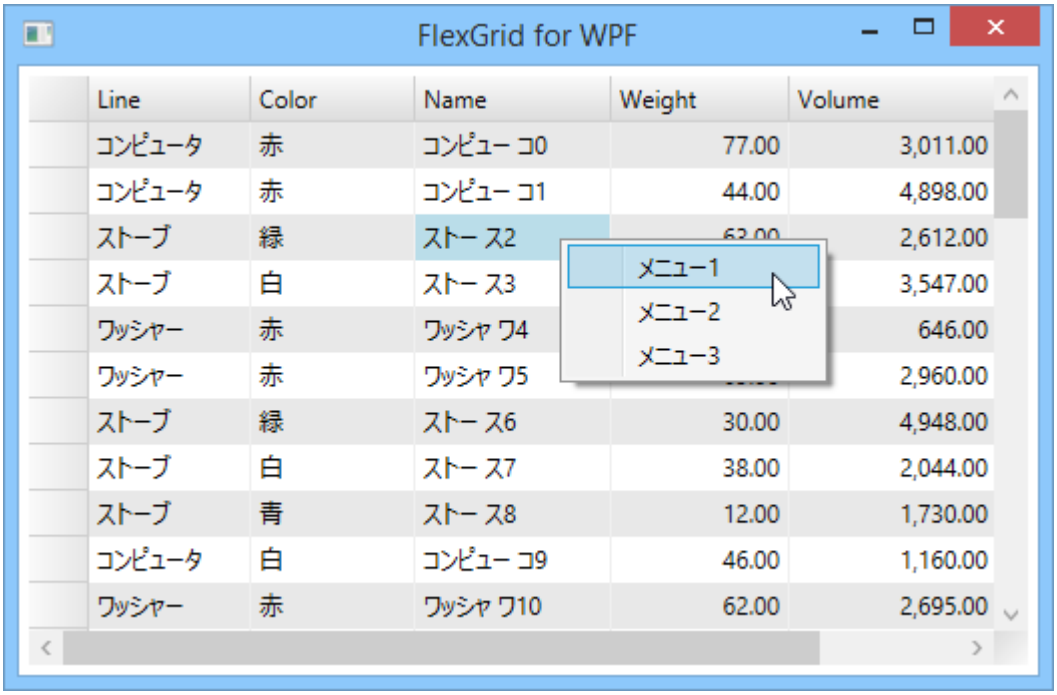
```
public class ToolTipCellFactory : CellFactory
{
    public override void CreateCellContent(C1FlexGrid grid, Border bdr, CellRange
rng)
    {

        // コンテンツを作成します
        base.CreateCellContent(grid, bdr, rng);
        // ツールチップを追加します
        var tip = string.Format("行: {0} 列: {1}\r\n内容: {2}\r\n列タグ: {3}",
            rng.Row,
            rng.Column,
            grid[rng.Row, rng.Column],
            grid.Columns[rng.Column].Tag);
        ToolTipService.SetToolTip(bdr, tip);

    }
}
```

コンテキストメニューを表示する

非編集時の場合

C1FlexGrid にはFrameworkElement から継承される **ContextMenu** プロパティが用意されています。任意のコンテキストメニューを作成して **C1FlexGrid** の **ContextMenu** プロパティに設定します。
たとえば、次の例では編集時に表示するコンテキストメニューと非編集時のコンテキストメニューを設定する方法を示します。

注意:WPF版ではFrameworkElement から継承されるContextMenuプロパティが用意されていますので、WinFormsとは異なり、ContextMenuStrip より ContextMenu プロパティの使用をお勧めします。



Visual Basic

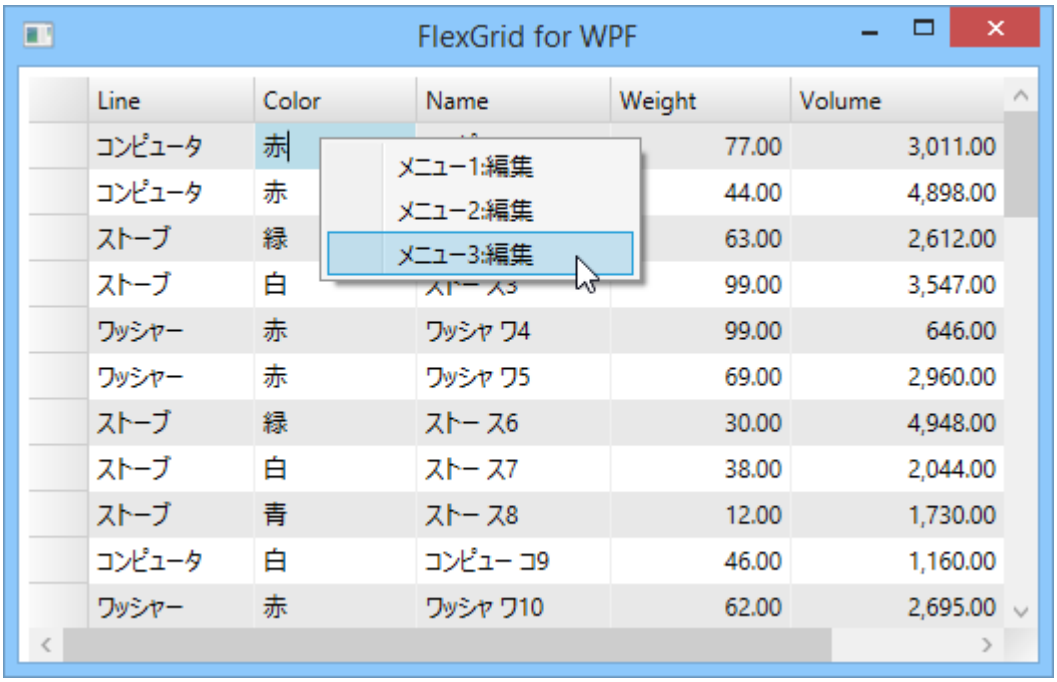
```
Public Sub New()
    InitializeComponent()
    Dim p = Product.GetProducts(50)
    _flex.ItemsSource = p
    ' コンテキストメニューを作成します
    Dim cm As New ContextMenu()
    cm.Items.Add("メニュー1")
    cm.Items.Add("メニュー2")
    cm.Items.Add("メニュー3")
    _flex.ContextMenu = cm
    AddHandler _flex.PrepareCellForEdit, AddressOf _flex_PrepareCellForEdit
End Sub
```

C#

```
public MainWindow()
{
    InitializeComponent();
    var p = Product.GetProducts(50);
    _flex.ItemsSource = p;
    // コンテキストメニューを作成します
    ContextMenu cm = new ContextMenu();
    cm.Items.Add("メニュー1");
    cm.Items.Add("メニュー2");
    cm.Items.Add("メニュー3");
    _flex.ContextMenu = cm;
    _flex.PrepareCellForEdit += _flex_PrepareCellForEdit;
}
```

編集中の場合

編集中は編集用エディタに設定されているデフォルトのコンテキストメニューが表示されます。デフォルトのコンテキストメニューを変更する場合は、**PrepareCellForEdit** イベントで編集用エディタを取得し、別の **ContextMenu** コントロールを設定します。



Visual Basic

```
    Private Sub _flex_PrepareCellForEdit(sender As Object, e As
C1.WPF.FlexGrid.CellEditEventArgs)
        ' 1列目のみのコンテキストメニューを変更します
        If e.Column = 1 Then
            ' Edit エディタを取得します
            Dim border As Border = DirectCast(_flex.ActiveEditor, Border)
            Dim control = TryCast(border.Child, Control)
            ' コンテキストメニューを作成します
            Dim cm2 As New ContextMenu()
            cm2.Items.Add("メニュー1:編集")
            cm2.Items.Add("メニュー2:編集")
            cm2.Items.Add("メニュー3:編集")
            ' コンテキストメニューを設定します
            control.ContextMenu = cm2
        Else
            ' コンテキストメニューを削除します(デフォルトに復元します)
            _flex.ActiveEditor.ContextMenu = Nothing
        End If
    End Sub
```

C#

```
void _flex_PrepareCellForEdit(object sender, C1.WPF.FlexGrid.CellEditEventArgs e)
{
    // 1列目のみのコンテキストメニューを変更します
    if (e.Column == 1)
    {
        // Edit エディタを取得します
        Border border = (Border)_flex.ActiveEditor;
        var control = border.Child as Control;
        // コンテキストメニューを作成します
        ContextMenu cm2 = new ContextMenu();
        cm2.Items.Add("メニュー1:編集");
        cm2.Items.Add("メニュー2:編集");
        cm2.Items.Add("メニュー3:編集");
        // コンテキストメニューを設定します
        control.ContextMenu = cm2;
    }
    else
    {
        // コンテキストメニューを削除します(デフォルトに復元します)
        _flex.ActiveEditor.ContextMenu = null;
    }
}
```

複数セルの行および列ヘッダ

ほとんどのグリッドは、行および列ヘッダ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、複数セルのヘッダをサポートします。たとえば、列ヘッダを 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#

C#

```
// 非連結の列ヘッダを設定します。
    var ch = fg.ColumnHeaders;
       ch.Rows.Add(new Row());
 for (int c = 0; c < ch.Columns.Count; c++)
  {
     // 年 ch[0, c] = 2009 + c / 4;
     // 四半期
      ch[1, c] = string.Format("Q {0}", c % 4 + 1);
  }
```

このコードは、次のようなグリッドを生成します

		2009	2009	2009	2009	2010	2010
		Q 1	Q 2	Q 3	Q 4	Q 1	Q 2
ヘッダ 0,	ヘッダ 0,	◢ レベル 0					
ヘッダ 0,	ヘッダ 1,	◢ レベル 1					
ヘッダ 1,	ヘッダ 2,	◢ レベル 2					
ヘッダ 1,	ヘッダ 3,	セル [0,0]	セル [0,1]	セル [0,2]	セル [0,3]	セル [0,4]	セル
ヘッダ 2,	ヘッダ 4,	セル [1,0]	セル [1,1]	セル [1,2]	セル [1,3]	セル [1,4]	セル
	ヘッダ 5,	セル [2,0]	セル [2,1]	セル [2,2]	セル [2,3]	セル [2,4]	セル
ヘッダ 3,	ヘッダ 6,	セル [3,0]	セル [3,1]	セル [3,2]	セル [3,3]	セル [3,4]	セル
	ヘッダ 7,	セル [4,0]	セル [4,1]	セル [4,2]	セル [4,3]	セル [4,4]	セル

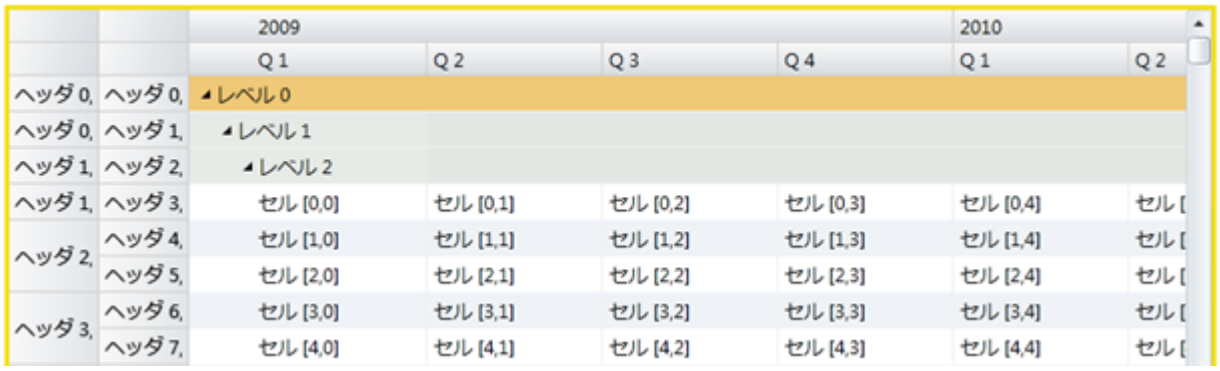
列ヘッダを表示するために 2 つの行が使用されていることに注目してください。従来のグリッドでも、改行を含む列ヘッダを使って同様の効果を得られますが、上端の固定行に結合するようにセルを追加すると、同じ年度を参照する列が自動的に結合されるため、その違いは明らかです。このためには、コードを 2 行追加するだけです。

C#

C#

```
// 非連結の列ヘッダを設定します。
  var ch = fg.ColumnHeaders;

// 年間と四半期を表す別々のヘッダ行
  ch.Rows.Add(new Row());
  for (int c = 0; c < ch.Columns.Count; c++)

  {

    // 年
    ch[0, c] = 2009 + c / 4;

    // 四半期
    ch[1, c] = string.Format("Q {0}", c % 4 + 1);
  }
// 最初の固定行のマージを許可します。
  fg.AllowMerging = AllowMerging.All;
  ch.Rows[0].AllowMerging = true;
```

この結果は、次の図のようになります。



		2009				2010	
		Q 1	Q 2	Q 3	Q 4	Q 1	Q 2
ヘッダ 0,	ヘッダ 0,	◢ レベル 0					
ヘッダ 0,	ヘッダ 1,	◢ レベル 1					
ヘッダ 1,	ヘッダ 2,	◢ レベル 2					
ヘッダ 1,	ヘッダ 3,	セル [0,0]	セル [0,1]	セル [0,2]	セル [0,3]	セル [0,4]	セル [
ヘッダ 2,	ヘッダ 4,	セル [1,0]	セル [1,1]	セル [1,2]	セル [1,3]	セル [1,4]	セル [
	ヘッダ 5,	セル [2,0]	セル [2,1]	セル [2,2]	セル [2,3]	セル [2,4]	セル [
ヘッダ 3,	ヘッダ 6,	セル [3,0]	セル [3,1]	セル [3,2]	セル [3,3]	セル [3,4]	セル [
	ヘッダ 7,	セル [4,0]	セル [4,1]	セル [4,2]	セル [4,3]	セル [4,4]	セル [

上端の固定行にある同じ年度を参照するセルが結合されることで、より見やすくなっていることを確認してください。この図に示したように、行ヘッダを結合する場合にも同じメカニズムを適用できます。

選択

選択範囲と選択モード

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っ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。

選択範囲の取得

In the the following section learn how to implement Monitor Selection in FlexGrid .NET 4.5.2 and .NET 5 versions.

.NET Framework

FlexGrid は、ユーザーの操作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、選択取得を監視し、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#

```
private void grid_SelectionChanged(object sender, CellRangeEventArgs e)
        {
            CellRange sel = grid.Selection;
            Console.WriteLine("選択範囲: {0}, {1} - {2}, {3}",
                sel.Row, sel.Column, sel.Row2, sel.Column2);
            Console.WriteLine("択範囲のコンテンツ: {0}",
            GetClipString(grid, sel));
        }

        static string GetClipString(C1FlexGrid grid, 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", grid[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** メソッドの for ループは、**Row**、**Row2**、**Column**、および **Column2** プロパティではなく、**CellRange** の **TopRow**、**BottomRow**、**LeftColumn**、および **RightColumn** プロパティを使用することに注意してください。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。**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 メソッドの代替実装を考えます。

C#

```
void grid_SelectionChanged(object sender, CellRangeEventArgs e)
{
  // 選択された範囲内の顧客を取得します。
  var customers =
    grid.Rows.GetDataItems(grid.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} 選択された項目, {1} アクティブ, 総体重: {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体重を取得します。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。

.NET

ユーザーの操作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、選択取得を監視し、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#

```
private void grid_SelectionChanged(object sender,
C1.WPF.Grid.GridCellRangeEventArgs e)
{
    //DOC Code Snippet1
    GridCellRange sel = grid.Selection;
    Console.WriteLine("selection: {0}, {1} - {2}, {3}", sel.Row, sel.Column,
sel.Row2, sel.Column2);
    Console.WriteLine("selection content: {0}",
    GetClipString(grid, sel));
}

static string GetClipString(FlexGrid grid, GridCellRange sel)
{
    var sb = new System.Text.StringBuilder();
    for (int r = sel.Row; r <= sel.Row2; r++)
    {
        for (int c = sel.Column; c <= sel.Column2; c++)
        {
            sb.AppendFormat("{0}\t", grid[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 **GridCellRange**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 メソッドの for ループは、**Row**、**Row2**、**Column**、および **Column2** プロパティではなく、GridCellRange の **TopRow**、**BottomRow**、**LeftColumn**、および **RightColumn** プロパティを使用することに注意してください。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。

コードを使用したセルおよびオブジェクトの選択

Selection プロパティは読み書き可能なので、コードを使って簡単にセル範囲を選択できます。また、Windows フォームの C1FlexGrid と同様に、**Select** メソッドを使って選択を実行することもできます。Select メソッドを使用すると、セルまたは範囲を選択できるほか、新しい選択範囲を表示範囲までスクロールしてユーザーに表示することもできます。

たとえば、グリッド内の最初のセルを選択し、そのセルがユーザーに表示されるようにするには、次のようなコードを使用します。

C#

```
// 行 0、列 0 を選択して、このセルが表示されるようにします
grid.Select(0, 0, true);
```

これらの選択方法はすべて、行および列インデックスに基づいて機能します。ただし、セルの内容に基づいて選択を行うこともできます。たとえば、次のコードは、グリッドの「氏名」列に特定の文字列を含む最初の行を選択します。

C#

```
bool SelectName(string name)
{
  // "Name" 列の特定の文字列を含む行を検索します
  int col = _flexGroup.Columns["Name"].Index;
  int row = FindRow(grid, name, grid.Selection.Row, col, true);
  if (row > -1)
  {
    grid.Select(row, col);
    return true;
  }
  // 見つからない場合...
  return false;
}
```

次のように定義された FindRow ヘルパーメソッドを使用します。

C#

```
// 指定した列に特定のテキストを含む行を検索します
int FindRow(C1FlexGrid grid, string text,
            int startRow, int col, bool wrap)
{
  int count = grid.Rows.Count;
  for (int off = 0; off <= count; off++)
  {
    // 下端まで到達し、折り返さない場合は、ここで終了します
    if (!wrap && startRow + off >= count)
    {
      break;
    }

    // 行からテキストを取得します
    int row = (startRow + off) % count;
    var content = grid[row, col];
```

```
    // 見つかった場合は、行インデックスを返します
    if (content != null &&
        content.ToString().IndexOf(text,
        StringComparison.OrdinalIgnoreCase) > -1)
    {
      return row;
    }
  }

  // 見つからない場合...
  return -1;
}
```

FindRow メソッドは、指定された行から開始して指定された列で文字列を検索します。オプションで、文字列が見つからなかった場合には折り返して先頭から検索をやり直します。これは、多くのシナリオで使用される柔軟性の高い方法です。選択を実行するもう 1 つの一般的なシナリオは、データソース内で特定のオブジェクトを選択する場合です。この場合は、**PagedCollectionView.IndexOf** メソッドを使用してソースコレクション内でオブジェクトのインデックスを見つけ、次にそのインデックスを使用して行を選択するという方法が考えられます。しかし、この方法には、データがグループ化されていない場合にしか機能しないという問題があります。データがグループ化されている場合は、グループ行もカウントされるため、データソース内の項目のインデックスはグリッドの行インデックスと一致しません。

この問題を簡単に解決する方法としては、行を列挙し、各行の **DataItem** プロパティと検索対象の項目を比較します。次のコードは、この方法を示しています。

C#

```
var customer = GetSomeCustomer;

#if false // ** これは使用しないでください。グループ化されたデータでは無効です

  int index = view.IndexOf(customer);
  if (index > -1)
  {
    grid.Select(index, 0);
  }

#else // こちらがグリッド内のオブジェクトを検索する安全な方法です

  for (int row = 0; row <= grid.Rows.Count; row++)
  {
    if (row.DataItem == customer)
    {
      grid.Select(row, 0);
      break;
    }
  }
#endif
```

選択範囲の表示のカスタマイズ

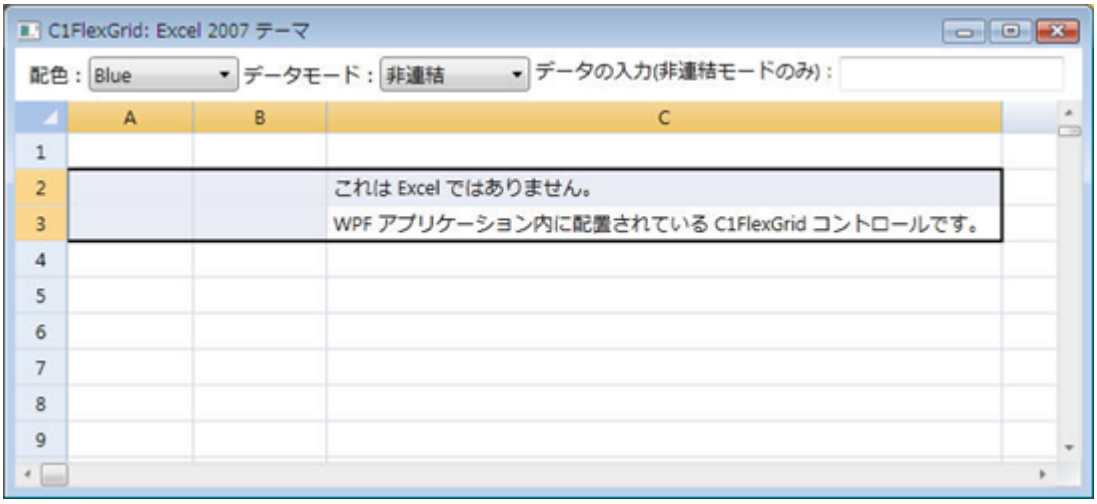
FlexGrid には、選択範囲の強調表示をカスタマイズできる 2 つの機能が含まれています。

- **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

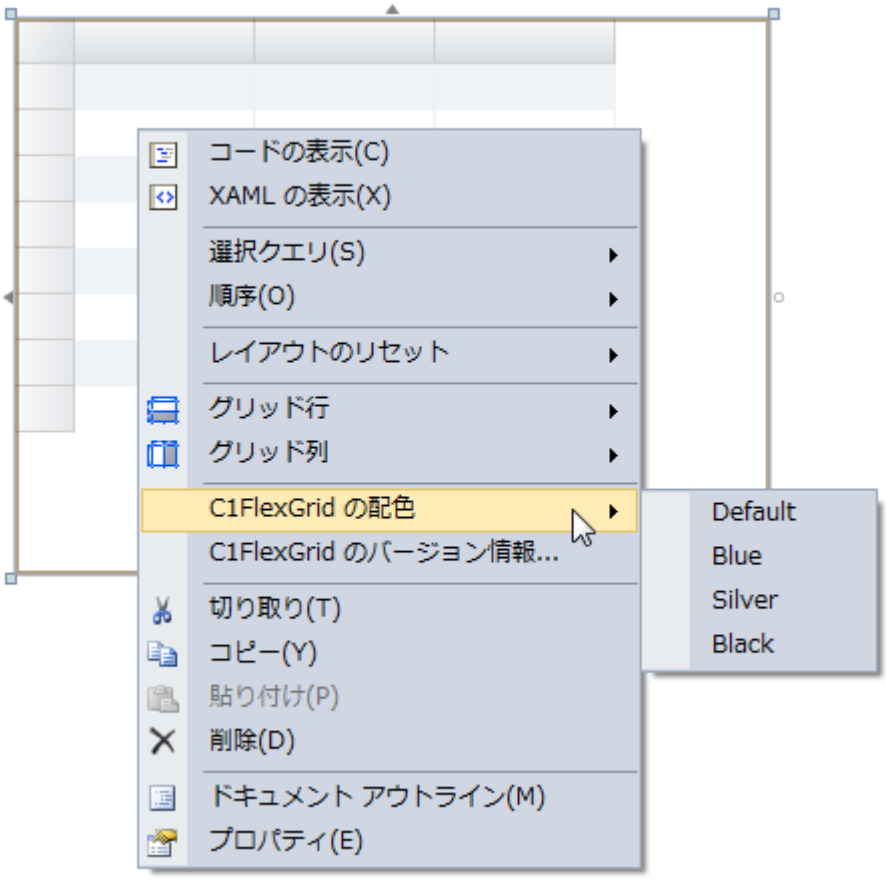
れて、グリッドがとても見やすくなります。デフォルトでは、このマーキーは 2 ピクセルの濃い黒色の四角形ですが、**Marquee** プロパティを使ってカスタマイズすることができます。

- **選択されたセルのヘッダ**：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。



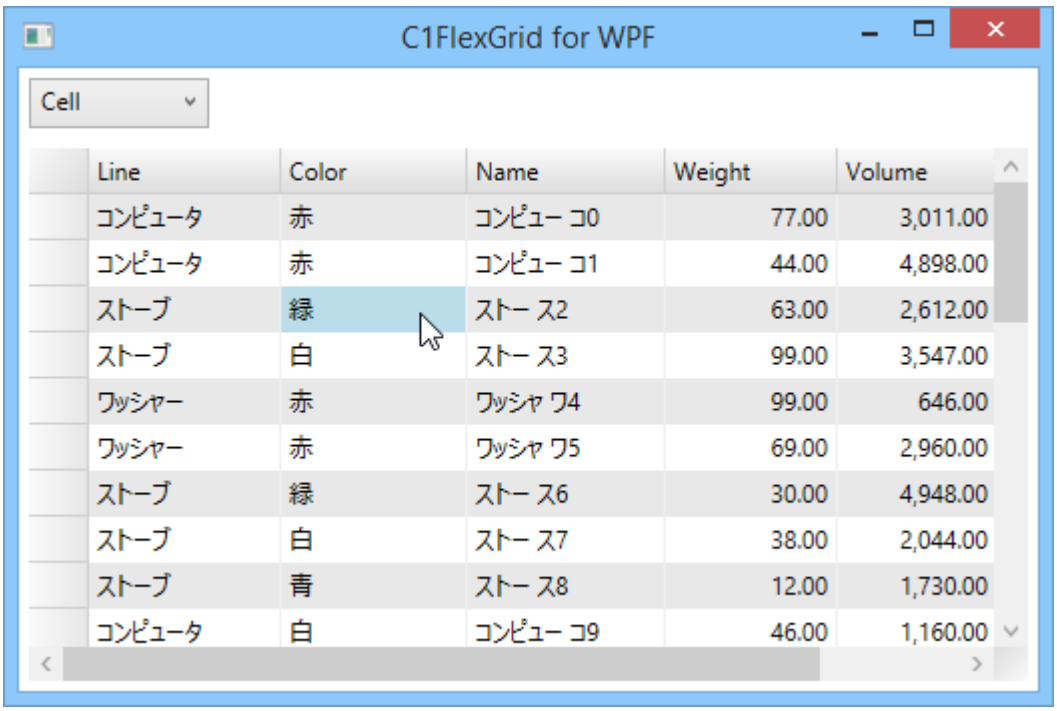
FlexGrid デザイナにはコンテキストメニューがあり、このメニューには Excel 形式の設定済み配色（Blue、Silver、Black）を選択できるオプションがあります。このデザイナによって生成された XAML は、再利用可能なスタイルリソースに簡単にコピーできます。



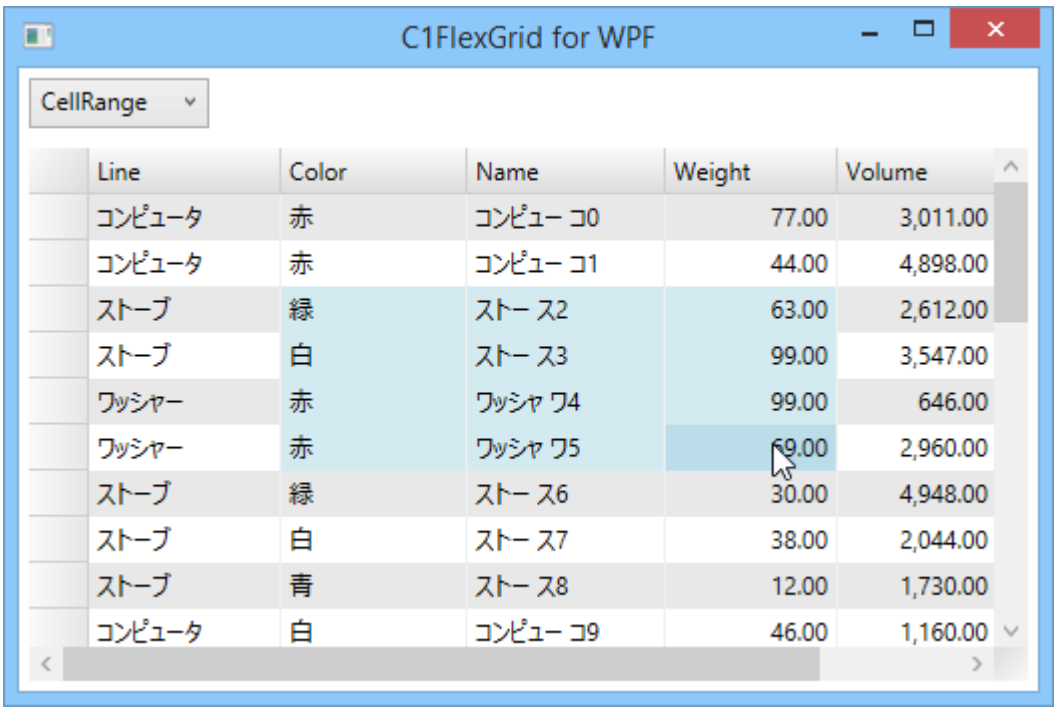
選択モードを設定する

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。デフォルトの **SelectionMode** は **CellRange** ですが、次の 5 つの選択モードを使用できます。

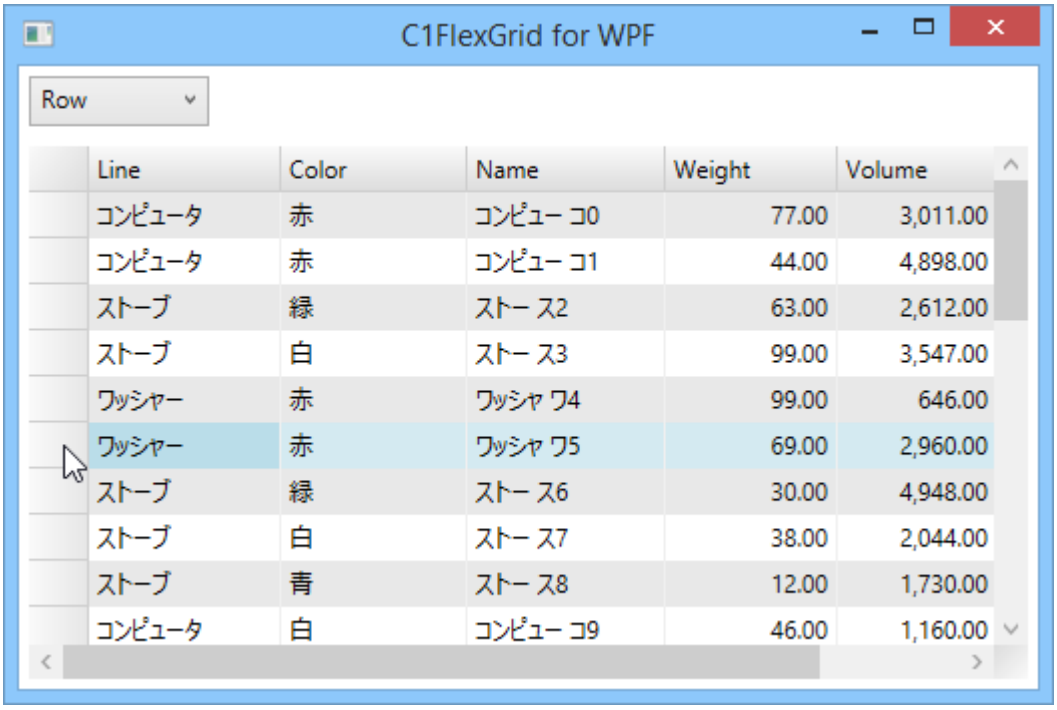
次は、各選択モードの実行例を示します。



[実行例] 単一セル、セル範囲の選択 (CellRange)

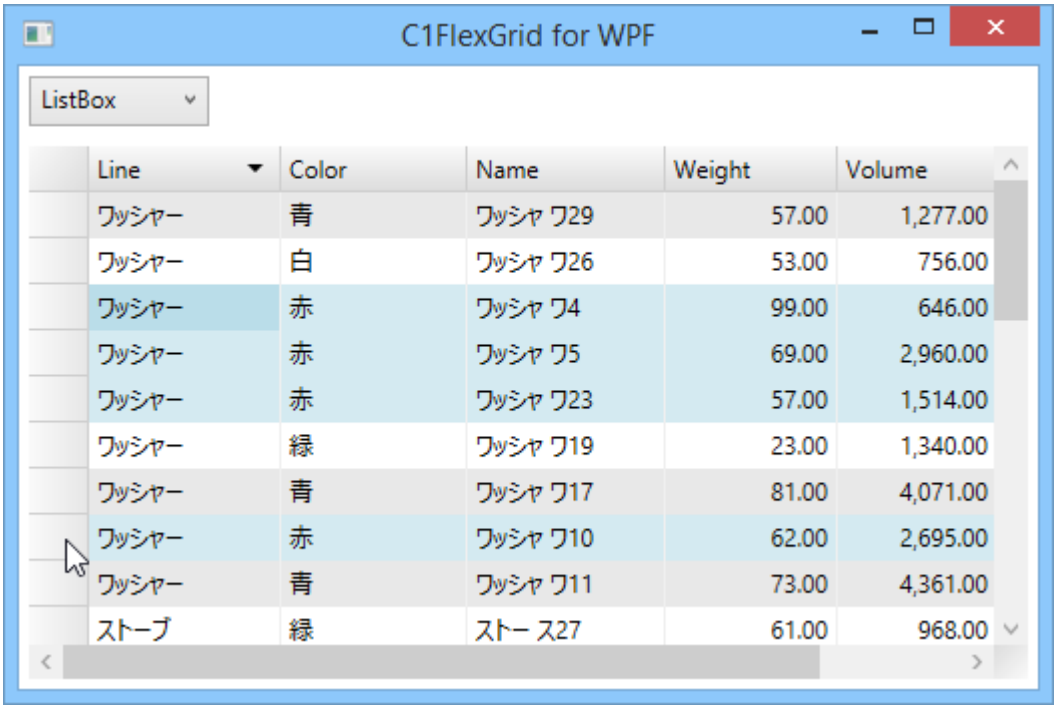


[実行例] 単一行の選択 (Row)



[実行例] 連続する複数行の選択 (RowRange)

[実行例] 行、複数行、連続しない複数行の選択 (ListBox)



VisualBasic

```
Private Sub selectMode_SelectionChanged(sender As Object, e As
SelectionChangedEventArgs)
    Dim mode As C1.WPF.FlexGrid.SelectionMode = C1.WPF.FlexGrid.SelectionMode.Cell
    Select Case selectMode.SelectedIndex
        Case 0
            mode = C1.WPF.FlexGrid.SelectionMode.Cell
        Case 1
            mode = C1.WPF.FlexGrid.SelectionMode.CellRange
        Case 2
            mode = C1.WPF.FlexGrid.SelectionMode.Row
        Case 3
            mode = C1.WPF.FlexGrid.SelectionMode.RowRange
        Case 4
            mode = C1.WPF.FlexGrid.SelectionMode.ListBox
        Case Else
            mode = C1.WPF.FlexGrid.SelectionMode.Cell
    End Select
    Dim origionMode As C1.WPF.FlexGrid.ScaleMode = _flex.SelectionMode
    _flex.SelectionMode = mode
    If origionMode = C1.WPF.FlexGrid.SelectionMode.ListBox Then
        Dim selectedRows As List(Of C1.WPF.FlexGrid.Row) = _flex.Rows.Selected
        selectedRows.ForEach(AddressOf ChangeSelected)
    End If
End Sub

Shared Sub ChangeSelected(ByVal r As C1.WPF.FlexGrid.Row)
    r.Selected = False
End Sub
```

C#

```
private void selectMode_SelectionChanged(object sender, SelectionChangedEventArgs
e)
{
    C1.WPF.FlexGrid.SelectionMode mode = C1.WPF.FlexGrid.SelectionMode.Cell;
    switch (selectMode.SelectedIndex)
    {
        case 0:
            mode = C1.WPF.FlexGrid.SelectionMode.Cell;
            break;
        case 1:
            mode = C1.WPF.FlexGrid.SelectionMode.CellRange;
            break;
        case 2:
            mode = C1.WPF.FlexGrid.SelectionMode.Row;
            break;
        case 3:
            mode = C1.WPF.FlexGrid.SelectionMode.RowRange;
            break;
        case 4:
            mode = C1.WPF.FlexGrid.SelectionMode.ListBox;
            break;
        default:
            mode = C1.WPF.FlexGrid.SelectionMode.Cell;
            break;

    }
    var origionMode = _flex.SelectionMode;
    _flex.SelectionMode = mode;
    if (origionMode == C1.WPF.FlexGrid.SelectionMode.ListBox)
    {
        var selectedRows = _flex.Rows.Selected;
        selectedRows.ForEach(r => r.Selected = false);
    }
}
```

セルを選択する

C1FlexGrid の**Select** メソッドを使用してセルを選択できます。**Select** メソッドを使用すると、セルまたは範囲を選択できるほか、新しい選択範囲を表示範囲までスクロールしてユーザーに表示することもできます。

たとえば、グリッド内のセルを選択し、そのセルがユーザーに表示されるようにするには、次のようなコードを使用します。

VisualBasic

```
' 行3、列2を選択して、このセルが表示されるようにします
_flex.Select(2, 1, True)
```

C#

```
// 行3、列2を選択して、このセルが表示されるようにします
_flex.Select(2, 1, true);
```

Enterキーで次のセルを選択する

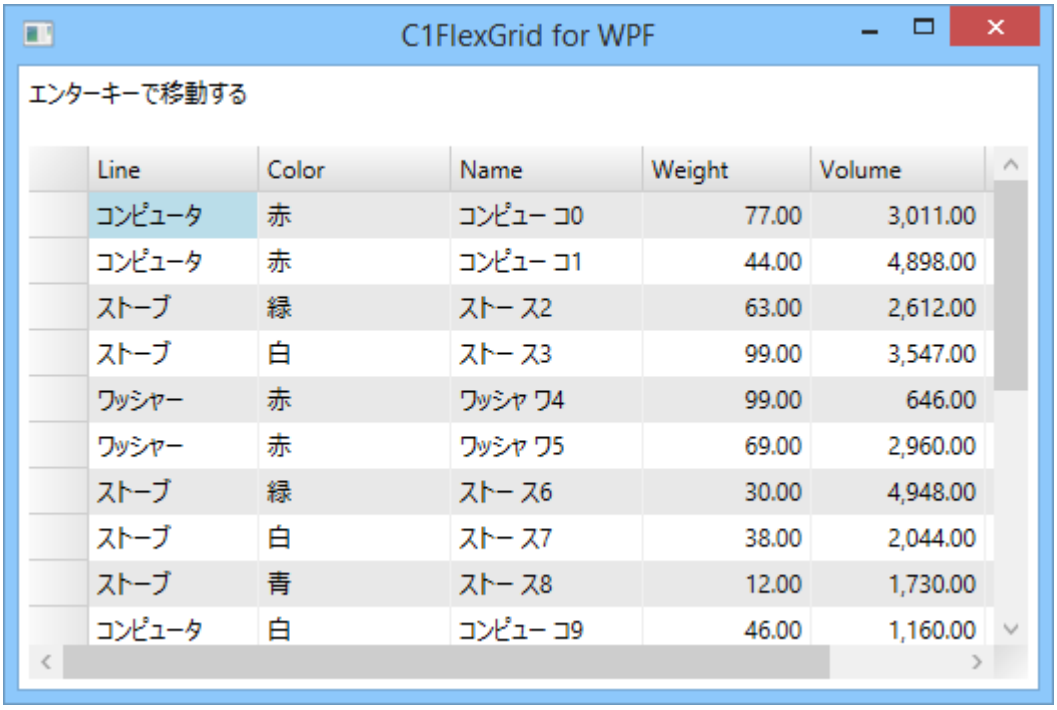
Enter キーでセルを移動するには、**KeyActionEnter** プロパティを変更します。(デフォルト値は **MoveDown**)

たとえば、次のコードでは、**MoveAcross** に設定してカーソルを右に1列移動します。

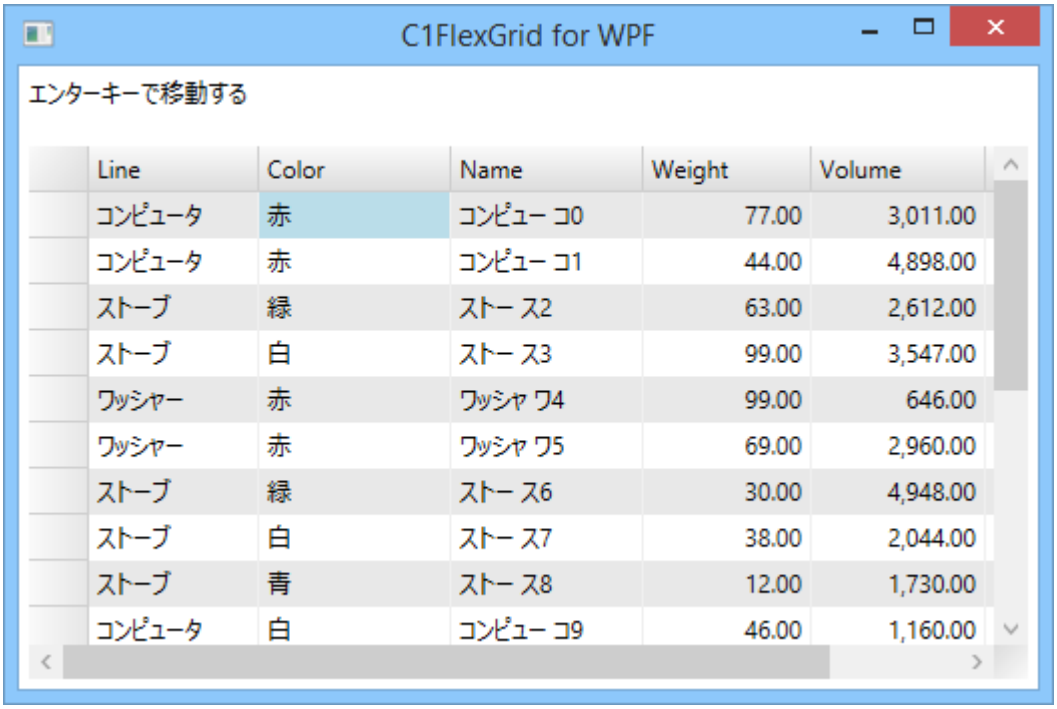
```
_flex.KeyActionEnter = C1.WPF.FlexGrid.KeyAction.MoveAcross;
```

なお、最終行・最終列のセルで Enter キーが押された際にグリッドの最初のセルに移動する場合は、別途コードで制御する必要があります。次のコードでは、グリッドをデータソースと連結していることを前提として、Enter キーでフォーカスを移動する方法を示します。

【実行例】



移動後イメージ



VisualBasic

```
Public Class MainWindow
    Public Sub New()
        InitializeComponent()
        _flex.ItemsSource = Product.GetProducts(20)
        _flex.KeyActionEnter = C1.WPF.FlexGrid.KeyAction.MoveAcross
    End Sub
    ' 編集時に最終行・最終列で[Enter]キーを押した際

    ' 最初のセルにフォーカスを移動します
    Private Sub _flex_PreviewKeyDown(sender As Object, e As KeyEventArgs) Handles
_flex.PreviewKeyDown
        If e.Key = Key.Enter And _flex.Selection.BottomRow = _flex.Rows.Count - 1
And _flex.Selection.RightColumn = _flex.Columns.Count - 1 Then
            _flex.Select(0, 0, True)
            e.Handled = True
        End If
    End Sub
End Class
```

C#

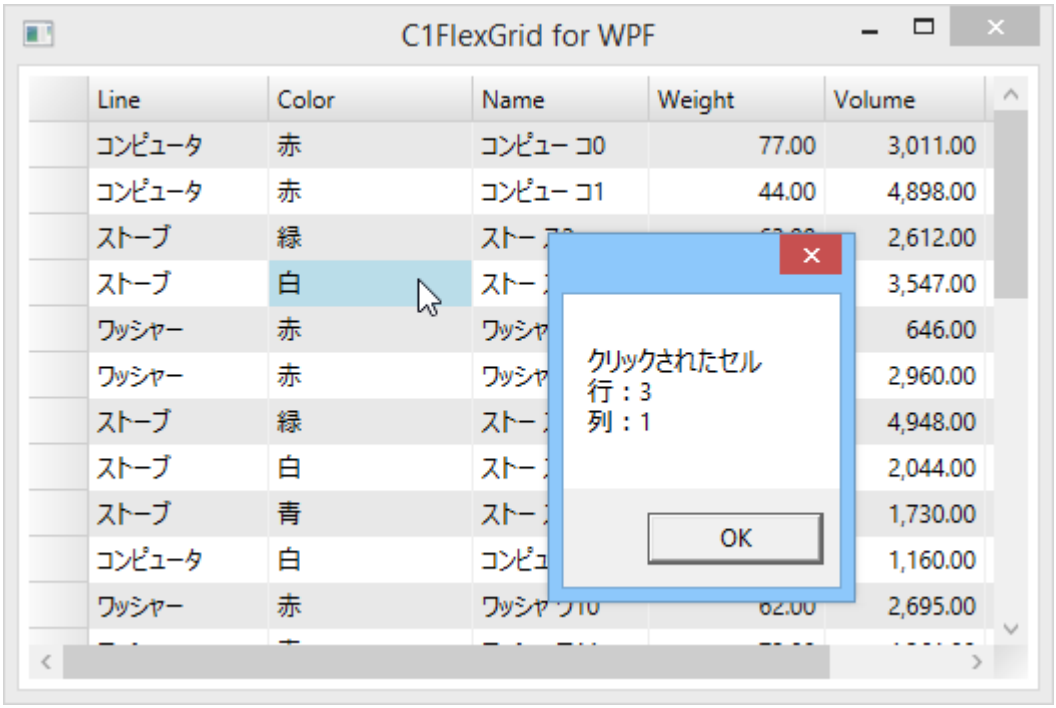
```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        _flex.ItemsSource = Product.GetProducts(20);
        _flex.KeyActionEnter = C1.WPF.FlexGrid.KeyAction.MoveAcross;
        _flex.PreviewKeyDown += _flex_PreviewKeyDown;
    }
    // 編集時に最終行・最終列で[Enter]キーを押した際

    //最初のセルにフォーカスを移動します

    private void _flex_PreviewKeyDown(object sender, KeyEventArgs e)
    {
        if (e.Key == Key.Enter
            && _flex.Selection.BottomRow == _flex.Rows.Count - 1
            && _flex.Selection.RightColumn == _flex.Columns.Count - 1)
        {
            _flex.Select(0, 0, true);
            e.Handled = true;
        }
    }
}
```

クリックされたセルを取得する

C1FlexGridでユーザーがどのパーツをクリックしたかを判定するために**HitTestInfo**を使用します。**HitTestInfo**は指摘された座標にあるコントロールのパーツに関する情報を保持します。

たとえば、次の例では、グリッドをデータソースと連結していることを前提として、クリックされたセルの行列インデックスを取得する方法を示します。

【実行例】



VisualBasic

Example Title

```
Private Sub _flex_Click(sender As Object, e As MouseButtonEventArgs) Handles
_flex.Click
    Dim ht As C1.WPF.FlexGrid.HitTestInfo = _flex.HitTest(e)
    MessageBox.Show("クリックされたセル\n" + Chr(13) + Chr(10) +
                "行:"  + ht.Row.ToString() + Chr(13) + Chr(10) +
                "列:" + ht.Column.ToString())
End Sub
```

C#

```
void _flex_Click(object sender, MouseButtonEventArgs e)
{
    var ht = _flex.HitTest(e);
    MessageBox.Show("クリックされたセル\n" +
                "行:" + ht.Row + "\n" +
                "列:" + ht.Column);
}
```

右クリックでセルを選択する

C1FlexGrid のデフォルト動作としては、右クリックでセルは移動しませんが、**MouseRightButtonDown** イベントを利用して右クリックを判別し、カレントセルを選択します。

次のコードでは、グリッドをデータソースと連結していることを前提として、右クリックでカレントセルを選択する方法を示します。

VisualBasic

```
Private Sub _flex_MouseRightButtonDown(sender As Object, e As MouseButtonEventArgs)
Handles _flex.MouseRightButtonDown
    Dim ht As C1.WPF.FlexGrid.HitTestInfo = _flex.HitTest(e)
    _flex.Select(ht.Row, ht.Column, True)
End Sub
```

C#

```
void _flex_MouseRightButtonDown(object sender, MouseButtonEventArgs e)
{
    var ht = _flex.HitTest(e);
    _flex.Select(ht.Row, ht.Column, true);
}
```

編集機能

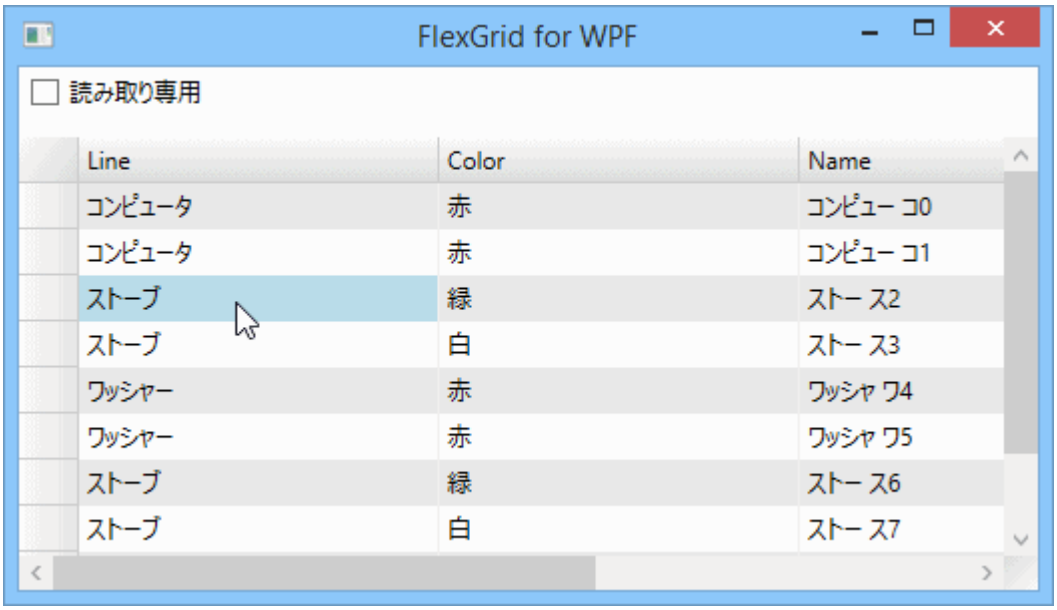
編集を許可する

C1FlexGrid の編集は、Excel の編集に似ています。セルで F2 キーを押すか、セルをダブルクリックすることで、グリッドは完全編集モードになります。このモードでは、ユーザーが[Enter]、[Tab]、または[Esc]キーを押すか、マウスを使って選択範囲を移動するまで、セルエディタはアクティブのままになります。完全編集モードでは、カーソルキーを押しても、グリッドの編集モードは終了しません。

なお、グリッドの行や列にある値の編集を制御するには、**IsReadOnly** プロパティを使用します。反対に、編集を許可するには**IsReadOnly** プロパティを False に設定します。

次のコードでは、グリッド全体の編集を制御する例を示します。

【実行例】



Visual Basic

```
Private Sub CheckBox_Checked(sender As Object, e As RoutedEventArgs)
    Dim chk = TryCast(sender, CheckBox)
    If chk IsNot Nothing Then
        Dim value = If(chk.IsChecked, False)
        C1FlexGrid.IsReadOnly = value
    End If
End Sub
```

C#

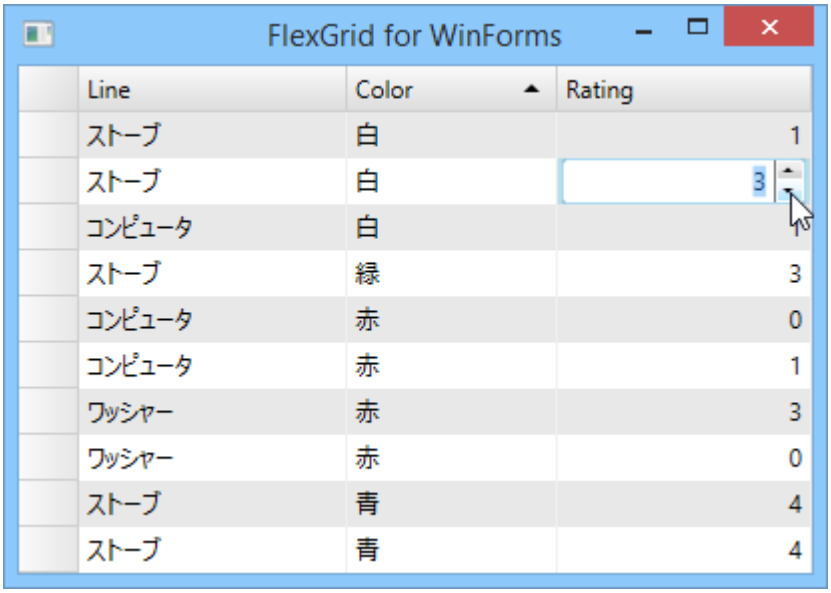
```
private void CheckBox_Checked(object sender, RoutedEventArgs e)
{
    var chk = sender as CheckBox;
    if (chk != null)
    {
        var value = chk.IsChecked ?? false;
        C1FlexGrid.IsReadOnly = value;
    }
}
```

エディタをカスタマイズする

行や列単位でエディタをカスタマイズする場合は、**C1FlexGrid** の**CellFactory** クラスを使用します。**CellFactory** クラスは**C1FlexGrid** のデフォルトのセルファクトリを実装するクラスで、**ICellFactory** インターフェイスを実装するクラスを作成してそのクラスのインスタンスを **CellFactory** プロパティに割り当ててカスタムエディタを作成することを可能とします。

次のコードでは、グリッドは Product.cs と連結していることを前提とし、CellFactory クラスをオーバーライドして**C1NumericBox** のカスタムエディタを列に埋め込む方法を示します。

【実行例】



Visual Basic

```
' カスタムエディタを作成します
Public Partial Class CustomEditor
    Inherits Window
    Public Sub New()
        InitializeComponent()
        C1FlexGrid.ItemsSource = Product.GetProducts(10)
        C1FlexGrid.CellFactory = New MyCellFactory()
    End Sub
    Public Class MyCellFactory
        Inherits CellFactory
        Public Overrides Function CreateCellEditor(grid As C1FlexGrid, cellType As
CellType, rng As CellRange) As FrameworkElement
            Dim control = MyBase.CreateCellEditor(grid, cellType, rng)
            Dim bdr = TryCast(control, Border)
            If bdr IsNot Nothing Then
                ' カスタムエディタを1列目の2行目に割り当てます
                If rng.Row = 2 AndAlso rng.Column = 1 Then
                    Dim textBox = TryCast(bdr.Child, TextBox)
                    textBox.Background = Brushes.Yellow
                End If
                ' 2行2列目のセルではC1NumericBoxをカスタムエディタに割り当てます
                If rng.Column = 2 Then
                    Dim source = grid.Rows(rng.Row).DataItem
                    Dim path = grid.Columns(rng.Column).Binding.Path
                    Dim myBinding As New Binding() With { _
                        .Path = path, _
                        .Source = source, _
                        .Mode = BindingMode.TwoWay _
                    }
                    Dim numericBox As New C1NumericBox()
                    numericBox.SetBinding(C1NumericBox.ValueProperty, myBinding)
                    bdr.Child = Nothing
                    bdr.Child = numericBox
                End If
            End If
            Return control
        End Function
    End Class
End Class
```

C#

```
// カスタムエディタを作成します
public partial class CustomEditor : Window
{
    public CustomEditor()
    {
        InitializeComponent();
        C1FlexGrid.ItemsSource = Product.GetProducts(10);
        C1FlexGrid.CellFactory = new MyCellFactory();
    }
    public class MyCellFactory : CellFactory
    {
        public override FrameworkElement CreateCellEditor(C1FlexGrid grid, CellType
cellType, CellRange rng)
        {
            var control = base.CreateCellEditor(grid, cellType, rng);
            var bdr = control as Border;
            if (bdr != null)
            {
                // カスタムエディタを1列目の2行目に割り当てます
                if (rng.Row == 2 && rng.Column == 1)
                {
                    var textBox = bdr.Child as TextBox;
                    textBox.Background = Brushes.Yellow;
                }
                // 2行2列目のセルではC1NumericBoxをカスタムエディタに割り当てます
                if (rng.Column == 2)
                {
                    var source = grid.Rows[rng.Row].DataItem;
                    var path = grid.Columns[rng.Column].Binding.Path;
                    Binding myBinding = new Binding() { Path = path, Source =
source, Mode = BindingMode.TwoWay };
                    C1NumericBox numericBox = new C1NumericBox();
                    numericBox.SetBinding(C1NumericBox.ValueProperty, myBinding);

                    bdr.Child = null;
                    bdr.Child = numericBox;
                }
            }
            return control;
        }
    }
}
```

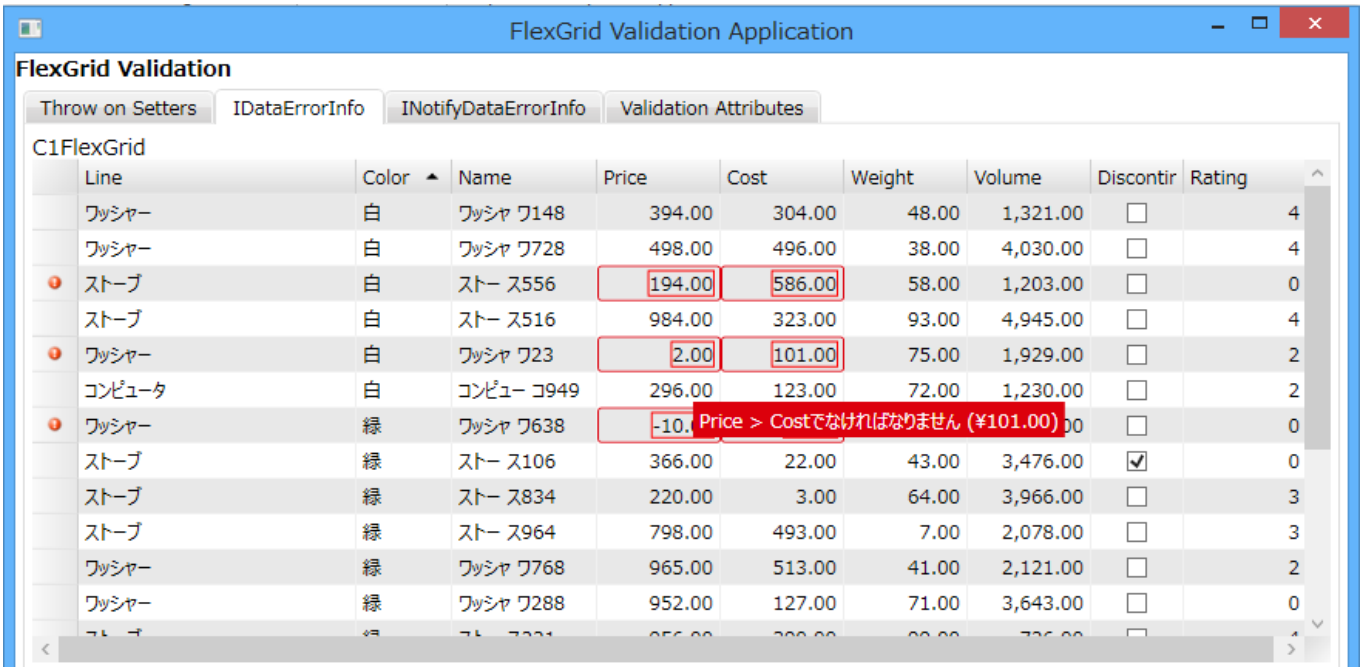
編集データをチェックする

C1FlexGrid の**ShowErrors** プロパティが True に設定されている場合、セルデータを検証できる方法を紹介します。(デフォルト値はTrue)

なお、検証エラーは、いくつかの方法でトリガされます。

1. データ項目のプロパティセッターが例外をスローした場合は、セルエディタがエラーメッセージを表示し、エラーが修正されるまで、または変更がキャンセルされるまで、エディタはアクティブな状態を維持します。
2. データ項目が **System.ComponentModel.IDataErrorInfo** インターフェイスを実装している場合は、デフォルトのインデクサを実装し、列固有のエラーメッセージ(セルエディタにも表示される)を返すことで、項目が列レベルの検証エラーを発生させることができます。または、Error プロパティをエラーメッセージに設定することで、項目レベルの検証エラーをトリガすることもできます。この場合は、いずれかの列固有のエラーとしてではなく、行ヘッダーの最初のセルにアイコンとしてエラーが表示されます。
3. データ項目が **System.ComponentModel.INotifyDataErrorInfo** インターフェイスを実装している場合は、上記の列レベルまたは項目レベルの検証エラーを非同期に発生させることができます。**System.ComponentModel.INotifyDataErrorInfo** は、**System.ComponentModel.IDataErrorInfo** より複雑で、実装も難しくなります。

【実行例】



下のコードでは、簡単な例外と System.ComponentModel.IDataErrorInfo インターフェイスを使用して、データクラスに検証を実装する方法を示します。下のコードでは、簡単な例外と System.ComponentModel.IDataErrorInfo インターフェイスを使用して、データクラスに検証を実装する方法を示します。また、このコードは製品サンプルのFlexGridSamplesを参考にしてください。

VisualBasic

```
Public Class Product
    Implements INotifyPropertyChanged
    Implements IEditableObject
    Implements IDataErrorInfo
    ' ** 方法 1:Price が負の値に設定されたときに例外をスローします
    Public Property Price() As System.Nullable(Of Double)
        Get
            Return DirectCast(GetValue("Price"), System.Nullable(Of Double))
        End Get
        Set
            If value <= 0 Then
                Throw New Exception("Price must be greater than zero!")
            End If
            SetValue("Price", value)
        End Set
    End Property

    ' ** 方法 2:特定の列のエラーを返します
    Private Default ReadOnly Property IDataErrorInfo_Item(columnName As String) As
String Implements IDataErrorInfo.this
        Get
            Dim msg As String = Nothing
            Select Case columnName
                Case "Cost"
                    If Cost <= 0 Then
                        msg = "Cost must be greater than zero!"
                    End If
                    Exit Select
            End Select
            Return msg
        End Get
    End Property

    ' ** 方法 3:行全体のエラーを返します
    ' (検証は複数の列に基づきます)
    Private ReadOnly Property IDataErrorInfo_Error() As String Implements
IDataErrorInfo.[Error]
        Get
            Return If(Price < Cost, "Price must be greater than Cost!", Nothing)
        End Get
    End Property
End Class
```

C#

```
public class Product :
        INotifyPropertyChanged,
        IEditableObject,
        IDataErrorInfo
{
    // ** 方法 1:Price が負の値に設定されたときに例外をスローします
    public double? Price
    {
        get { return (double?)GetValue("Price"); }
        set
        {
            if (value <= 0)
            {
                throw new Exception("Price must be greater than zero!");
            }
            SetValue("Price", value);
        }
    }

    // ** 方法 2:特定の列のエラーを返します
    string IDataErrorInfo.this[string columnName]
    {
        get
        {
            string msg = null;
            switch (columnName)
            {
                case "Cost":
                    if (Cost <= 0)
                    {
                        msg = "Cost must be greater than zero!";
                    }
                    break;
            }
            return msg;
        }
    }

    // ** 方法 3:行全体のエラーを返します
    // (検証は複数の列に基づきます)
    string IDataErrorInfo.Error
    {
        get
        {
            return Price < Cost
                ? "Price must be greater than Cost!"
                : null;
        }
    }
```

```
}
```

セル編集時の改行を許可する

デフォルトではセル編集時に **[Alt]+[Enter]**、または **[Ctrl]+[Enter]**、または **[Shift]+[Enter]**、**[Ctrl]+[J]**、**[Ctrl]+[Shift]+[J]**）で、テキストを改行することができます。この動作を制御する必要がある場合は、下記のように、**CellFactory** クラスをオーバーライドしたカスタムセルファクトリを使用してセル上の動作をカスタマイズできます。

Visual Basic

```
Public Sub New()
    InitializeComponent()
    C1FlexGrid.ItemsSource = Product.GetProducts(10)
    ' カスタムセルファクトリを作成します
    C1FlexGrid.CellFactory = New CustomerCellFactory()
End Sub
Public Class CustomerCellFactory
    Inherits CellFactory
    Public Overrides Function CreateCellEditor(grid As C1FlexGrid, cellType As
CellType, rng As CellRange) As FrameworkElement
        Dim control = MyBase.CreateCellEditor(grid, cellType, rng)
        Dim bdr = TryCast(control, Border)
        Dim tb = TryCast(bdr.Child, TextBox)
        AddHandler tb.PreviewKeyUp, AddressOf tb_PreviewKeyUp
        Return control
    End Function
    Private Sub tb_PreviewKeyUp(sender As Object, e As KeyEventArgs)
        '左側の[Ctrl]＋[Enter]キーを押下すると、改行を許可します
        If e.Key = Key.Enter AndAlso Keyboard.IsKeyDown(Key.LeftCtrl) Then
            Dim tb = TryCast(sender, TextBox)
            tb.TextWrapping = TextWrapping.Wrap
            If tb IsNot Nothing Then
                Dim linebreak As String = vbCr & vbLf
                Dim start = tb.SelectionStart
                Dim text1 = tb.Text.Substring(0, tb.SelectionStart)
                Dim text2 = tb.Text.Substring(tb.SelectionStart)
                Dim sb As New StringBuilder()
                Dim newText =
sb.Append(text1).Append(linebreak).Append(text2).ToString()
                start = start + linebreak.Length
                tb.Text = newText
                tb.SelectionStart = start
            End If
            e.Handled = True
        End If
    End Sub
End Class
```

C#

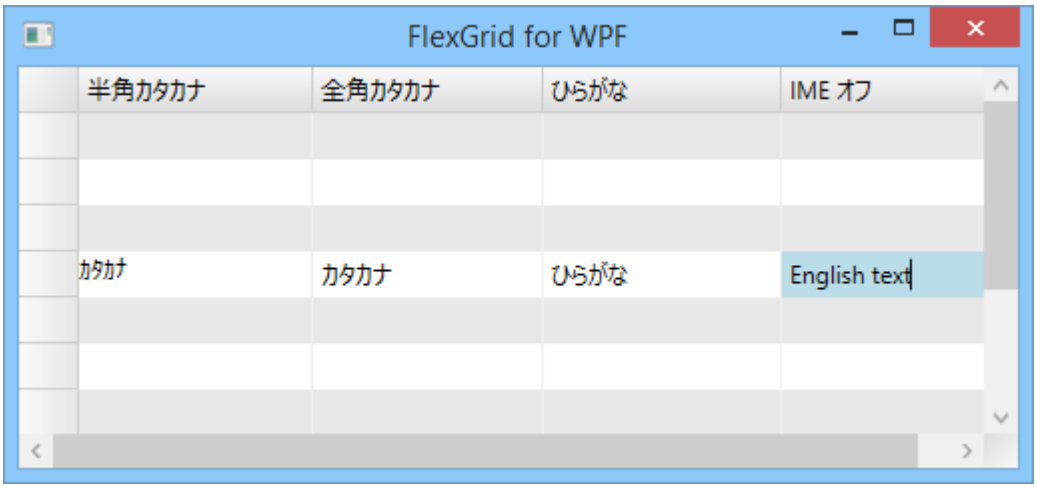
```
public MultilineEditor()
{
    InitializeComponent();
    C1FlexGrid.ItemsSource = Product.GetProducts(10);
    // カスタムセルファクトリを作成します
    C1FlexGrid.CellFactory = new CustomerCellFactory();
}

public class CustomerCellFactory : CellFactory
{
    public override FrameworkElement CreateCellEditor(C1FlexGrid grid, CellType
cellType, CellRange rng)
    {
        var control = base.CreateCellEditor(grid, cellType, rng);
        var bdr = control as Border;
        var tb = bdr.Child as TextBox;
        tb.PreviewKeyUp += tb_PreviewKeyUp;
        return control;
    }
    void tb_PreviewKeyUp(object sender, KeyEventArgs e)
    {
        //左側の[Ctrl]＋[Enter]キーを押下すると、改行を許可します
        if (e.Key == Key.Enter && Keyboard.IsKeyDown(Key.LeftCtrl))
        {
            var tb = sender as TextBox;
            tb.TextWrapping = TextWrapping.Wrap;
            if (tb != null)
            {
                string linebreak = "\r\n";
                var start = tb.SelectionStart;
                var text1 = tb.Text.Substring(0, tb.SelectionStart);
                var text2 = tb.Text.Substring(tb.SelectionStart);
                StringBuilder sb = new StringBuilder();
                var newText =
sb.Append(text1).Append(linebreak).Append(text2).ToString();
                start = start + linebreak.Length;
                tb.Text = newText;
                tb.SelectionStart = start;
            }
            e.Handled = true;
        }
    }
}
```

IMEモードを切り替える

C1FlexGrid コントロールにおいてIME制御を行うには、InputMethod クラスを使用してIMEの状態を取得や設定します。また、グリッドでセル編集時のIME 変換モードを切りかえる方法として、**InputMethod** を使用します。

たとえば、次のコードでは、列ごとに編集時のIMEの変換モードを切り替える方法を示します。最初の列はXAML上でIMEを制御する方法（編集用のTextBoxに対してIME制御関連の設定を行う）を示します。2 列目以降は、コード上でIMEを制御する方法を示します。それに加えて、セルが選択された時点で強制的に編集モードに切り替える方法も示します。

【実行例】



マークアップ

```
<c1:C1FlexGrid x:Name="c1FlexGrid1" AutoGenerateColumns="False" ItemsSource="
{Binding}" Margin="0" >
    <c1:C1FlexGrid.Columns>
        <c1:Column Header="半角カタカナ">
            <c1:Column.CellTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Column0, Mode=OneWay}" />
                </DataTemplate>
            </c1:Column.CellTemplate>
            <c1:Column.CellEditingTemplate>
                <DataTemplate>
                    <TextBox Text="{Binding Column0, Mode=TwoWay}"
InputMethod.PreferredImeState="On" InputMethod.PreferredImeConversionMode="Native" />
                </DataTemplate>
            </c1:Column.CellEditingTemplate>
        </c1:Column>
        <c1:Column x:Name="column2" Header="全角カタカナ" Binding="{Binding Column1}" />
        <c1:Column x:Name="column3" Header="ひらがな" Binding="{Binding Column2}" />
        <c1:Column x:Name="column4" Header="IME オフ" Binding="{Binding Column3}" />
    </c1:C1FlexGrid.Columns>
</c1:C1FlexGrid>
```

Visual Basic

```
Public Partial Class MainWindow
    Inherits Window
    Public Sub New()
        InitializeComponent()
        Me.DataContext = SampleData.GetSampleData()
        AddHandler Me.c1FlexGrid1.SelectionChanged, AddressOf
c1FlexGrid1_SelectionChanged
        AddHandler Me.c1FlexGrid1.BeginningEdit, AddressOf
c1FlexGrid1_BeginningEdit
        AddHandler Me.c1FlexGrid1.CellEditEnded, AddressOf
c1FlexGrid1_CellEditEnded
    End Sub
    Private Sub c1FlexGrid1_BeginningEdit(sender As Object, e As CellEditEventArgs)
        ' 条件に基づいて、TextBoxの IME 変換モードを切り替えます。
        Select Case e.Column
            Case 1
                ' 全角カタカナ
                InputMethod.Current.ImeState = InputMethodState.[On]
                InputMethod.Current.ImeConversionMode =
ImeConversionModeValues.Katakana Or ImeConversionModeValues.FullShape
                Exit Select
            Case 2
                ' ひらがな
                InputMethod.Current.ImeState = InputMethodState.[On]
                InputMethod.Current.ImeConversionMode =
ImeConversionModeValues.Native Or ImeConversionModeValues.FullShape
                Exit Select
            Case 3
                ' IME オフ
                InputMethod.Current.ImeState = InputMethodState.Off
                Exit Select
        End Select
    End Sub
    Private Sub c1FlexGrid1_CellEditEnded(sender As Object, e As CellEditEventArgs)
        InputMethod.Current.ImeState = InputMethodState.Off
    End Sub
    Private Sub c1FlexGrid1_SelectionChanged(sender As Object, e As
CellRangeEventArgs)
        ' 必要に応じて、セルを編集状態にします。
        ' これは、非編集状態でセル内に IME 入力を行おうとすると、セルに対するインライン入力が行われない
という制限に対する対処法となります。
        Me.c1FlexGrid1.StartEditing(False)
    End Sub
End Class
```

C#

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        this.DataContext = SampleData.GetSampleData();

        this.c1FlexGrid1.SelectionChanged += c1FlexGrid1_SelectionChanged;
        this.c1FlexGrid1.BeginningEdit += c1FlexGrid1_BeginningEdit;
        this.c1FlexGrid1.CellEditEnded += c1FlexGrid1_CellEditEnded;
    }
    private void c1FlexGrid1_BeginningEdit(object sender, CellEditEventArgs e)
    {
        // 条件に基づいて、TextBoxの IME 変換モードを切り替えます。
        switch (e.Column)
        {
            case 1: // 全角カタカナ
                InputMethod.Current.ImeState = InputMethodState.On;
                InputMethod.Current.ImeConversionMode =
ImeConversionModeValues.Katakana | ImeConversionModeValues.FullShape;
                break;
            case 2: // ひらがな
                InputMethod.Current.ImeState = InputMethodState.On;
                InputMethod.Current.ImeConversionMode =
ImeConversionModeValues.Native | ImeConversionModeValues.FullShape;
                break;
            case 3: // IME オフ
                InputMethod.Current.ImeState = InputMethodState.Off;
                break;
        }
    }
    private void c1FlexGrid1_CellEditEnded(object sender, CellEditEventArgs e)
    {
        InputMethod.Current.ImeState = InputMethodState.Off;
    }

    private void c1FlexGrid1_SelectionChanged(object sender, CellRangeEventArgs e)
    {
        // 必要に応じて、セルを編集状態にします。
        // これは、非編集状態でセル内に IME 入力を行おうとすると、セルに対するインライン入力が行われない
という制限に対する対処法となります。
        this.c1FlexGrid1.StartEditing(false);
    }
}
```

Enterキーで編集を開始する

既定では**Enter**キーでセルを移動する動作となりますが、**Enter**キーによって編集を開始するようにするには、**StartEditing** メソッドを使用できます。

次のコードでは、PreviewKeyUp / PreviewKeyDown イベント内に**StartEditing** メソッドでセルを強制に編集モードに設定する方法を示します。

Visual Basic

```
Public Sub New()
    InitializeComponent()
    For i As Integer = 0 To 9
        _flex.Columns.Add(New Column())
    Next
    For j As Integer = 0 To 9
        _flex.Rows.Add(New Row())
    Next
    For row As Integer = 0 To 9
        For col As Integer = 0 To 9
            _flex(row, col) = String.Format("[{0},{1}]", row.ToString(),
col.ToString())
        Next
    Next
    AddHandler _flex.PreviewKeyDown, AddressOf _flex_PreviewKeyDown
    AddHandler _flex.PreviewKeyUp, AddressOf _flex_PreviewKeyUp
End Sub
' Enterキーによる編集の開始を有効にします
Private Sub _flex_PreviewKeyUp(sender As Object, e As KeyEventArgs)
    If e.Key = Key.Enter Then
        e.Handled = True
    End If
End Sub
Private Sub _flex_PreviewKeyDown(sender As Object, e As KeyEventArgs)
    If e.Key = Key.Enter Then
        _flex.StartEditing(False, _flex.Selection.Row, _flex.Selection.Column)
        e.Handled = True
    End If
End Sub
```

C#

```
public MainWindow()
{
    InitializeComponent();
    for (int i = 0; i < 10; i++)
    {
        _flex.Columns.Add(new Column());
    }
    for (int j = 0; j < 10; j++)
    {
        _flex.Rows.Add(new Row());
    }
    for (int row = 0; row < 10; row++)
    {
        for (int col = 0; col < 10; col++)
        {
            _flex[row, col] = string.Format("[{0},{1}]", row.ToString(),
col.ToString());
        }
    }
    _flex.PreviewKeyDown += _flex_PreviewKeyDown;
    _flex.PreviewKeyUp += _flex_PreviewKeyUp;
}
// Enterキーによる編集の開始を有効にします
void _flex_PreviewKeyUp(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Enter)
        e.Handled = true;
}

void _flex_PreviewKeyDown(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Enter)
    {
        _flex.StartEditing(false, _flex.Selection.Row, _flex.Selection.Column);
        e.Handled = true;
    }
}
```

矢印キーで編集を終了させない

矢印キーで編集が終了されないようにするには、**StartEditing()** を呼び出すだけで済みます。次の例をご参照ください。

Visual Basic

```
    InitializeComponent()

    Using _flex.Rows.DeferNotifications()
        Dim rowCount As Integer = 10
        Dim colCount As Integer = 7
        For i As Integer = 0 To rowCount - 1
            _flex.Rows.Add(New C1.WPF.FlexGrid.Row())
        Next
        For i As Integer = 0 To colCount - 1
            _flex.Columns.Add(New C1.WPF.FlexGrid.Column())
        Next
        For r As Integer = 0 To rowCount - 1
            For c As Integer = 0 To colCount - 1
                _flex(r, c) = String.Format("[{0},{1}]", r.ToString(),
c.ToString())
            Next
        Next
        ' 矢印キーによる編集の終了を無効にします
        _flex.StartEditing(True)
    End Using
End Sub
```

C#

```
InitializeComponent();
using (_flex.Rows.DeferNotifications())
{
    int rowCount = 10;
    int colCount = 7;
    for (int i = 0; i < rowCount; i++)
    {
        _flex.Rows.Add(new C1.WPF.FlexGrid.Row());
    }
    for (int i = 0; i < colCount; i++)
    {
        _flex.Columns.Add(new C1.WPF.FlexGrid.Column());
    }
    for (int r = 0; r < rowCount; r++)
    {
        for (int c = 0; c < colCount; c++)
        {
            _flex[r, c] = string.Format("[{0},{1}]", r.ToString(), c.ToString());
        }
    }
    // 矢印キーによる編集の終了を無効にします
    _flex.StartEditing(true);
}
```

ダブルクリック時にテキストを選択状態にする

グリッドでセルをダブルクリックするまたはF2キーを押すと、セルのテキスト全体が選択されます。これは **C1FlexGrid** のデフォルト動作となりますが、セルに直接入力を行う場合や編集状態のとき入力を行う場合、新たに入力した値を以前の値の最後の位置に挿入することも可能です。**C1FlexGrid** でカスタム **CellFactory** クラスを実装して実現できます。たとえば、次のコードでは、連結グリッドで直接入力を行ったとき追記編集を実現する方法を示します。

Visual Basic

```
Public Sub New()
    InitializeComponent()

    Dim p As ICollectionView = Product.GetProducts(100)
    _flex.ItemsSource = p
    _flex.CellFactory = New MyCellFactory()
End Sub
' カスタムセルファクトリを作成します
Public Class MyCellFactory
    Inherits CellFactory
    Private originText As String = String.Empty
    Public Overrides Function CreateCellEditor(grid As C1FlexGrid, cellType As
CellType, rng As CellRange) As FrameworkElement
        Dim control = MyBase.CreateCellEditor(grid, cellType, rng)
        Dim bdr = TryCast(control, Border)
        Dim txt = TryCast(bdr.Child, TextBox)
        originText = txt.Text
        AddHandler txt.TextChanged, AddressOf txt_TextChanged
        Return control
    End Function

    Private Sub txt_TextChanged(sender As Object, e As TextChangedEventArgs)
        Dim t = TryCast(sender, TextBox)
        RemoveHandler t.TextChanged, AddressOf txt_TextChanged
        t.Text = originText & Convert.ToString(t.Text)
        t.SelectionStart = t.Text.Length
    End Sub
End Class
```

C#

```
public partial class MainWindow : Window
{

    public MainWindow()
    {
        InitializeComponent();
        _flex.ItemsSource = Product.GetProducts(100);
        _flex.CellFactory = new MyCellFactory();
    }
}
// カスタムセルファクトリを作成します
public class MyCellFactory : CellFactory
{
    string originText = string.Empty;
    public override FrameworkElement CreateCellEditor(C1FlexGrid grid, CellType
cellType, CellRange rng)
    {
        var control = base.CreateCellEditor(grid, cellType, rng);
        var bdr = control as Border;
        var txt = bdr.Child as TextBox;
        originText = txt.Text;
        txt.TextChanged += txt_TextChanged;
        return control;
    }
    void txt_TextChanged(object sender, TextChangedEventArgs e)
    {
        var t = (sender as TextBox);
        t.TextChanged -= txt_TextChanged;
        t.Text = originText + t.Text;
        t.SelectionStart = t.Text.Length;
    }

}
```

常時入力モードにする

C1FlexGrid の編集はデフォルトで有効になっています。そして、通常入力動作は MSExcel に似ています。セルにテキストを直接入力すると、グリッドはクイック編集モードになります。ユーザーが Enter、Tab、Esc、またはいずれかの矢印キーを押すまで、セルエディタはアクティブなままになります。クイック編集モードでは、カーソルキーを押すと、グリッドの編集モードは終了します。

なお、グリッドがフォーカスを得たとき、または選択されたセルが変更されたときに、グリッドを自動的に入力モードにするには、次のように**StartEditing** メソッドを使用できます。

Visual Basic

```
Public Sub New()
    InitializeComponent()

    For i As Integer = 0 To 9
        _flex.Rows.Add(New Row())
    Next
    For j As Integer = 0 To 9
        _flex.Columns.Add(New Column())
    Next
    For r As Integer = 0 To 9
        For c As Integer = 0 To 9
            _flex(r, c) = String.Format("[{0},{1}]", r.ToString(), c.ToString())
        Next
    Next
End Sub

Private Sub _flex_SelectionChanged(sender As Object, e As CellRangeEventArgs)
Handles _flex.SelectionChanged
    _flex.StartEditing(False, _flex.Selection.Row, _flex.Selection.Column)
End Sub
```

C#

```
public MainWindow()
{
    InitializeComponent();
    for (int i = 0; i < 10; i++)
    {
        _flex.Rows.Add(new Row());
    }
    for (int j = 0; j < 10; j++)
    {
        _flex.Columns.Add(new Column());
    }
    for (int row = 0; row < 10; row++)
    {
        for (int col = 0; col < 10; col++)
        {
            _flex[row, col] = String.Format("[{0},{1}]", row, col);
        }
    }
    _flex.SelectionChanged += _flex_SelectionChanged;
}

void _flex_SelectionChanged(object sender, CellRangeEventArgs e)
{
    _flex.StartEditing(false, _flex.Selection.Row, _flex.Selection.Column);
}
```

オートコンプリートとマップ列

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って実装されます。

以下に、このクラスで処理するオートコンプリート機能の 2 つのシナリオを示します

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ことができる列。たとえば、文字列型の「都道府県」列と都道府県のリストがあるとします。ユーザーはリストから都道府県を選択しなければならず、リストにない都道府県は入力できません。

このシナリオは、2 行のコードを使って処理できます。

WPF

```
var c = _flexEdit.Columns["都道府県"];
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）

いくつかの特定の値を含み、それ以外の値も受け入れる列。たとえば、文字列型の「都道府県」列がある場合に、ユーザーが簡単に選択できるように都道府県のリストを提供しつつ、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 行のコードを使って処理できます。

WPF

```
var c = _flexEdit.Columns["都道府県"];
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。

次の項目に、データマップ機能の詳細を示します。

データマップ列

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、列の内容は都道府県コードを表す整数であっても、対応する都道府県をユーザーが表示および編集できるようにします。

次のセクションでは、FlexGrid .NET 4.5.2および.NET 5 バージョンでデータマップ列を実装する方法を学習します。

.NET Framework

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#

```
// Key-Valueディクショナリを構築します
var dct = new Dictionary<int, string>();
foreach (var country in GetCountryNames())
{
  dct[dct.Count] = country;
}

// 辞書を列に割り当てます
var c = _flexEdit.Columns["CountryID"];
c.ValueConverter = new ColumnValueConverter(dct);
c.HorizontalAlignment = HorizontalAlignment.Left;
```

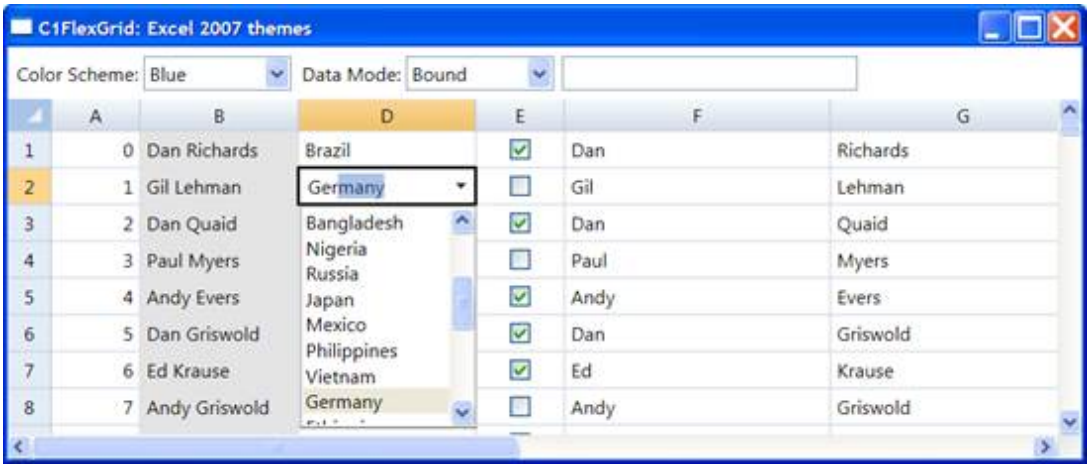
このコードは、最初に、都道府県コードの値（整数）を都道府県（文字列）にマップする Dictionary を構築します。

次に、その Dictionary を使って **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、Dictionary に存在するすべての都道府県を選択でき、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、氏名を表示するため、左揃えに設定します。

以下の図は、リストから値を選択しているときのエディターの外観を示しています。エディターがスマートオートコンプリートをサポートしていることに注目してください。ユーザーが「Ger」と入力すると、ドロップダウンは自動的に唯一の有効なオプション「Germany」を選択します（「Guatemala」、「Eritrea」、「Romania」の順に選択しません）。



.NET

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#

```
public void DataMapColumn()
{
    //Hide Country Column
    var country_col = _flexEdit.Columns["Country"];
    country_col.IsVisible = false;

    //Make the CountryID column read only
    var c = _flexEdit.Columns["CountryID"];
    c.IsReadOnly = true;

    //Display Country name instead of CountryID
    //using GridDataMap
    var columnMap = new GridDataMap();
    columnMap.ItemsSource = view;
    columnMap.SelectedValuePath = "CountryID";
    columnMap.DisplayMemberPath = "Country";
    c.DataMap = columnMap;
```

このコードは、**SelectedValuePath**プロパティと**DisplayMemberPath**プロパティを使用して、Country IDキー値（整数）をCountry Names（文字列）にマップする**GridDataMap**クラスを使用します。

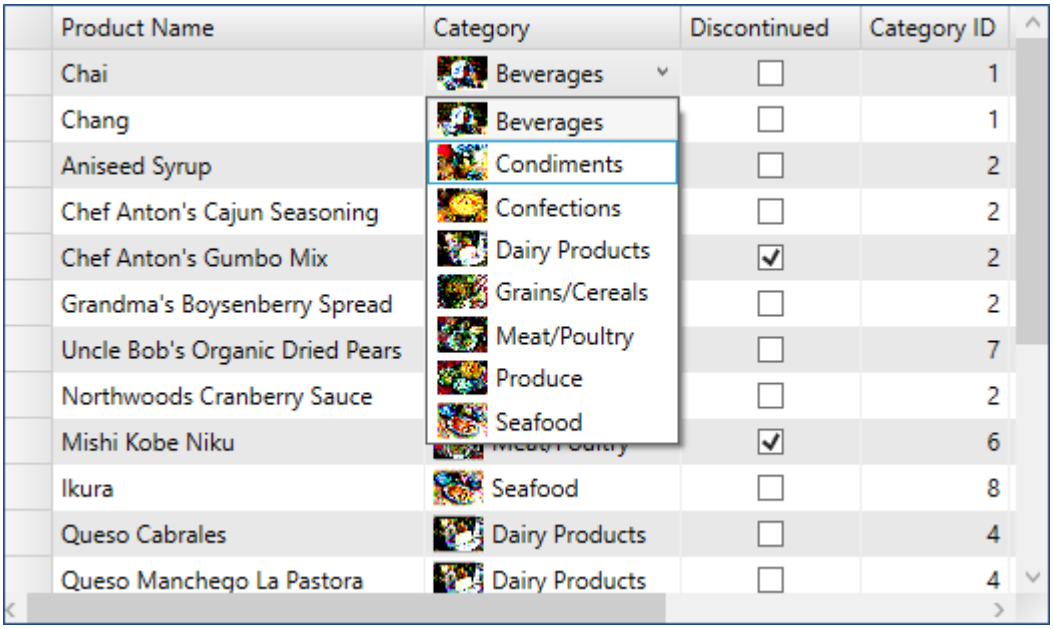
ItemsSourceプロパティは、コレクションをマップするために使用されます。以下の図は、リストから値を選択しているときのエディターの外観を示しています。

ID	Name	CountryID	Active	First	Last	Hired
1	Jack Myers	Germany	☐	Jack	Myers	12/08/2019
2	Dan Trask	Indonesia	☑	Dan	Trask	01/10/2019
3	Jack Bishop	Egypt	☐	Jack	Bishop	01/08/2019
4	Dan Jammers	Egypt	☑	Dan	Jammers	07/08/2019
5	Ted Richards	Egypt	☑	Ted	Richards	02/04/2020
6	Rich Lehman	Iran	☑	Rich	Lehman	09/10/2019
7	Vic Heath	Egypt	☐	Vic	Heath	18/12/2019
8	Noah Evers	Congo	☑	Noah	Evers	22/03/2020
9	Paul Orsted	ited Kingdom	☐	Paul	Orsted	18/03/2020
10	Larry Richards	India	☑	Larry	Richards	29/10/2019

カスタムエディタの使用

FlexGrid コントロールには、チェックボックスとコンボボックスという 2 つの組み込みエディタが付属します。チェックボックスは Boolean 値を表すためにサポートされ、コンボボックスでは、テキストを入力したり、オートコンプリートリストから値を選択することができます。このコンボボックスは、通常のテキストボックスをオートコンプリートリスト付きに拡張する **C1FlexComboBox** クラスを実装します。

次の図に、FlexGrid でエディタとして使用されるチェックボックスとコンボボックスを示します。



ただし、FlexGrid では、組み込みエディタ以外にも、独自のエディタを作成して使用できます。カスタムエディタを作成するには、次の方法があります。

- 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。

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** イベントを使用することで、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.SelectionBrush = new SolidColorBrush(Colors.Blue);
  tb.SelectionTextBrush = new SolidColorBrush(Colors.Yellow);
}
```

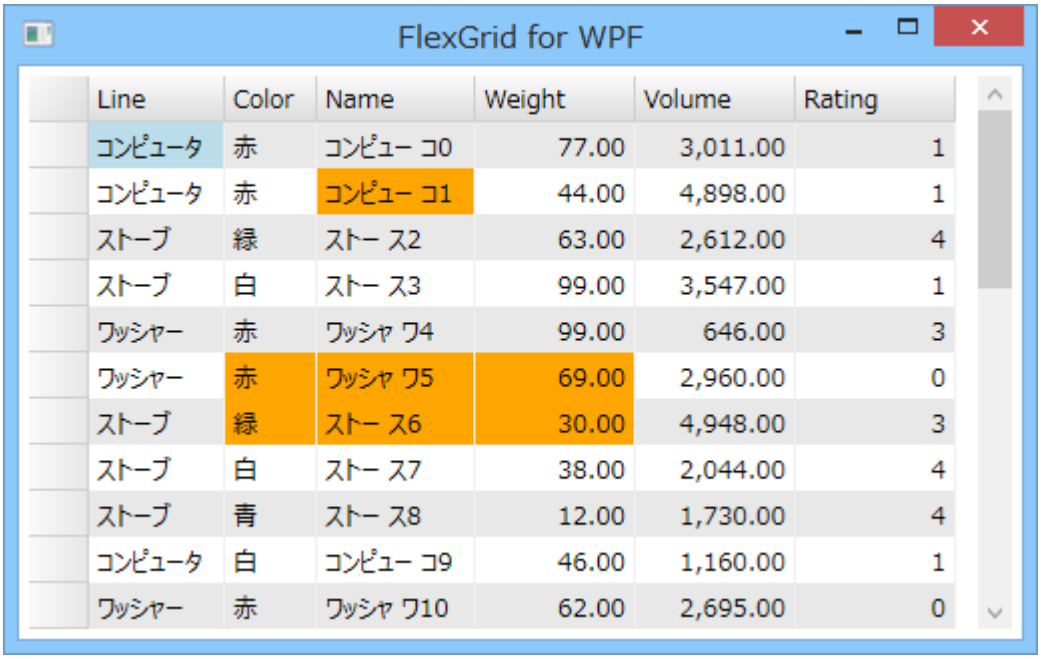
スタイル

セルのスタイルを設定する

CellFactoryクラスを継承したカスタムセルファクトリを使用して特定のセルやセル範囲をアクセスする手法でセルに任意のスタイルを設定することができます。

次のコードでは、グリッドをデータソースと連結していることを前提として、**CellFactory** クラスの使用方法を示します。

【実行例】



	Line	Color	Name	Weight	Volume	Rating
	コンピュータ	赤	コンピュー コ0	77.00	3,011.00	1
	コンピュータ	赤	コンピュー コ1	44.00	4,898.00	1
	ストーブ	緑	ストー ス2	63.00	2,612.00	4
	ストーブ	白	ストー ス3	99.00	3,547.00	1
	ワッシャー	赤	ワッシャ ワ4	99.00	646.00	3
	ワッシャー	赤	ワッシャ ワ5	69.00	2,960.00	0
	ストーブ	緑	ストー ス6	30.00	4,948.00	3
	ストーブ	白	ストー ス7	38.00	2,044.00	4
	ストーブ	青	ストー ス8	12.00	1,730.00	4
	コンピュータ	白	コンピュー コ9	46.00	1,160.00	1
	ワッシャー	赤	ワッシャ ワ10	62.00	2,695.00	0

VisualBasic

```
'CellFactoryクラスを継承してカスタムセルファクトリを使用します
Public Class CustomCellFactory
    Inherits CellFactory
    Public Overrides Function CreateCell(grid As C1FlexGrid, cellType As CellType,
rng As CellRange) As FrameworkElement
        Dim ctrl = MyBase.CreateCell(grid, cellType, rng)
        Dim bdr = DirectCast(ctrl, Border)

        If bdr IsNot Nothing AndAlso rng.Row = 1 AndAlso rng.Column = 2 Then
            bdr.Background = Brushes.Orange
        End If

        Dim cellRange As New CellRange(5, 1, 6, 3)
        If cellRange.Contains(rng) Then
            bdr.Background = Brushes.Orange
        End If
        Return ctrl
    End Function
End Class
```

C#

```
//CellFactoryクラスを継承してカスタムセルファクトリを使用します
public class CustomCellFactory : CellFactory
{
    public override FrameworkElement CreateCell(C1FlexGrid grid, CellType cellType,
CellRange rng)
    {
        var ctrl = base.CreateCell(grid, cellType, rng);
        var bdr = ctrl as Border;

        if (bdr != null &&
            rng.Row == 1 && rng.Column == 2)
        {
            bdr.Background = Brushes.Orange;
        }

        CellRange cellRange = new CellRange(5, 1, 6, 3);
        if (cellRange.Contains(rng))
        {
            bdr.Background = Brushes.Orange;
        }
        return ctrl;
    }
}
```

セルのスタイル設定

FlexGrid provides an ability to style the cell in an easier way. You can set the style of the column through column header menu at runtime as well as through code and define a cell style with a specific background, foreground and border. The following gif shows different style applied on the columns.

Id	First Name	Last Name	Last Order Date	Order Total	Last Order Time
0	Quince	Griswold	01-06-2021	₹ 6,085.02	07:00
1	Ben	Lehman	08-11-2020	₹ 1,972.69	19:41
2	Mark	Paulson	13-07-2021	₹ 5,999.99	07:34
3	Charlie	Richards	22-09-2020	₹ 4,871.29	22:26
4	Ben	Orsted	18-10-2020	₹ 9,139.66	21:51
5	Steve	Heath	18-06-2021	₹ 6,493.88	07:08
6	Ed	Heath	06-07-2021	₹ 6,034.46	13:22
7	Rich	Krause	16-11-2020	₹ 9,565.30	08:00
8	Xavier	Cole	21-10-2020	₹ 296.45	05:54
9	Zeb	Richards	16-01-2021	₹ 9,871.32	11:37
10	Jack	Bishop	29-05-2021	₹ 2,370.33	15:52
11	Jack	Danson	21-05-2021	₹ 8,440.73	20:31

The following code illustrates adding cell style option in the column header menu.

C#

```
private Color?[] _colors = new Color?[] { null, Colors.AliceBlue,
Colors.AntiqueWhite, Colors.Azure, Colors.Beige, Colors.Bisque,
Colors.BlanchedAlmond, Colors.BurlyWood, Colors.Cornsilk,
Colors.LightGoldenrodYellow, Colors.Linen, Colors.MistyRose, Colors.Moccasin,
Colors.PapayaWhip, Colors.SeaShell, Colors.Black };
private void OnColumnOptionsLoading(object sender, GridColumnOptionsLoadingEventArgs
e)
{
    if (e.Menu.Items.Count > 0)
        e.Menu.Items.Add(new C1MenuSeparator());
    var menuItem = new C1MenuItem();
    menuItem.Header = "Column background";
    menuItem.Icon = new Border { BorderThickness = new Thickness(1), BorderBrush =
new SolidColorBrush(Colors.Black), Background = e.Column.Background };
    foreach (var color in _colors)
    {
        var colorMenuItem = new C1MenuItem();
        colorMenuItem.Tag = color;
        colorMenuItem.Header = color?.ToString() ?? "None";
        colorMenuItem.Icon = new Border { BorderThickness = new Thickness(1),
BorderBrush = new SolidColorBrush(Colors.Black), Background = color.HasValue ? new
SolidColorBrush(color.Value) : null };
        colorMenuItem.Click += (s, oe) =>
        {
```

```
            var color = (Color?)(s as C1MenuItem).Tag;
            e.Column.Background = color.HasValue ? new SolidColorBrush(color.Value) :
null;
            e.Close(); //This closes the menu.
        };
        menuItem.Items.Add(colorMenuItem);
    }
}
```

条件付き書式を設定する

グリッドでセルのデータに応じて書式を設定する方式では、C1FlexGrid の Columnオブジェクトに含まれる CellTemplate および CellEditingTemplate プロパティを使用して列内のセルを編集するためのビジュアル要素を指定します。

たとえば、次の例では、セルの値に応じて赤や緑色で表示する場合、条件付き列に対して CellTemplate を指定してカスタムセルを作成します。CellTemplate は必要なバウンドプロパティを含むTextBlock 要素を持ちます。

マークアップ

```
<c1:Column Binding="{Binding Price}">
    <c1:Column.CellTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Price, StringFormat=n0}"
HorizontalAlignment="Right"
            Foreground="{Binding Price, Converter={StaticResource
ForegroundConverter}, ConverterParameter={StaticResource PriceRange} }"
            FontWeight="{Binding Price, Converter={StaticResource
FontWeightConverter}, ConverterParameter={StaticResource PriceRange} }" />
        </DataTemplate>
    </c1:Column.CellTemplate>
</c1:Column>
```

上記のコードでは、カスタムセルテンプレートの列を作成してデータソースの「Price」フィールドを TextBlock 要素内に「Text」、「Foreground」、「FontWeight」プロパティに連結しています。なお、「Text」プロパティはDouble型で自動に文字列に変換されてデータに連結できます。しかし、「Foreground」、「FontWeight」プロパティの場合はDouble値はブラシなどに変換できませんので、コンバータを指定する必要があります。

ForegroundConverter と FontWeightConverter コンバータは、範囲を指定するパラメータをサポートしています。コンバータはこの範囲を使って適当なブラッシュやフォントウエートを選択します。また、コンバータとパラメータの両方がページリソースとしてXAML内に指定されます。詳細については、次のコードをご参照ください。

【実行例】

マークアップ

```
<Window.Resources>
    <!-- ForeGroundConverter.cs 内にコンバータを実現します-->
    <local:ForegroundConverter x:Key="ForegroundConverter" />
    <!-- FontWeightConverter.cs 内にコンバータを実現します-->
    <local:FontWeightConverter x:Key="FontWeightConverter" />
    <!-- コンバータ範囲 （ポイントの値として実現し、
         X は最低値と Y は最高値となります） -->
    <Point x:Key="PriceRange" X="500" Y="1500" />
    <Point x:Key="WeightRange" X="200" Y="1000" />
    <Point x:Key="CostRange" X="200" Y="1000" />
    <Point x:Key="VolumeRange" X="2000" Y="6000" />
</Window.Resources>
```

コンバータの実現方法については、次のコードをご参照ください。

VisualBasic

```
'ForegroundConverter
Public Class ForegroundConverter
    Implements IValueConverter
    Shared _brBlack As New SolidColorBrush(Colors.Black)
    Shared _brRed As New SolidColorBrush(Colors.Red)
    Shared _brGreen As New SolidColorBrush(Colors.Green)

    Public Function Convert(value As Object, targetType As Type, parameter As
Object, culture As Globalization.CultureInfo) As Object Implements
System.Windows.Data.IValueConverter.Convert
        Dim range = DirectCast(parameter, Point)
        Dim val As Double = CDbl(value)
        Return If(val < range.X, _brRed, If(val > range.Y, _brGreen, _brBlack))
    End Function
    Public Function ConvertBack(value As Object, targetType As Type, parameter As
Object, culture As Globalization.CultureInfo) As Object Implements
System.Windows.Data.IValueConverter.ConvertBack
        Throw New NotImplementedException()
    End Function
End Class
```

```
'FontWeightConverter
Public Class FontWeightConverter
    Implements IValueConverter
    Public Function Convert(value As Object, targetType As Type, parameter As
Object, culture As Globalization.CultureInfo) As Object Implements
System.Windows.Data.IValueConverter.Convert
        Dim range = DirectCast(parameter, Point)
        Dim val As Double = CDbl(value)
        Return If(val < range.X, FontWeights.Bold, If(val > range.Y,
FontWeights.Bold, FontWeights.Normal))
    End Function
    Public Function ConvertBack(value As Object, targetType As Type, parameter As
Object, culture As Globalization.CultureInfo) As Object Implements
System.Windows.Data.IValueConverter.ConvertBack
        Throw New NotImplementedException()
    End Function
End Class
```

C#

```
//ForegroundConverter
public class ForegroundConverter : System.Windows.Data.IValueConverter
{
    static SolidColorBrush _brBlack = new SolidColorBrush(Colors.Black);
    static SolidColorBrush _brRed = new SolidColorBrush(Colors.Red);
    static SolidColorBrush _brGreen = new SolidColorBrush(Colors.Green);

    public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
    {
        var range = (Point)parameter;
        double val = (double)value;
        return
            val < range.X ? _brRed :
            val > range.Y ? _brGreen :
            _brBlack;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

```
//FontWeightConverter
public class FontWeightConverter : System.Windows.Data.IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
    {
        var range = (Point)parameter;
        double val = (double)value;
        return
            val < range.X ? FontWeights.Bold :
            val > range.Y ? FontWeights.Bold :
            FontWeights.Normal;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

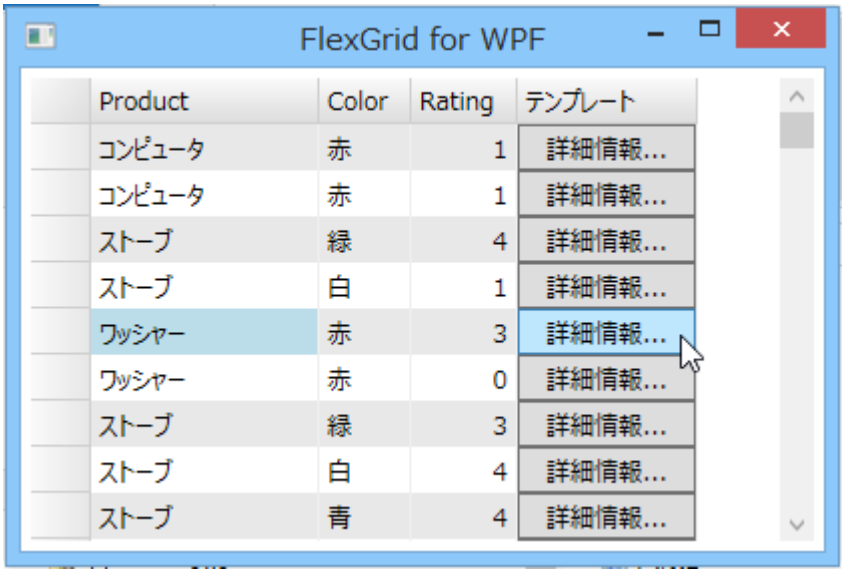
ボタン

ボタンを表示する

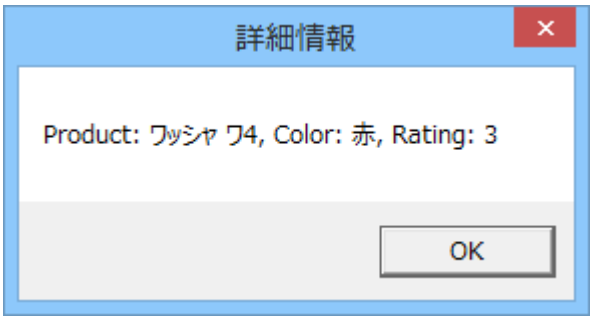
Column クラスの **CellTemplate** プロパティは編集モードでないセルのテンプレートを定義します。**C1FlexGrid** で **CellTemplate** ロパティを使用してテンプレートセルを作成できます。

たとえば、次の例は、ボタンセルのテンプレート列を追加してボタンクリックで行の情報を表示する方法を示します。

【実行例】



ボタンクリックで表示する詳細情報メッセージボックス



マークアップ

```
<c1:C1FlexGrid x:Name="_flex" Grid.Row="1" AutoGenerateColumns="False"
Margin="5,5,5,5" >
    <c1:C1FlexGrid.Columns>
        <c1:Column Header="Product" Binding="{Binding Line}" />
        <c1:Column Header="Color" Binding="{Binding Color}" />
        <c1:Column Header="Rating" Binding="{Binding Rating}" />
        <!--テンプレート列を追加します-->
        <c1:Column ColumnName="テンプレート">
            <c1:Column.CellTemplate>
                <DataTemplate>
                    <Button Content="詳細情報..." Click="ShowDetail_Click" />
                </DataTemplate>
            </c1:Column.CellTemplate>
        </c1:Column>
    </c1:C1FlexGrid.Columns>
</c1:C1FlexGrid>
```

VisualBasic

```
Private Sub ShowDetail_Click(sender As Object, e As RoutedEventArgs)
    Dim product = GetProduct(sender)
    If product IsNot Nothing Then
        Dim detail = String.Format("Product: {0}, Color: {1}, Rating: {2}",
product.Name, product.Color, product.Rating)
        MessageBox.Show(detail, "詳細情報", MessageBoxButton.OK)
    End If
End Sub

'グリッド上にコントロールとして表示されるProductを取得します
Private Function GetProduct(control As Object) As Product
    Dim e As FrameworkElement = TryCast(control, FrameworkElement)
    Return TryCast(e.DataContext, Product)
End Function
```

C#

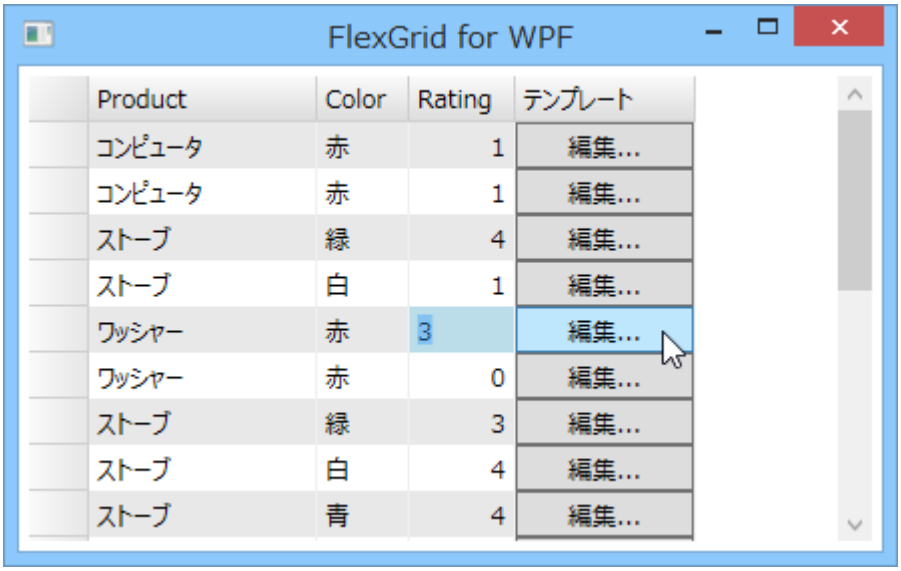
```
private void ShowDetail_Click(object sender, RoutedEventArgs e)
{
    var product = GetProduct(sender);
    if (product != null)
    {
        var detail = string.Format("Product: {0}, Color: {1}, Rating: {2}",
            product.Name, product.Color, product.Rating);
        MessageBox.Show(detail, "詳細情報", MessageBoxButton.OK);
    }
}

//グリッド上にコントロールとして表示されるProductを取得します
Product GetProduct(object control)
{
    FrameworkElement e = control as FrameworkElement;
    return e.DataContext as Product;
}
```

編集ボタンを表示する

Column クラスの **CellEditingTemplate** プロパティは編集モードでセルを作成します。また、**CellEditingTemplate** プロパティは XAML で定義でき、列内のセルを表すビジュアル要素の作成に使用されます。

たとえば、次の例は、ボタンセルのテンプレート列を追加してボタンクリックでセルを編集モードにする方法を示します。

【実行例】



マークアップ

```
<c1:C1FlexGrid x:Name="_flex" Grid.Row="1" AutoGenerateColumns="False"
Margin="5,5,5,5">
    <c1:C1FlexGrid.Columns>
        <c1:Column Header="Product" Binding="{Binding Line}"/>
        <c1:Column Header="Color" Binding="{Binding Color}"/>
        <c1:Column Header="Rating" Binding="{Binding Rating}"/>
        <c1:Column ColumnName="テンプレート">
            <c1:Column.CellTemplate>
                <DataTemplate>
                    <Button Content="編集..." Click="Button_Click" />
                </DataTemplate>
            </c1:Column.CellTemplate>
        </c1:Column>
    </c1:C1FlexGrid.Columns>
</c1:C1FlexGrid>
```

VisualBasic

```
Private Sub Button_Click(sender As Object, e As RoutedEventArgs)
    Dim product = GetProduct(sender)
    Dim list = TryCast(_flex.CollectionView.SourceCollection, IList(Of Product))
    Dim index = list.IndexOf(product)
    If index >= 0 Then
        _flex.StartEditing(False, index, 2)
    End If
End Sub
'グリッド上にコントロールとして表示されるProductを取得します
Private Function GetProduct(control As Object) As Product
    Dim e As FrameworkElement = TryCast(control, FrameworkElement)
    Return TryCast(e.DataContext, Product)
End Function
```

C#

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var product = GetProduct(sender);
    var list = _flex.CollectionView.SourceCollection as IList<Product>;
    var index = list.IndexOf(product);
    if (index >= 0)
    {
        _flex.StartEditing(false, index, 2);
    }
}
//グリッド上にコントロールとして表示されるProductを取得します
Product GetProduct(object control)
{
    FrameworkElement e = control as FrameworkElement;
    return e.DataContext as Product;
}
```

削除ボタンを表示する

C1FlexGrid で **CellTemplate** プロパティを使用して、テンプレートセルを作成して削除ボタンを表示することができます。たとえば、次の例は、ボタンセルのテンプレート列を追加してボタンクリックで行の情報を削除する方法を示します。

【実行例】

マークアップ

```
<c1:C1FlexGrid x:Name="_flex" Grid.Row="1" AutoGenerateColumns="False"
Margin="5,5,5,5">
    <c1:C1FlexGrid.Columns>
        <c1:Column Header="Product" Binding="{Binding Line}"/>
        <c1:Column Header="Color" Binding="{Binding Color}"/>
        <c1:Column Header="Rating" Binding="{Binding Rating}"/>
        <c1:Column ColumnName="テンプレート">
            <c1:Column.CellTemplate>
                <DataTemplate>
                    <Button Content="削除" Click="Button_Click" />
                </DataTemplate>
            </c1:Column.CellTemplate>
        </c1:Column>
    </c1:C1FlexGrid.Columns>
</c1:C1FlexGrid>
```

VisualBasic

```
Private Sub Button_Click(sender As Object, e As RoutedEventArgs)
    Dim product = GetProduct(sender)
    If product IsNot Nothing Then
        Dim result = MessageBox.Show("このレコードを削除しますか？", "確認",
MessageBoxButton.OKCancel)
        If result = MessageBoxResult.OK Then
            Dim ev = TryCast(_flex.CollectionView,
System.ComponentModel.IEditableCollectionView)
            ev.Remove(product)
        End If
    End If
End Sub
'グリッド上にコントロールとして表示されるProductを取得します
Private Function GetProduct(control As Object) As Product
    Dim e As FrameworkElement = TryCast(control, FrameworkElement)
    Return TryCast(e.DataContext, Product)
End Function
```

C#

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var product = GetProduct(sender);
    if (product != null)
    {
        var result = MessageBox.Show("このレコードを削除しますか？", "確認",
MessageBoxButton.OKCancel);
        if (result == MessageBoxResult.OK)
        {
            var ev = _flex.CollectionView as
System.ComponentModel.IEditableCollectionView;
            ev.Remove(product);
        }
    }
}
//グリッド上にコントロールとして表示されるProductを取得します
Product GetProduct(object control)
{
    FrameworkElement e = control as FrameworkElement;
    return e.DataContext as Product;
}
```

グループ化

グループ化を実行する

グリッドにグループ化を実現してデータを簡単に階層化して表示することができます。たとえば、製品に含まれる **MainTestApplication** サンプルにあるように、顧客を都道府県とアクティブごとにグループ化するには、コードを次のようにす

るだけで済みます。

WPF

C#

```
List<Customer> list = GetCustomerList();
ListCollectionView view = new ListCollectionView(list);
using (view.DeferRefresh())
{
    view.GroupDescriptions.Clear();
    view.GroupDescriptions.Add(new PropertyGroupDescription("都道府県"));
    view.GroupDescriptions.Add(new PropertyGroupDescription("アクティブ"));
}
_flexGrid.ItemsSource = view;
```

"using (view.DeferRefresh())" 文はオプションです。この文を追加すると、すべてのグループが設定されるまでデータソースからの通知が保留になるため、パフォーマンスが向上します。

次の図に、この結果を示します。

データ項目は、都道府県、およびアクティブな状態ごとにグループ化されます。ユーザーは、グループヘッダのアイコンをクリックして、グループを折りたたんだり展開することができます（**TreeView** コントロールを使用する場合）。

グループ化をグリッドレベルで無効にする場合は、グリッドの **GroupRowPosition** プロパティを **GroupRowPosition.None** に設定します(AboveData または BelowData にも設定可能)

データのグループ化(ICollectionView を使用)

FlexGrid supports grouping through IDataCollection interface. This section discusses about grouping in FlexGrid .NET 4.5.2 and .NET 5 versions using DataCollection.

.NET Framework

FlexGrid は、**IDataCollection** インタフェースを使用したグループ化をサポートします。**PropertyGroupDescription** クラスを使用してグループ化のレベルをそれぞれ定義することで、FlexGrid で階層化ビューを作成できます。**PropertyGroupDescription** オブジェクトを使用すると、データをグループ化するためのプロパティを選択し、**ValueConverter** を実装してグループ化の際にプロパティ値を使用する方法を決定できます。グリッドの**GroupRowPosition** プロパティを None に設定すると、グリッドレベルでグループ化を無効にすることもできます。

次の図は、国とアクティブ状態に基づいてグループ化したデータを示します。ユーザーは、グループヘッダーのアイコンをクリックして、グループを折りたたんだり展開することができます（TreeView コントロールを使用する場合）。

次のコード例は、DataCollection を使用して国とアクティブ状態に基づいてグループ化されたデータを示します。また、このコードは製品サンプルのFlexGridSamplesを参考にしてください。

C#

```
List<Customer> list = GetCustomerList();
PagedCollectionView view = new PagedCollectionView(list);
using (view.DeferRefresh())
{
  view.GroupDescriptions.Clear();
  view.GroupDescriptions.Add(new PropertyGroupDescription("Country"));
  view.GroupDescriptions.Add(new PropertyGroupDescription("Active"));
}
grid.ItemsSource = view;
```

メモ: using (view.DeferRefresh()) 文はオプションです。この文を追加すると、すべてのグループが設定されるまでデータソースからの通知が保留になるため、パフォーマンスが向上します。

国のイニシャルに基づいてデータをグループ化するには

国の列でグループ化するだけでなく、FlexGrid ではエンドユーザーのさまざまな要件に応じてグループ化を実装できます。たとえば、データを国の列自体ではなく国のイニシャルでグループ化するという独自のシナリオを考えます。次の図は、国名全体ではなく、国名の最初の文字によって FlexGrid でグループ化されたデータを示しています。

コードでこのシナリオを実行するには、カスタムクラス **CountryInitialConverter** を作成する必要があります。このクラスは、IValueConverter インタフェースを実装し、グループ化に使用するために国名全体ではなく国名の最初の文字を返します。さらに、以下のコード例で示すように、MainWindow.xaml.cs でコンバータを設定する必要があります。

```
class CountryInitialConverter : IValueConverter
{
  public object Convert(object value, Type targetType,
    object parameter,
    System.Globalization.CultureInfo culture)
  {
    return ((string)value)[0].ToString().ToUpper();
  }
  public object ConvertBack(object value, Type targetType,
    object parameter,
    System.Globalization.CultureInfo culture)
  {
    throw new NotImplementedException();
  }
}

List<Customer> list = GetCustomerList();
PagedCollectionView view = new PagedCollectionView(list);
using (view.DeferRefresh())
{
  view.GroupDescriptions.Clear();
  view.GroupDescriptions.Add(new PropertyGroupDescription("Country"));
  view.GroupDescriptions.Add(new PropertyGroupDescription("Active"));
  var gd = view.GroupDescriptions[0] as PropertyGroupDescription;
  gd.Converter = new CountryInitialConverter();
}
grid.ItemsSource = view;
```

グループ行には、その行に対応するグループに関する情報が表示されます。具体的には、グループ化に使用されたプロパティや値、項目数などです。この情報をカスタマイズするには、新しい **IValueConverter** クラスを作成し、グリッドの **GroupHeaderConverter** プロパティに割り当てます。たとえば、デフォルトのグループヘッダーコンバータは、次のコードで実装されます。

C#

```
// グループキャプションの表示形式を設定するクラス
public class GroupHeaderConverter : IValueConverter
{
```

```
  public object Convert(object value,
    Type targetType, object parameter,
    System.Globalization.CultureInfo culture)
  {
    var gr = parameter as GroupRow;
    var group = gr.Group;
    if (group != null && gr != null && targetType == typeof(string))
    {
      var desc = gr.Grid.View.GroupDescriptions[gr.Level] as
                 PropertyGroupDescription;
      return desc != null
        ? string.Format("{0}: {1} ({2:n0} items)",
                 desc.PropertyName, group.Name, group.ItemCount)
        : string.Format("{0} ({1:n0} items)",
                 group.Name, group.ItemCount);
    }
    return value;
  }
  public object ConvertBack(object value, Type targetType,
    object parameter,
    System.Globalization.CultureInfo culture)
  {
    return value;
  }
}
```

.NET

FlexGrid は、**IDataCollection** インタフェースを使用したグループ化をサポートします。**PropertyGroupDescription** クラスを使用してグループ化のレベルをそれぞれ定義することで、FlexGrid で階層化ビューを作成できます。**PropertyGroupDescription** オブジェクトを使用すると、データをグループ化するためのプロパティを選択し、**ValueConverter** を実装してグループ化の際にプロパティ値を使用する方法を決定できます。グリッドの**GroupRowPosition** プロパティを **None** に設定すると、グリッドレベルでグループ化を無効にすることもできます

次の図は、国とアクティブ状態に基づいてグループ化したデータを示します。ユーザーは、グループヘッダーのアイコンをクリックして、グループを折りたたんだり展開することができます（TreeView コントロールを使用する場合）。

Country	Active	ID	Name	Country ID	First Name	Last Name
Country: China (13 items)						
Active: False (13 items)						
China	☐	0	Alaric Rodriguez	0	Alaric	Rodriguez
China	☐	4	Gina Frommer	0	Gina	Frommer
China	☐	5	Ed Salvatore	0	Ed	Salvatore
China	☐	14	Herb Rodriguez	0	Herb	Rodriguez
China	☐	18	Ed Bishop	0	Ed	Bishop
China	☐	20	Ben Saltzman	0	Ben	Saltzman
China	☐	25	Fred Frommer	0	Fred	Frommer
China	☐	26	Fred Frommer	0	Fred	Frommer
China	☐	27	Herb Ambers	0	Herb	Ambers
China	☐	31	Charlie Evers	0	Charlie	Evers
China	☐	32	Herb Danson	0	Herb	Danson
China	☐	37	Alaric Bishop	0	Alaric	Bishop

次のコード例は、DataCollection を使用して国とアクティブ状態に基づいてグループ化されたデータを示します。また、このコードは製品サンプルのFlexGridExplorerを参考にしてください。

C#

```
// Generate FlexGrid data
var data = new ObservableCollection<Customer>();
for (int i = 0; i < 200; i++)
{
    data.Add(new Customer(i));
}
var view = new MyCollectionView(data);
using (view.DeferRefresh())
{
    view.GroupDescriptions.Clear();
    view.GroupDescriptions.Add(new PropertyGroupDescription("Country"));
    view.GroupDescriptions.Add(new PropertyGroupDescription("Active"));
```

```
    var gd = view.GroupDescriptions[0] as PropertyGroupDescription;
}

// bind grids to ListCollectionView
grid.ItemsSource = view;
```

メモ: using (view.DeferRefresh()) 文はオプションです。この文を追加すると、すべてのグループが設定されるまでデータソースからの通知が保留になるため、パフォーマンスが向上します。

国のイニシャルに基づいてデータをグループ化するには

国の列でグループ化するだけでなく、FlexGrid ではエンドユーザーのさまざまな要件に応じてグループ化を実装できます。たとえば、データを国の列自体ではなく国のイニシャルでグループ化するという独自のシナリオを考えます。次の図は、国名全体ではなく、国名の最初の文字によって FlexGrid でグループ化されたデータを示しています。

Country	Active	ID	Name	CountryID	First
Country: Bangladesh (7 items)					
Country: Brazil (8 items)					
Country: China (8 items)					
Country: Congo (9 items)					
Country: Egypt (6 items)					
Active: True (4 items)					
Egypt	☑	84	Zeb Stevens	15	Zeb
Egypt	☑	41	Herb Neiman	15	Herb
Egypt	☑	86	Zeb Frommer	15	Zeb
Egypt	☑	143	Noah Myers	15	Noah

コードでこのシナリオを実行するには、カスタムクラス **CountryInitialGrouping** を作成する必要があります。このクラスは、**IValueConverter** インタフェースを実装し、グループ化に使用するために国名全体ではなく国名の最初の文字を返します。さらに、以下のコード例で示すように、MainWindow.xaml.cs でコンバータを設定する必要があります。

```
<code>public void CountryInitialGrouping()
{
    // FlexGridデータを生成します
    var data = new ObservableCollection<Customer>();
    for (int i = 0; i < 200; i++)
    {
        data.Add(new Customer(i));
    }
    var view = new MyCollectionView(data);
    using (view.DeferRefresh())
    {
        view.GroupDescriptions.Clear();
        view.GroupDescriptions.Add(new PropertyGroupDescription("Country"));
        view.GroupDescriptions.Add(new PropertyGroupDescription("Active"));
        var gd = view.GroupDescriptions[0] as PropertyGroupDescription;
        gd.Converter = new CountryInitialConverter();
    }

    //グリッドをListCollectionViewにバインドします
    grid.ItemsSource = view;
```

グループ行には、その行に対応するグループに関する情報が表示されます。具体的には、グループ化に使用されたプロパティや値、項目数などです。この情報をカスタマイズするには、新しい **IValueConverter** クラスを作成し、グリッドの **GroupHeaderConverter** プロパティに割り当てます。たとえば、デフォルトのグループヘッダーコンバータは、次のコードで実装されます。

C#

```
// グループキャプションの表示形式を設定するクラス
public class GroupHeaderConverter : IValueConverter
{
  public object Convert(object value,
    Type targetType, object parameter,
    System.Globalization.CultureInfo culture)
  {
    var gr = parameter as GroupRow;
    var group = gr.Group;
    if (group != null && gr != null && targetType == typeof(string))
    {
      var desc = gr.Grid.View.GroupDescriptions[gr.Level] as
                 PropertyGroupDescription;
      return desc != null
        string.Format("{0}: {1} ({2:n0} items)",
                 desc.PropertyName, group.Name, group.ItemCount)
        : string.Format("{0} ({1:n0} items)",
                 group.Name, group.ItemCount);
    }
    return value;
  }
  public object ConvertBack(object value, Type targetType,
    object parameter,
    System.Globalization.CultureInfo culture)
  {
    return value;
  }
}
```

C1FlexGridGroupPanel を使用してグループ領域を表示する

C1FlexGridGroupPanel は、**C1FlexGrid** コントロールでグループ化を管理できるため UI を提供する新しいコントロールです。

C1FlexGridGroupPanel コントロールは現在、別なアセンブリ (C1.WPF.FlexGrid.GroupPanel.4.dll)として実装されています。

カスタムのグループコンバータ

FlexGrid は、列を明示したデータのグループ化をサポートするほか、カスタムグループコンバータを作成し、カテゴリに基づいてデータをグループ化することもサポートします。たとえば、任意の注文日や価格でデータをグループ化したり、その両方を混在させることもできます。しかも、列の価格をさらに高、中、低などのサブグループに区分できれば、データの組織化がさらに意味のあるものになります。FlexGridGroupPanel コントロールを使用すると、サブグループを作成したり範囲を定義して、このようなグループ化要件を実現できます。

次の図に、価格範囲に基づいたカスタムデータグループ化を示します。

コードでカスタムグループコンバータを作成するには

1. Visual Studio で WPF アプリケーションを作成します。
2. XAML デザイナに FlexGrid コントロールをドラッグします。
3. 次の XAML コードを使用して、FlexGrid コントロールに FlexGridGroupPanel を追加します。

XAML

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition />
    </Grid.RowDefinitions>
          <c1:C1FlexGridGroupPanel
```

```
            x:Name="_groupPanelCustomGrouping" Grid.Row="0"
Background="WhiteSmoke"
            WatermarkText="Drag column headers here to create groups."
            MaxGroups="8"
            HideGroupedColumns="False"
            DragMarkerColor="#FF5C54"
            FlexGrid="{Binding ElementName=_flexCustomGrouping}"

PropertyGroupCreated="_groupPanelCustomGrouping_PropertyGroupCreated"/>
         <c1:C1FlexGrid
            x:Name="_flexCustomGrouping" Grid.Row="1"
            TopLeftCellBackground="Bisque"
            ColumnHeaderBackground="Bisque"
            RowHeaderBackground="Bisque"
            GroupRowBackground="LightGoldenrodYellow"
            RowBackground="White"
            AlternatingRowBackground="White"/>
</Grid>
```

4. プロジェクト Products.cs に新しいクラスを追加し、グリッドにデータを表示します。

C#

```
public class Product : BaseObject
{
    string _name, _color, _line;
    double _price, _cost;
    DateTime _date;
    bool _discontinued;

    static Random _rnd = new Random();
    static string[] _names = "Macko|Surfair|Pocohey|Studeby".Split('|');
    static string[] _lines = "コンピュータ|ワッシャ|ストーブ|車".Split('|');
    static string[] _colors = "赤|緑|青|白".Split('|');

    public Product()
    {
        名前 = PickOne(_names);
        製品 = PickOne(_lines);
        色 = PickOne(_colors);
        価格 = 30 + _rnd.NextDouble() * 1000;
        コスト = 3 + _rnd.NextDouble() * 300;
        廃止 = _rnd.NextDouble() < .2;
        発売日 = DateTime.Today.AddDays(_rnd.Next(-600, 0));
    }
    string PickOne(string[] options)
    {
        return options[_rnd.Next() % options.Length];
    }

    //[Display(Name = "Name")]
    public string 名前
    {
```

```
        get { return _name; }
        set { SetProperty("Name", ref _name, value); }
    }

    //[Display(Name = "Color")]
    public string 色
    {
        get { return _color; }
        set { SetProperty("Color", ref _color, value); }
    }

    //[Display(Name = "Line")]
    public string 製品
    {
        get { return _line; }
        set { SetProperty("Line", ref _line, value); }
    }

    //[Display(Name = "Price")]
    public double 価格
    {
        get { return _price; }
        set { SetProperty("Price", ref _price, value); }
    }

    //[Display(Name = "Cost")]
    public double コスト
    {
        get { return _cost; }
        set { SetProperty("Cost", ref _cost, value); }
    }

    //[Display(Name = "Introduced")]
    public DateTime 発売日
    {
        get { return _date; }
        set { SetProperty("Introduced", ref _date, value); }
    }

    //[Display(Name = "Discontinued")]
    public bool 廃止
    {
        get { return _discontinued; }
        set { SetProperty("Discontinued", ref _discontinued, value); }
    }
}

public class BaseObject : INotifyPropertyChanged, IEditableObject
{
    protected bool SetProperty<T>(string propName, ref T field, T value)
    {
        if (EqualityComparer<T>.Default.Equals(field, value)) return false;
        field = value;
```

```
            OnPropertyChanged(propName);
            return true;
        }

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

        object _clone = null;
        public void BeginEdit()
        {
            _clone = this.MemberwiseClone();
        }
        public void CancelEdit()
        {
            foreach (var p in this.GetType().GetProperties())
            {
                var value = p.GetValue(_clone, null);
                p.SetValue(this, value, null);
            }
        }
        public void EndEdit()
        {
            _clone = null;
        }
}
```

5. プロジェクトに 2 つのクラス AmountGroupConverter.cs と DateTimeGroupConverter.cs を追加します。これらのクラスは、**Price** および **Cost** 列の double 値と **Introduced** および **Discontinued** 列の DateTime 値を適切に定義された範囲にグループ化するシンプルなコンバータです。

AmountGroupConverter

```
public class AmountGroupConverter : IValueConverter
{
    double _maxValue;
    public AmountGroupConverter(double maxValue)
    {
        _maxValue = maxValue;
    }
    public object Convert(object value, Type targetType,
        object parameter, System.Globalization.CultureInfo culture)
    {
        var pct = (double)value / _maxValue;
        if (pct < .25) return "Very Low";
        if (pct < .50) return "Low";
        if (pct < .75) return "Moderate";
        return "High";
    }
    public object ConvertBack(object value, Type targetType,
        object parameter, System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

DateTimeGroupConverter

```
public class DateTimeGroupConverter : IValueConverter
{
    public object Convert(object value, Type targetType,
        object parameter, System.Globalization.CultureInfo culture)
    {
        var list = new List<string>();
        var date = (DateTime)value;
        var today = DateTime.Today;
        if (today.Subtract(date).TotalDays <= 7) list.Add("This week");
        if (date.Year == today.Year && date.Month == today.Month)
list.Add("This month");
        if (date.Year == today.Year)
        {
            list.Add("This year");
        }
        else if (date.Year == today.Year - 1)
        {
            list.Add("Last year");
        }
        else
        {
            list.Add("Before last year");
        }
        return list;
    }
    public object ConvertBack(object value, Type targetType,
        object parameter, System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

6. MainWindow.cs ファイルに切り替え、次のコードを追加して FlexGrid にデータを追加します。

C#

```
// データソースを作成します
var list = new List<Product>();
for (int i = 0; i < 200; i++)
{
    list.Add(new Product());
}
// グリッドにデータソースを割り当てます
var cvs = new CollectionViewSource() { Source = list };
_flexCustomGrouping.ItemsSource = cvs.View;
```

7. PropertyGroupCreated イベントのハンドラに次のコードを追加して、データソースのいくつかの列にカスタムコンバータを割り当てます。

C#

```
private void _groupPanelCustomGrouping_PropertyGroupCreated
    (object sender, C1.WPF.FlexGrid.PropertyGroupCreatedEventArgs e)
{
    var pgd = e.PropertyGroupDescription;
    switch (pgd.PropertyName)
    {
        case "発売日":
            pgd.Converter = new DateTimeGroupConverter();
            break;
        case "廃止":
            pgd.Converter = new DateTimeGroupConverter();
            break;
        case "価格":
            pgd.Converter = new AmountGroupConverter(1000);
            break;
        case "コスト":
            pgd.Converter = new AmountGroupConverter(300);
            break;
    }
}
```

FlexGridGroupPanelの操作

FlexGridGroupPanel は、テキストブロック付きのグリッド要素とスタックパネルで構成されるカスタムコントロールです。テキストブロックにはウォーターマークメッセージが表示され、スタックパネルにはソース **ICollectionView** にあるグループが表示されます。グループは GroupMarker 要素で表され、これをクリックしてグループをソートまたは閉じたり、ドラッグしてグループ化の順序を変更することができます。このコントロールは、**DragDropManager** ユーティリティクラスに付属し、次の 4 つのドラッグアクションを処理します。

- グループを再配置するには、グループ領域内で GroupMarker をドラッグします。
- グループを削除して、指定された位置に列を復元するには、GroupMarker をグリッドにドラッグします。
- 新しいグループを作成するには、ColumnHeader 要素をグリッドからグループ化領域にドラッグします。
- グリッド列を再配置するには、ColumnHeader 要素をグリッド内でドラッグします。

最初の 2 つのドラッグアクションは、**GroupMarker** クラスによって開始されます。このクラスは、マウスのドラッグアクションを検出し、マーカーをパラメータとして **DoDragDrop** メソッドを呼び出します。最後の 2 つのアクションは、FlexGrid の **DraggingColumn** イベントに応答して開始されます。**DoDragDrop** メソッドが呼び出されると、DragDropManager は、透明

な要素をページ全体に表示し、マウスをキャプチャし、呼び出し元がドロップ位置を更新できるように Dragging イベントを発生させます。ユーザーがマウスを放すと、**DragDropManager** は、呼び出し元がドラッグドロップアクションを完了できるように **Dropped** イベントを発生させます。

> **メモ**: グループ化機能は、**C1.WPF.FlexGridGroupPanel**拡張アセンブリとして実装されます。これは、次の目的で独立して提供されています。
>
> - FlexGrid のフットプリントを最小限に抑え、コンパクトさと高速性を保ちます。
> - ユーザーが独自にカスタムバージョンを作成できるように、グループ化 UI でカスタマイズを可能にします。
> - FlexGrid コントロールの柔軟性と拡張性を提供します。

グループ集計

FlexGrid lets you compute and display aggregate value for each group created after grouping data. The control shows groups in a collapsible outline format and automatically displays the number of items in each group. However, you can go one step further and display aggregate values for every grouped column. This feature enables users in drawing out more insights from random data.

FlexGrid では、データをグループ化した後に、作成された各グループの集計値を計算して表示できます。コントロールは、折りたたみ可能なアウトライン形式でグループを表示し、各グループの項目数を自動的に表示します。ただし、さらに詳しくグループ化列の集計値を表示することもできます。この機能を使用して、ランダムなデータからさらに深い洞察を得ることができます。

次のセクションでは、FlexGrid .NET 4.5.2および.NET 5バージョンでデータ集約を実装する方法を説明します。

.NET Framework

たとえば、国または製品カテゴリに基づいてグループ化された企業の売上データには、グリッド自体に各国および製品カテゴリごとの売上集計値が示されるとさらに便利です。**Column** クラスには **GroupAggregate** プロパティがあり、これを **Aggregate** に設定すると、自動的に集計値が計算および表示されます。**GroupAggregate** プロパティを設定して計算された集計は、データが変更されると自動的に集計値が再計算されます。

次の図に、列に集計値が表示された FlexGrid を示します。

Line	Color	Name	Price	Cost	Weight	Volume
Total: (200 items)			103,610.00	61,744.00	10,089.00	573,086.00
Line: Washers (55 items)			27,656.00	16,865.00	2,697.00	144,544.00
Color: Green (16 items)			8,150.00	4,658.00	669.00	33,853.00
Price: Medium (1 items)			68.00	233.00	3.00	1,171.00
Washers	Green	P 0	68.00	233.00	3.00	1,171.00
Price: Very High (8 items)			5,994.00	1,797.00	402.00	18,476.00
Washers	Green	P 2	678.00	201.00	12.00	2,748.00
Washers	Green	P 23	840.00	283.00	60.00	2,077.00
Washers	Green	P 42	687.00	107.00	59.00	1,856.00
Washers	Green	P 88	747.00	408.00	88.00	899.00
Washers	Green	P 91	630.00	249.00	82.00	3,505.00
Washers	Green	P 132	658.00	15.00	13.00	3,857.00
Washers	Green	P 187	889.00	349.00	68.00	2,952.00
Washers	Green	P 194	865.00	185.00	20.00	582.00
Price: High (6 items)			2,084.00	2,476.00	193.00	12,436.00
Washers	Green	P 67	487.00	521.00	65.00	2,196.00
Washers	Green	P 75	257.00	350.00	19.00	526.00

コードから集計を設定するには

グリッドの **AreGroupHeadersFrozen** プロパティを **false** に設定し、各列の **GroupAggregate** プロパティをサポートされる集計値のいずれかに設定すると、FlexGrid の各グループに集計値を表示できます。サポートされている集計値には、**合計**、**平均**、**カウント**、**最小**、**最大**などがあります。

> **メモ**: 集計はグループヘッダー列に表示されるため、**AreGroupHeadersFrozen** プロパティを **false** に設定してヘッダーを表示する必要があります。

次のコードは、Price、Cost、Weight、Volume の各列の合計を XAML から表示します。

XAML

```
<fg:C1FlexGrid x:Name="_flex" AutoGenerateColumns="False"
  AreRowGroupHeadersFrozen="False">
  <fg:C1FlexGrid.Columns>
    <fg:Column Header="Line" Binding="{Binding Line}" />
    <fg:Column Header="Color" Binding="{Binding Color}" />
    <fg:Column Header="Name" Binding="{Binding Name}" />
    <fg:Column Header="Price" Binding="{Binding Price}"
      Format="n2" HorizontalAlignment="Right" Width="*"
      GroupAggregate="Sum"/>
    <fg:Column Header="Cost" Binding="{Binding Cost}"
      Format="n2" HorizontalAlignment="Right" Width="*"
      GroupAggregate="Sum"/>
```

```
    <fg:Column Header="Weight" Binding="{Binding Weight}"
      Format="n2" HorizontalAlignment="Right" Width="*"
      GroupAggregate="Sum"/>
    <fg:Column Header="Volume" Binding="{Binding Volume}"
      Format="n2" HorizontalAlignment="Right" Width="*"
      GroupAggregate="Sum"/>
  </fg:C1FlexGrid.Columns>
</fg:C1FlexGrid>
```

たとえば、国または製品カテゴリに基づいてグループ化された企業の売上データには、グリッド自体に各国および製品カテゴリごとの売上集計値が示されるとさらに便利です。**GridColumn** クラスには **Aggregate** プロパティがあり、これを **Aggregate** に設定すると、自動的に集計値が計算および表示されます。GroupAggregate プロパティを設定して計算された集計は、データが変更されると自動的に集計値が再計算されます。

次の図に、列に集計値が表示された FlexGrid を示します。

Id	Active	Name	Country	Weight
Country: France (11 items)	11	11		834.4197641519922
Active: True (6 items)	6	6		461.4267032413867
Active: False (5 items)	5	5		372.9930609106055
97	☐	Fred Danson	France	62.093174067369276
100	☐	Herb Bishop	France	74.73954235889927
128	☐	Ed Cole	France	98.05842607657351
136	☐	Larry Heath	France	52.77836928273475
198	☐	Ed Griswold	France	85.32354912502856
Country: United Kingdom (6 i	6	6		510.8772392668190
Active: True (5 items)	5	5		422.8746191239332

コードから集計を設定するには

FlexGridの各グループの集計値を表示するには、各列のAggregateプロパティをサポートされている集計値の1つに設定します。サポートされている集計値には、**合計**、**平均**、**カウント**、**最小**、**最大**などがあります。

次のコードは、XAMLにて[Active]列と[Name]列の[カウント]、および[Weight]の合計を表示しています。

XAML

```
<c1:FlexGrid Name='grid' AutoGenerateColumns="False" Style="{StaticResource
excelBlue}"
MinColumnWidth="10" MaxColumnWidth="300" HorizontalAlignment="Center"
GroupHeaderFormat="{} {name}:
{value} ({count:n0} items)" HeadersVisibility="All" Height="700" Width="1000">
   <c1:FlexGrid.Columns>
      <c1:GridColumn Binding="Id" Width="200"/>
      <c1:GridColumn Binding="Active" HorizontalAlignment="Right"
Aggregate="Count"/>
      <c1:GridColumn Binding="Name" HorizontalAlignment="Right" Aggregate="Count"/>
      <c1:GridColumn Binding="Country" Width="200" HorizontalAlignment ="Center" />
      <c1:GridColumn Binding="Weight" Width="200" HorizontalAlignment ="Center"
Aggregate="Sum"/>
   </c1:FlexGrid.Columns>
```

```
</c1:FlexGrid>
```

マージ

セルのマージ

FlexGrid は、セル結合をサポートし、複数の行および列にまたがってデータを表示することができます。セル結合機能を使用して、データの外観を向上させることができます。

次のセクションでは、FlexGrid .NET 4.5.2 および.NET 5 バージョンでセルのマージを実装する方法を説明します。

.NET Framework

セル結合は、コードで **C1FlexGrid** の**AllowMerging** プロパティを設定して有効にできます。列を結合するには、結合する各列の AllowMerging プロパティを true に設定します。

同様に、行を結合するには、結合する各行で AllowMerging プロパティを **true** に設定します。**AllowMerging 列挙**を使用して、マージするグリッドのCells、ColumnHeadersなどの領域を指定できます。

これで、次のようにグリッドが作成されます。

たとえば、次のコードは、都道府県と名を含むセルを結合します。

C#

```
// スクロール可能な領域での結合を有効にします
grid.AllowMerging = AllowMerging.Cells;
// "都道府県" 列と "名" 列
grid.Columns["Country"].AllowMerging = true;
grid.Columns["FirstName"].AllowMerging = true;
```

.NET

セル結合は、コードで **GridBase** の**AllowMerging** プロパティを設定して有効にできます。列を結合するには、結合する各列の AllowMerging プロパティを true に設定します。

同様に、行を結合するには、結合する各行で AllowMerging プロパティを **true** に設定します。**GridAllowMerging 列挙**を使用して、マージするグリッドのCells、ColumnHeadersなどの領域を指定できます。

これで、次のようにグリッドが作成されます。

Country	Active	ID	Name	Country ID	First Name	Last Name
Country: India (7 items)						
Active: False (7 items)						
India	☐	3	Fred Ambers	1	Fred	Ambers
	☐	11	Fred Evers	1		Evers
	☐	16	Charlie Frommer	1	Charlie	Frommer
	☐	25	Jim Salvatore	1	Jim	Salvatore
	☐	31	Gina Danson	1	Gina	Danson
	☐	35	Dan Saltzman	1	Dan	Saltzman
	☐	44	Elena Evers	1	Elena	Evers
Country: China (17 items)						
Active: False (17 items)						
	☐	6	Stefan Frommer	0	Stefan	Frommer
	☐	9	Gil Evers	0	Gil	Evers
	☐	12	Dan Ambers	0	Dan	Ambers

たとえば、次のコードは、同じ都道府県を含むセルを結合します。

C#

```
//スクロール可能な領域での結合を有効にします
grid.AllowMerging = GridAllowMerging.Cells;
//"Country" 列と "FirstName"列にマージそ行います
grid.Columns["Country"].AllowMerging = true;
grid.Columns["FirstName"].AllowMerging = true;
```

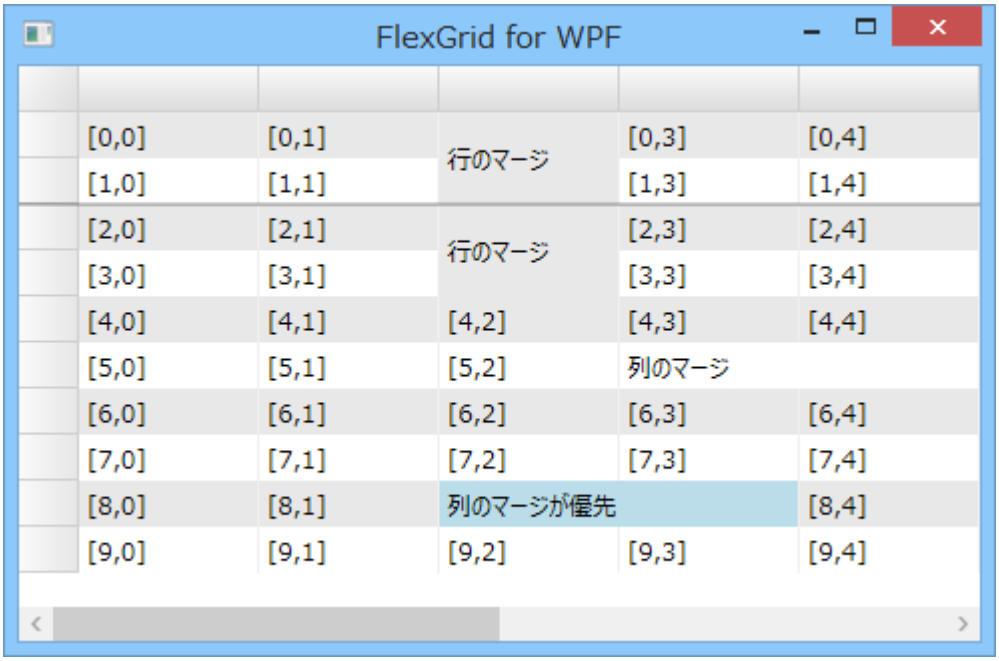
同じ値を持つセルをマージする

グリッドの**AllowMerging**プロパティを**C1.WPF.FlexGrid.AllowMerging.Cells** に設定して、特定の行および列の**AllowMerging**プロパティを**True**に設定することで、固定セル・通常セルをマージして表示できます。

たとえば、次の例では、列2で2行目、5行目と8行目で同じ値を持つセルをマージして表示します。また、ここで行8と列8のマー

ジが同時に行われたので、列のマージが優先されることに注目してください。

【実行例】



サンプルコードは次のようになります。

VisualBasic

```
_flex.Rows.Frozen = 2
_flex.AllowMerging = C1.WPF.FlexGrid.AllowMerging.Cells
_flex.Columns(2).AllowMerging = True
_flex.Rows(5).AllowMerging = True
_flex.Rows(8).AllowMerging = True
' データを設定します
_flex(0, 2) = "行のマージ"
_flex(1, 2) = "行のマージ"
_flex(2, 2) = "行のマージ"
_flex(3, 2) = "行のマージ"
_flex(5, 3) = "列のマージ"
_flex(5, 4) = "列のマージ"
_flex(8, 2) = "列のマージが優先"
_flex(8, 3) = "列のマージが優先"
```

C#

```
_flex.Rows.Frozen = 2;
_flex.AllowMerging = C1.WPF.FlexGrid.AllowMerging.Cells;
_flex.Columns[2].AllowMerging = true;
_flex.Rows[5].AllowMerging = true;
_flex.Rows[8].AllowMerging = true;
// データを設定します
_flex[0, 2] = "行のマージ";
_flex[1, 2] = "行のマージ";
_flex[2, 2] = "行のマージ";
_flex[3, 2] = "行のマージ";
_flex[5, 3] = "列のマージ";
_flex[5, 4] = "列のマージ";
_flex[8, 2] = "列のマージが優先";
_flex[8, 3] = "列のマージが優先";
```

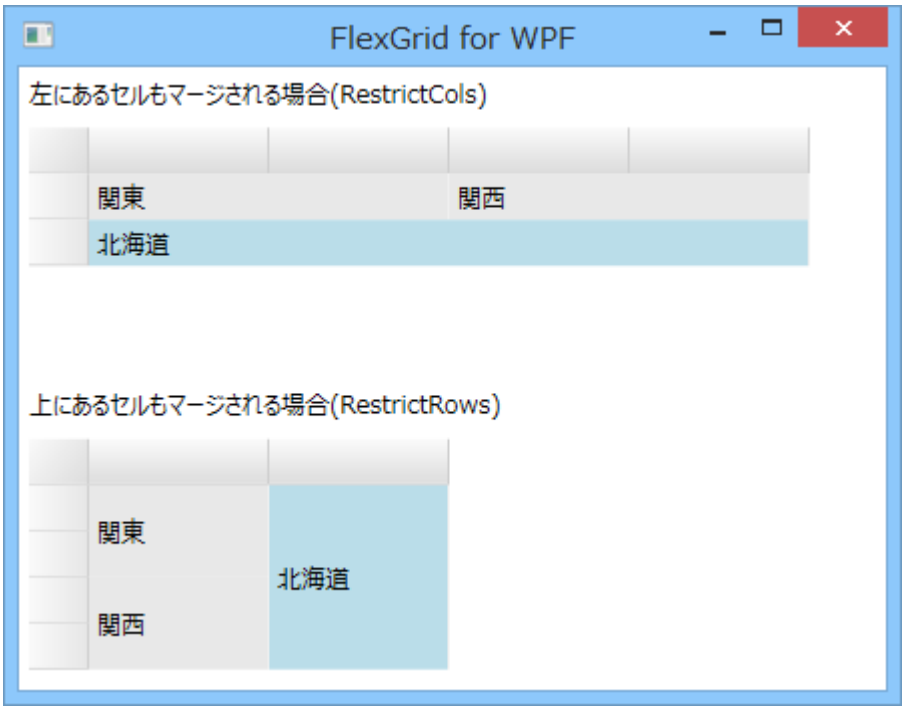
注意:

- 固定セルと通常セルはマージされません。
- 自動マージの場合、特定の位置でのみマージする(同一データをマージしない)といった動作はできません。この場合は、自動マージのカスタマイズや制限付きマージ、カスタムマージを検討してください。
- 行と列のマージが同時に行われた場合、列のマージが優先されます。
- カスタムマージと自動マージを同時に実装することはできません。

同じ値を持つセルをマージする(制限付き)

グリッドの**AllowMerging**プロパティを**All**に設定して、**IMergeManager**インタフェースを実現することでマージ動作をカスタマイズすることが用意されています。たとえば、セルの上にあるセルもマージされる場合の動作は以下のRestrictRowsメソッドのように実現できます。それと同じように、左にあるセルもマージする場合、以下のRestrictColsメソッドのようにに実現できます。

【実行例】



サンプルコードは次のようになります。

VisualBasic

```
'左にあるセルもマージする場合
Private Sub InitializeRestrictCols()
    For i As Integer = 0 To 1
        _restrictColsflex.Columns.Add(New C1.WPF.FlexGrid.Column())
    Next
    For i As Integer = 0 To 3
        _restrictColsflex.Rows.Add(New C1.WPF.FlexGrid.Row())
    Next
    ' データを設定する場合
    _restrictColsflex(0, 0) = "関東"
    _restrictColsflex(1, 0) = "関東"
    _restrictColsflex(2, 0) = "関西"
    _restrictColsflex(3, 0) = "関西"
    _restrictColsflex(0, 1) = "北海道"
    _restrictColsflex(1, 1) = "北海道"
    _restrictColsflex(2, 1) = "北海道"
    _restrictColsflex(3, 1) = "北海道"

    _restrictColsflex.AllowMerging = AllowMerging.All
    _restrictColsflex.MergeManager = New RestrictColsMergeManager()

End Sub

'上にあるセルもマージする場合
Private Sub InitializeRestrictRows()
    For i As Integer = 0 To 3
        _restrictRowsflex.Columns.Add(New C1.WPF.FlexGrid.Column())
    Next
    For i As Integer = 0 To 1
        _restrictRowsflex.Rows.Add(New C1.WPF.FlexGrid.Row())
    Next
    _restrictRowsflex.AllowMerging = AllowMerging.All
    _restrictRowsflex.MergeManager = New RestrictRowsMergeManager()
    _restrictRowsflex(0, 0) = "関東"
    _restrictRowsflex(0, 1) = "関東"
    _restrictRowsflex(0, 2) = "関西"
    _restrictRowsflex(0, 3) = "関西"
    _restrictRowsflex(1, 0) = "北海道"
    _restrictRowsflex(1, 1) = "北海道"
    _restrictRowsflex(1, 2) = "北海道"
    _restrictRowsflex(1, 3) = "北海道"
End Sub
```

C#

```
//左にあるセルもマージする場合
private void InitializeRestrictCols()
{
    for (int i = 0; i < 2; i++)
    {
        _restrictColsflex.Columns.Add(new C1.WPF.FlexGrid.Column());
    }
    for (int i = 0; i < 4; i++)
    {
        _restrictColsflex.Rows.Add(new C1.WPF.FlexGrid.Row());
    }
    // データを設定する場合
    _restrictColsflex[0, 0] = "関東";
    _restrictColsflex[1, 0] = "関東";
    _restrictColsflex[2, 0] = "関西";
    _restrictColsflex[3, 0] = "関西";
    _restrictColsflex[0, 1] = "北海道";
    _restrictColsflex[1, 1] = "北海道";
    _restrictColsflex[2, 1] = "北海道";
    _restrictColsflex[3, 1] = "北海道";

    _restrictColsflex.AllowMerging = AllowMerging.All;
    _restrictColsflex.MergeManager = new RestrictColsMergeManager();

}

//上にあるセルもマージする場合
private void InitializeRestrictRows()
{
    for (int i = 0; i < 4; i++)
    {
        _restrictRowsflex.Columns.Add(new C1.WPF.FlexGrid.Column());
    }
    for (int i = 0; i < 2; i++)
    {
        _restrictRowsflex.Rows.Add(new C1.WPF.FlexGrid.Row());
    }
    _restrictRowsflex.AllowMerging = AllowMerging.All;
    _restrictRowsflex.MergeManager = new RestrictRowsMergeManager();
    _restrictRowsflex[0, 0] = "関東";
    _restrictRowsflex[0, 1] = "関東";
    _restrictRowsflex[0, 2] = "関西";
    _restrictRowsflex[0, 3] = "関西";
    _restrictRowsflex[1, 0] = "北海道";
    _restrictRowsflex[1, 1] = "北海道";
    _restrictRowsflex[1, 2] = "北海道";
    _restrictRowsflex[1, 3] = "北海道";
}
```

IMergeManager インターフェイスを実装するクラスは、次のように作成します。

VisualBasic

```
Class RestrictRowsMergeManager
    Implements C1.WPF.FlexGrid.IMergeManager

    Public Function GetMergedRange(grid As C1FlexGrid, cellType__1 As CellType, rg
As CellRange) As CellRange Implements IMergeManager.GetMergedRange
        ' データセルの結合を対象します
        ' （行または列ヘッダを結合しない場合）
        If cellType__1 = CellType.Cell Then
            Dim isMerged As Boolean = False
            If grid.Tag Is Nothing Then
                grid.Tag = New List(Of Boolean)()
            End If
            Dim list = TryCast(grid.Tag, List(Of Boolean))
            If rg.Row = 0 OrElse (rg.Row > 0 AndAlso list(rg.Row - 1)) Then
                ' 左右に拡張します
                For i As Integer = rg.Column To grid.Columns.Count - 2
                    If GetDataDisplay(grid, rg.Row, i) <> GetDataDisplay(grid,
rg.Row, i + 1) Then
                        Exit For
                    End If
                    rg.Column2 = i + 1
                    isMerged = True
                Next
                For i As Integer = rg.Column To 1 Step -1
                    If GetDataDisplay(grid, rg.Row, i) <> GetDataDisplay(grid,
rg.Row, i - 1) Then
                        Exit For
                    End If
                    rg.Column = i - 1
                    isMerged = True
                Next
            End If

            list.Add(isMerged)
        End If

        ' 完了
        Return rg
    End Function
    Private Function GetDataDisplay(grid As C1FlexGrid, r As Integer, c As Integer)
As String
        Return grid(r, c).ToString()
    End Function
```

C#

```
class RestrictRowsMergeManager : IMergeManager
{
    public CellRange GetMergedRange(C1FlexGrid grid, CellType cellType, CellRange
rg)
    {
        // データセルの結合を対象します
        // （行または列ヘッダを結合しない場合）
        if (cellType == CellType.Cell)
        {
            bool isMerged = false;
            if (grid.Tag == null)
                grid.Tag = new List<bool>();
            var list = grid.Tag as List<bool>;
            if (rg.Row == 0 || (rg.Row > 0 && list[rg.Row - 1]))
            {
                // 左右に拡張します
                for (int i = rg.Column; i < grid.Columns.Count - 1; i++)
                {
                    if (GetDataDisplay(grid, rg.Row, i) != GetDataDisplay(grid,
rg.Row, i + 1)) break;
                    rg.Column2 = i + 1;
                    isMerged = true;
                }
                for (int i = rg.Column; i > 0; i--)
                {
                    if (GetDataDisplay(grid, rg.Row, i) != GetDataDisplay(grid,
rg.Row, i - 1)) break;
                    rg.Column = i - 1;
                    isMerged = true;
                }
            }

            list.Add(isMerged);
        }

        // 完了
        return rg;
    }
    string GetDataDisplay(C1FlexGrid grid, int r, int c)
    {
        return grid[r, c].ToString();
    }
}
```

注意:

- 固定セルと通常セルはマージされません。
- 行と列のマージが同時に行われた場合、列のマージが優先されます。
- カスタムマージと自動マージを同時に実装することはできません。

任意のセルをマージする

グリッドの**AllowMerging** プロパティをAllに設定して、マージしたいセル範囲を**IMergeManager**インタフェースを実装したクラス内に指定して任意のセルをマージできます。

VisualBasic

Visual Basic

```
Class MyMergeManager
    Implements IMergeManager

    Public Function GetMergedRange(grid As C1FlexGrid, cellType__1 As CellType, rg
As CellRange) As CellRange Implements IMergeManager.GetMergedRange
        ' データセルの結合を対象します
        '（行または列ヘッダを結合しない場合）
        If cellType__1 = CellType.Cell Then
            If rg.Column >= 1 AndAlso rg.Column <= 2 AndAlso rg.Row >= 1 AndAlso
rg.Row <= 4 Then
                rg.Column = 1
                rg.Column2 = 2
                rg.Row = 1
                rg.Row2 = 4
            End If
        End If

        ' 完了
        Return rg
    End Function
    Private Function GetDataDisplay(grid As C1FlexGrid, r As Integer, c As Integer)
As String
        Return grid(r, c).ToString()
    End Function
```

C#

C#

```
class MyMergeManager : IMergeManager
{
    public CellRange GetMergedRange(C1FlexGrid grid, CellType cellType, CellRange
rg)
    {
        // データセルの結合を対象します
        // （行または列ヘッダを結合しない場合）
        if (cellType == CellType.Cell)
        {
            if (rg.Column >= 1 && rg.Column <= 2 && rg.Row >= 1 && rg.Row <= 4)
            {
                rg.Column = 1;
                rg.Column2 = 2;
                rg.Row = 1;
                rg.Row2 = 4;
            }
        }

        // 完了
        return rg;
    }
    string GetDataDisplay(C1FlexGrid grid, int r, int c)
    {
        return grid[r, c].ToString();
    }
}
```

注意:

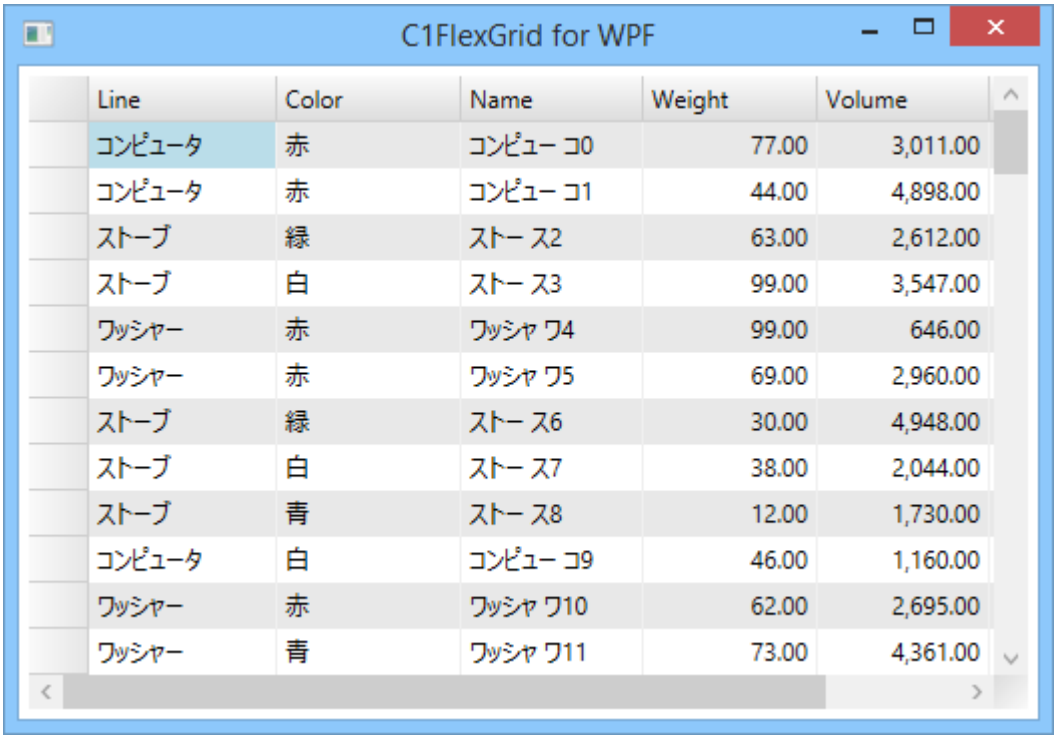
- マージ（結合）を解除する場合は、**MergedRanges** コレクションから削除（**Clear**／**RemoveAt**）します。
- カスタムマージと自動マージを同時に実装することはできません。

スクロール

スクロールバーを表示する

C1FlexGrid でスクロールバーの表示を制御するには、**HorizontalScrollBarVisibility** および **VerticalScrollBarVisibility** プロパティを設定します。デフォルト値は **Auto** ですが、**HorizontalScrollBarVisibility.Disable** に設定すると、スクロールバーを非表示に設定できます。

【実行例】



VisualBasic

```
Public Sub New()
    InitializeComponent()
    _flex.ItemsSource = Product.GetProducts(100)
    _flex.HorizontalScrollBarVisibility = Controls.ScrollBarVisibility.Visible
    _flex.VerticalScrollBarVisibility = Controls.ScrollBarVisibility.Visible
End Sub
```

C#

```
public MainWindow()
{
    InitializeComponent();
    _flex.ItemsSource = Product.GetProducts(100);
    _flex.HorizontalScrollBarVisibility = ScrollBarVisibility.Visible;
    _flex.VerticalScrollBarVisibility = ScrollBarVisibility.Visible;
}
```

注意：スクロールバーの幅を変更する機能は用意されていません。C1FlexGrid コントロールではシステムのスクロールバーを使用します。

スクロール位置を設定する

C1FlexGrid でスクロールの位置を設定する場合は、**ScrollIntoView** メソッドにおいて行と列のインデックスを指定して表示範囲を設定できます。

たとえば、次のコードでは、連結グリッドで[10,3]のセル位置を設定してスクロールバーの位置を設定する方法を示します。

VisualBasic

```
Private Sub btn_Click(sender As Object, e As RoutedEventArgs)
    _flex.ScrollIntoView(10, 3)
End Sub
```

C#

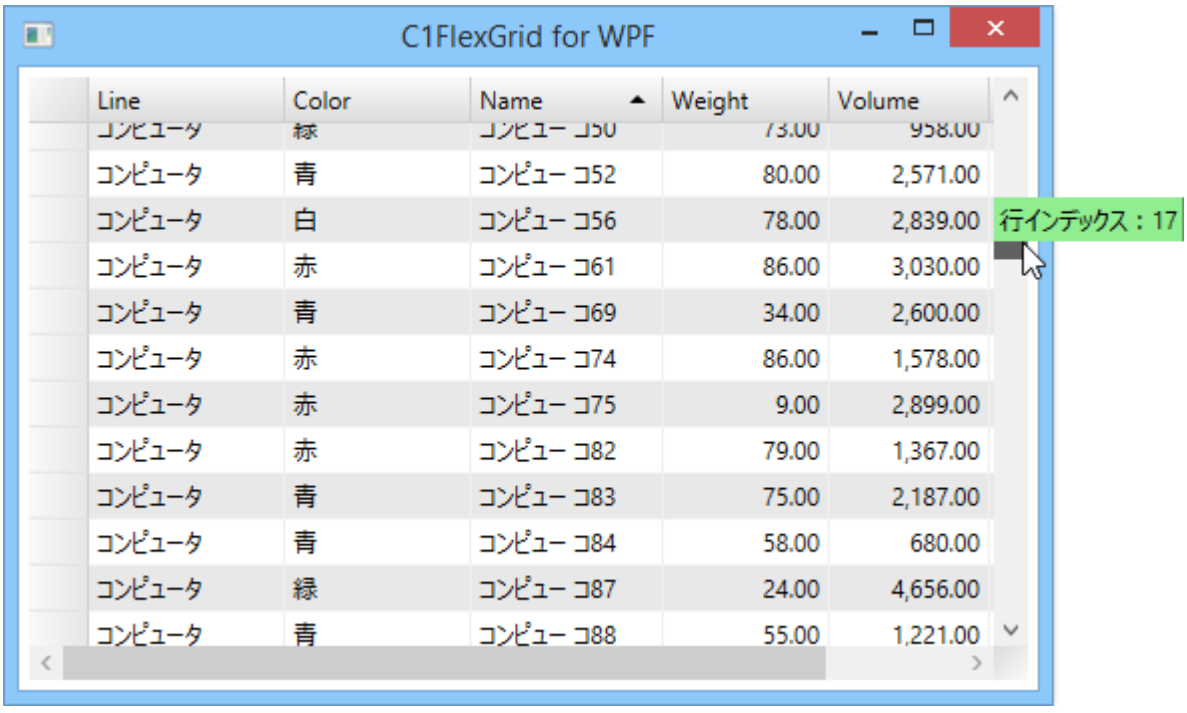
```
private void btn_Click(object sender, RoutedEventArgs e)
{
    _flex.ScrollIntoView(10, 3);
}
```

スクロールチップを表示する

グリッドで垂直スクロールバーのスクロールチップを表示するには、サムをドラッグしたときの行インデックスを取得してスクロールチップにテキストとして表示できます。ユーザーが垂直スクロールバーのサムをドラッグすると、そのトップ行のインデックスはカレントビューポートに入力されます。

たとえば、次のコードでは、グリッドをデータソースと連結していることを前提として、スクロールバーのサムを移動する際にスクロールチップを表示する方法を示します。

【実行例】



VisualBasic

```
Private Shared _vsb As ScrollBar

Public Sub New()
    InitializeComponent()
    _flex.ItemsSource = Product.GetProducts(100)
End Sub

Private Sub MainWindow_Loaded(sender As Object, e As RoutedEventArgs) Handles
Me.Loaded
    Dim scroll As DependencyObject = FindInVisualTreeDown(_flex,
GetType(ScrollBar), AddressOf IsVerticalScrollBar)
    _vsb = DirectCast(scroll, ScrollBar)
    AddHandler _vsb.Track.Thumb.MouseLeave, AddressOf Thumb_MouseLeave
    AddHandler _vsb.ValueChanged, AddressOf _vsb_ValueChanged
End Sub

Private Sub Thumb_MouseLeave(sender As Object, e As MouseEventArgs)
    _popup.IsOpen = False
End Sub

Private Sub _vsb_ValueChanged(sender As Object, e As
RoutedPropertyChangedEventArgs(Of Double))
    If _vsb.Track.Thumb.IsMouseOver And Mouse.LeftButton = MouseButtonState.Pressed
Then
        _popup.IsOpen = True
        Dim p As Point = Mouse.GetPosition(layoutRoot)
        Dim relativePoint As Point =
_vsb.TransformToAncestor(layoutRoot).Transform(New Point(0, 0))
        _popup.VerticalOffset = p.Y
        _popup.HorizontalOffset = relativePoint.X
        content.Text = String.Format("Row number is {0}", _flex.ViewRange.TopRow)
    End If
End Sub

Private Function IsVerticalScrollBar(obj As DependencyObject) As Boolean
    Dim scrollbar As ScrollBar = DirectCast(obj, ScrollBar)
    If scrollbar IsNot Nothing Then
        Return scrollbar.Orientation = Orientation.Vertical
    End If
    Return False
End Function

Public Shared Function FindInVisualTreeDown(obj As DependencyObject, type As Type,
func As Func(Of DependencyObject, Boolean)) As DependencyObject
    If obj IsNot Nothing Then
        If obj.[GetType]() = type AndAlso func(obj) Then
            Return obj
        End If
```

```
        For i As Integer = 0 To VisualTreeHelper.GetChildrenCount(obj) - 1
            Dim childReturn As DependencyObject =
FindInVisualTreeDown(VisualTreeHelper.GetChild(obj, i), type, func)
            If childReturn IsNot Nothing Then
                Return childReturn
            End If
        Next
    End If

    Return Nothing
End Function
```

C#

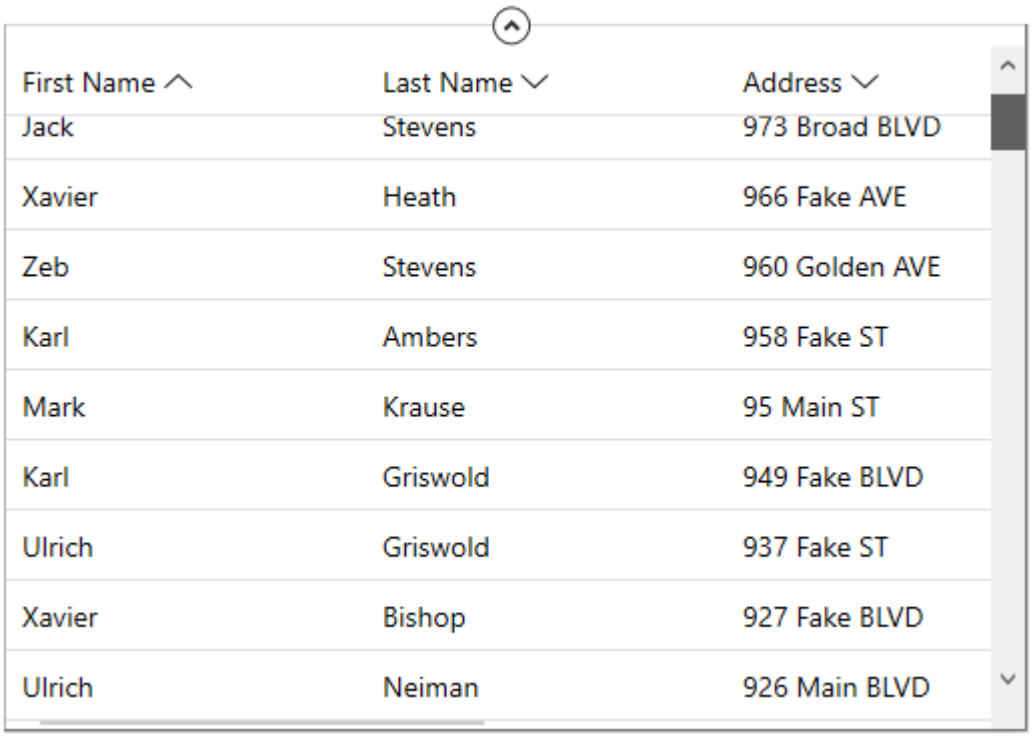
```
public MainWindow()
{
    InitializeComponent();
    this.Loaded += MainWindow_Loaded;
    _flex.ItemsSource = Product.GetProducts(100);
}

ScrollBar _vsb;
private void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
    var scroll = FindInVisualTreeDown(_flex, typeof(ScrollBar),
IsVerticalScrollBar);
    _vsb = scroll as ScrollBar;
    _vsb.ValueChanged += _vsb_ValueChanged;
    _vsb.Track.Thumb.MouseLeave += Thumb_MouseLeave;
}

void Thumb_MouseLeave(object sender, MouseEventArgs e)
{
    _popup.IsOpen = false;
}

void _vsb_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (_vsb.Track.Thumb.IsMouseOver && Mouse.LeftButton ==
MouseButtonState.Pressed)
    {
        _popup.IsOpen = true;
        var p = Mouse.GetPosition(layoutRoot);
        Point relativePoint = _vsb.TransformToAncestor(layoutRoot)
                  .Transform(new Point(0, 0));
        _popup.VerticalOffset = p.Y;
        _popup.HorizontalOffset = relativePoint.X;
        content.Text = string.Format("行インデックス:{0}", _flex.ViewRange.TopRow);
    }
}

private bool IsVerticalScrollBar(DependencyObject obj)
{
    var scrollbar = obj as ScrollBar;
    if (scrollbar != null)
    {
        return scrollbar.Orientation == Orientation.Vertical;
    }
    return false;
}

public static DependencyObject FindInVisualTreeDown(DependencyObject obj, Type
type, Func<DependencyObject, bool> func)
```

```
{
    if (obj != null)
    {
        if (obj.GetType() == type && func(obj))
        {
            return obj;
        }

        for (int i = 0; i < VisualTreeHelper.GetChildrenCount(obj); i++)
        {
            DependencyObject childReturn =
FindInVisualTreeDown(VisualTreeHelper.GetChild(obj, i), type, func);
            if (childReturn != null)
            {
                return childReturn;
            }
        }
    }

    return null;
}
```

ソート

複数列のソート

The Multi-Column sorting feature allows users to sort multiple columns in the grid, that is sort one column by ascending and the other column by descending without hampering either column's sorting.



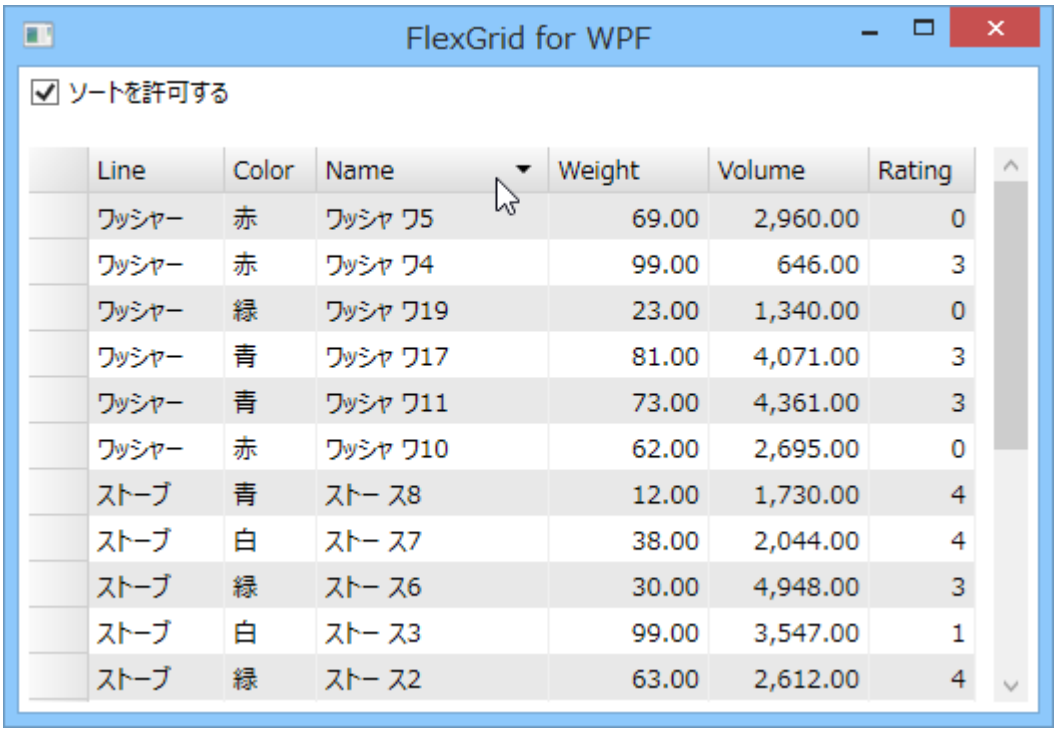
Multi-column sorting can be performed over the grid by pressing either the **shift** or **ctrl** key and clicking the **column**

header.

ソートを許可する

C1FlexGrid の **AllowSorting** プロパティはブール型ですので、各列の **AllowSorting** プロパティを **True**や**False**に設定することで、ユーザーが特定の列を基準にしたソートを許可・不可にすることができます。

たとえば、次のコードでは、グリッドをデータソースと連結していることを前提として、ユーザーによるソートを許可します。



Visual Basic

Visual Basic

```
Private Sub sortChk_Click(sender As Object, e As RoutedEventArgs) Handles
sortChk.Click
    Dim chk = TryCast(sender, CheckBox)
    Dim isChked = If(chk.IsChecked, False)
    If _flex IsNot Nothing Then
        _flex.AllowSorting = isChked
    End If
End Sub
```

C#

C#

```
private void sortChk_Click(object sender, RoutedEventArgs e)
{
    var chk = sender as CheckBox;
    var isChked = chk.IsChecked ?? false;
    if (_flex != null)
        _flex.AllowSorting = isChked;
}
```

ソートを実行する

ほとんどのグリッドと同様に、**C1FlexGrid** はソートをサポートします。列ヘッダをクリックすると、データは昇順または降順でソートされます。グリッドをソートすると、対応する列に、現在のソート方向を示す三角形が表示されます。

C1FlexGrid では、一般的なソート切り替え動作に加えて、[Ctrl]キーを押しながら列ヘッダをクリックすることでソートを解除できます。これにより、列に適用されているソートは解除され、データは元の順序で表示されます。

グループ化と同様に、実際のソートは、データソースとして使用される **ICollectionView** によって実行されます。グリッドは単にマウスクリックを検出するだけであり、ソートはこのデータソースオブジェクトに委任されます。また、コードを使ってデータをソートすることもできます。

たとえば、次のコードは、データを氏名および都道府県に基づいてソートします。

Visual Basic

```
' 初期状態にします。

view.SortDescriptions.Clear()

' 氏名に基づいてソートします。

view.SortDescriptions.Add(New SortDescription("氏名", ListSortDirection.Ascending))

' 次に、都道府県に基づいてソートします。

view.SortDescriptions.Add(New SortDescription("都道府県",
ListSortDirection.Ascending))
```

C#

```
// 初期状態にします。
view.SortDescriptions.Clear();

// 氏名に基づいてソートします。
view.SortDescriptions.Add(
  new SortDescription("氏名", ListSortDirection.Ascending));

// 次に、都道府県に基づいてソートします。
view.SortDescriptions.Add(
  new SortDescription("都道府県", ListSortDirection.Ascending));
```

このコードが呼び出されると、グリッドは、自動的にデータを新しいソート順で表示し、都道府県の列ヘッダに三角形のソートインジケータを表示します。

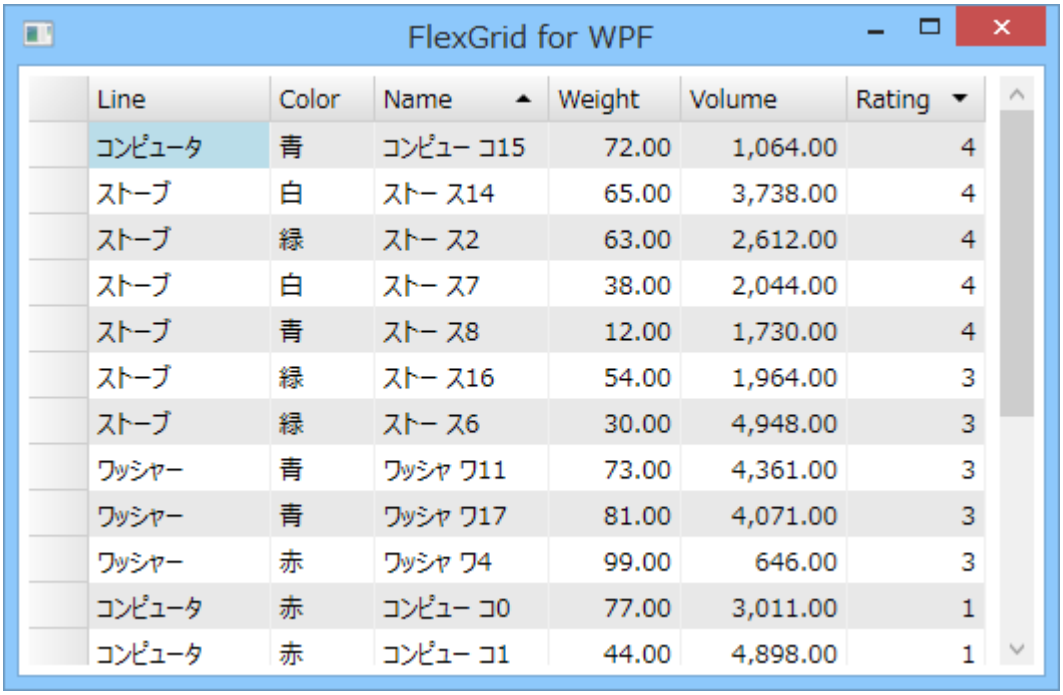
グリッドのソートを無効にするには、**AllowSorting** プロパティを false に設定します。特定の列のソートを無効にするには、**Column.AllowSorting** プロパティを false に設定します。また、**ShowSort** プロパティを false に設定することで、三角形のソートインジケータを非表示にすることもできます。

複数列のソートを実行する

グリッドで複数列をソートできるように、**ICollectionView.SortDescriptions** に対象列の名前を指定して実現できます。

たとえば、次のコードでは、グリッドはデータソースと連結していることを前提として、「Name」と「Rating」列に基づいてソートする方法を示します。

> **メモ**:グリッドで列の表示順番によってソート結果が異なりますのでご注意ください。たとえば、「Rating」列は「Name」の先に表示されている場合「Rating」のソート順番が優先されてその後「Name」に基づいてソートされます。



VisualBasic

```
Public Sub New()
    InitializeComponent()
    _flex.ItemsSource = Product.GetProducts(30)
    'ICollectionView.SortDescriptions を使用します
    _flex.CollectionView.SortDescriptions.Add(New
System.ComponentModel.SortDescription("Rating",
System.ComponentModel.ListSortDirection.Ascending))
    _flex.CollectionView.SortDescriptions.Add(New
System.ComponentModel.SortDescription("Name",
System.ComponentModel.ListSortDirection.Ascending))
End Sub
```

C#

```
public MainWindow()
{
    InitializeComponent();
    _flex.ItemsSource = Product.GetProducts(30);
    //ICollectionView.SortDescriptions を使用します
    _flex.CollectionView.SortDescriptions.Add(new
System.ComponentModel.SortDescription("Rating",
System.ComponentModel.ListSortDirection.Descending));
    _flex.CollectionView.SortDescriptions.Add(new
System.ComponentModel.SortDescription("Name",
System.ComponentModel.ListSortDirection.Ascending));

}
```

ソートを解除する

C1FlexGrid でソートを解除したい場合は、グリッドのソートの記述（SortDescriptions）を消去する方法を紹介します。たとえば、_flex と指定した C1FlexGrid で **_flex.CollectionView.SortDescriptions.Clear()** 使用して行ったソート処理を無効にできます。

1. DataTable を作成してグリッドに連結します。

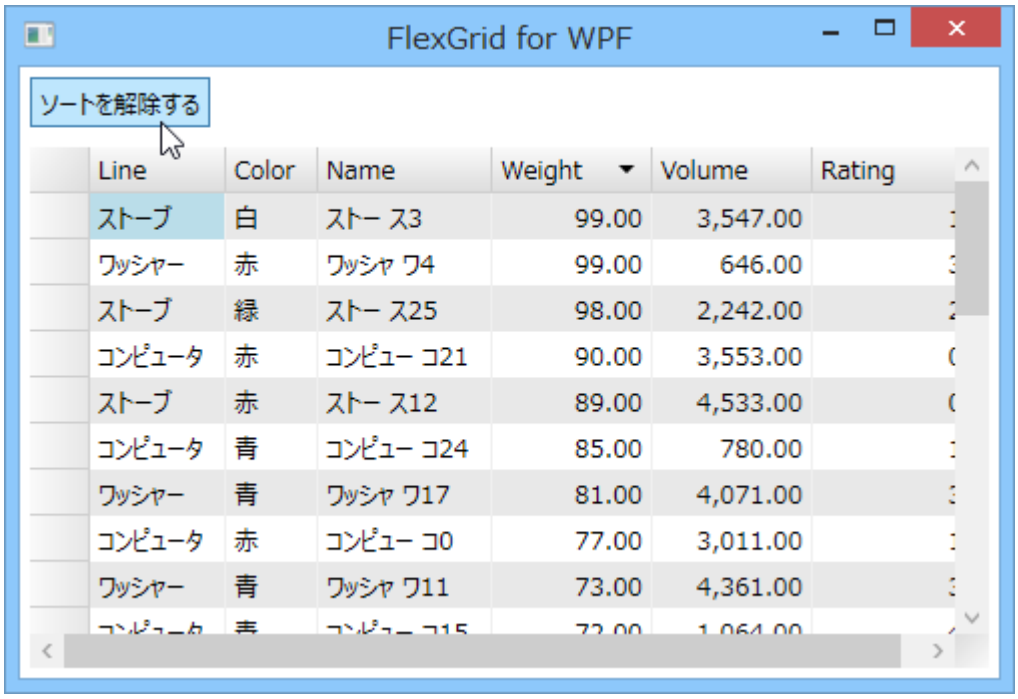
VisualBasic

```
InitializeComponent()
_flex.ItemsSource = Product.GetProducts(30)
_flex.CollectionView.SortDescriptions.Add(New
System.ComponentModel.SortDescription("Name",
System.ComponentModel.ListSortDirection.Descending))
```

C#

```
InitializeComponent();
_flex.ItemsSource = Product.GetProducts(30);
_flex.CollectionView.SortDescriptions.Add(new
System.ComponentModel.SortDescription("Name",
System.ComponentModel.ListSortDirection.Descending));
```

2. クリックされたときテーブルのソートを解除するボタンを作成します。次のコードを removeBtn_Click イベントに追加します。



VisualBasic

```
Private Sub removeBtn_Click(sender As Object, e As RoutedEventArgs)
    If _flex.CollectionView.SortDescriptions.Count > 0 Then
        _flex.CollectionView.SortDescriptions.Clear()
    End If
End Sub
```

C#

```
private void removeBtn_Click(object sender, RoutedEventArgs e)
{
    if (_flex.CollectionView.SortDescriptions.Count > 0)
    {
        _flex.CollectionView.SortDescriptions.Clear();
    }
}
```

メモ:DataTable.DefaultView は DataTable の DataView を返します。ソート文字列を null に設定すると、DataView は直前のソートを元に戻します。

なお、(列ヘッダに表示されている)ソート方向を示すインジケータのみを消去したい場合は、**ShowSort** プロパティを False に設定してアイコンを非表示にします。ShowSort プロパティを False に設定した場合は、再度ソートを行ってもアイコンは表示されなくなりますので、注意してください。非連結モードのグリッドでも使用可能です。

フィルタリング

フィルタリングを許可する

C1FlexGridFilter 拡張クラスを使用し、C1FlexGrid コントロールで列フィルタを実現する方法を紹介します。C1FlexGridFilter 拡張クラスは C1FlexGrid を拡張して Excel 形式のフィルタ処理を提供します。フィルタ処理を有効にすれば、グリッドの列ヘッダ上にマウスポインタを置いた際に、ドロップダウンアイコンが表示されます。このドロップダウンでは、列のフィルタを編集できます。フィルタには2種類のモードがあります。

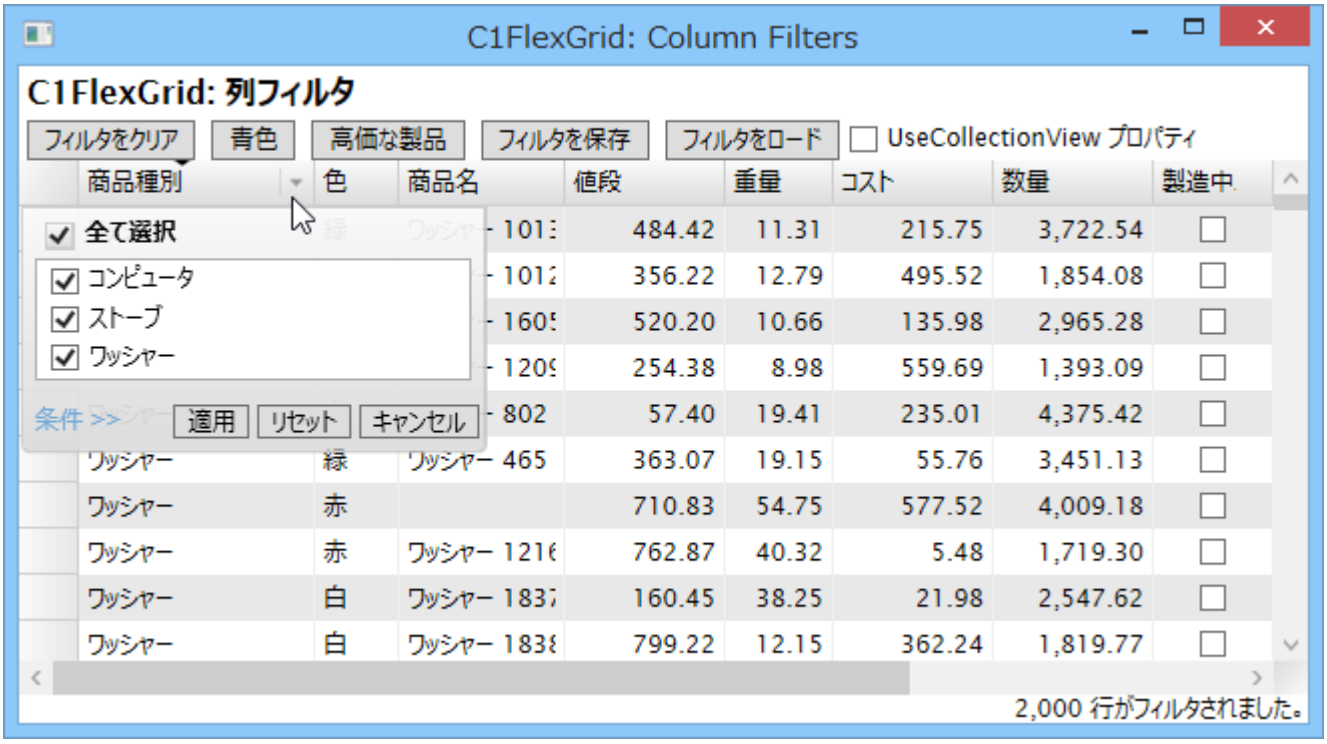
値モード:このモードでは、列のすべての値がリストに表示されます。ユーザーは、表示する値の1つまたは2つをリストから選択できます。

条件モード:このモードでは、エディタが演算子と比較パラメータ(例えば、「より大きい」、「2.5」)から成る 2 つの条件を指定します。これらの条件自体は、AND または OR 演算子で組み合わされます。

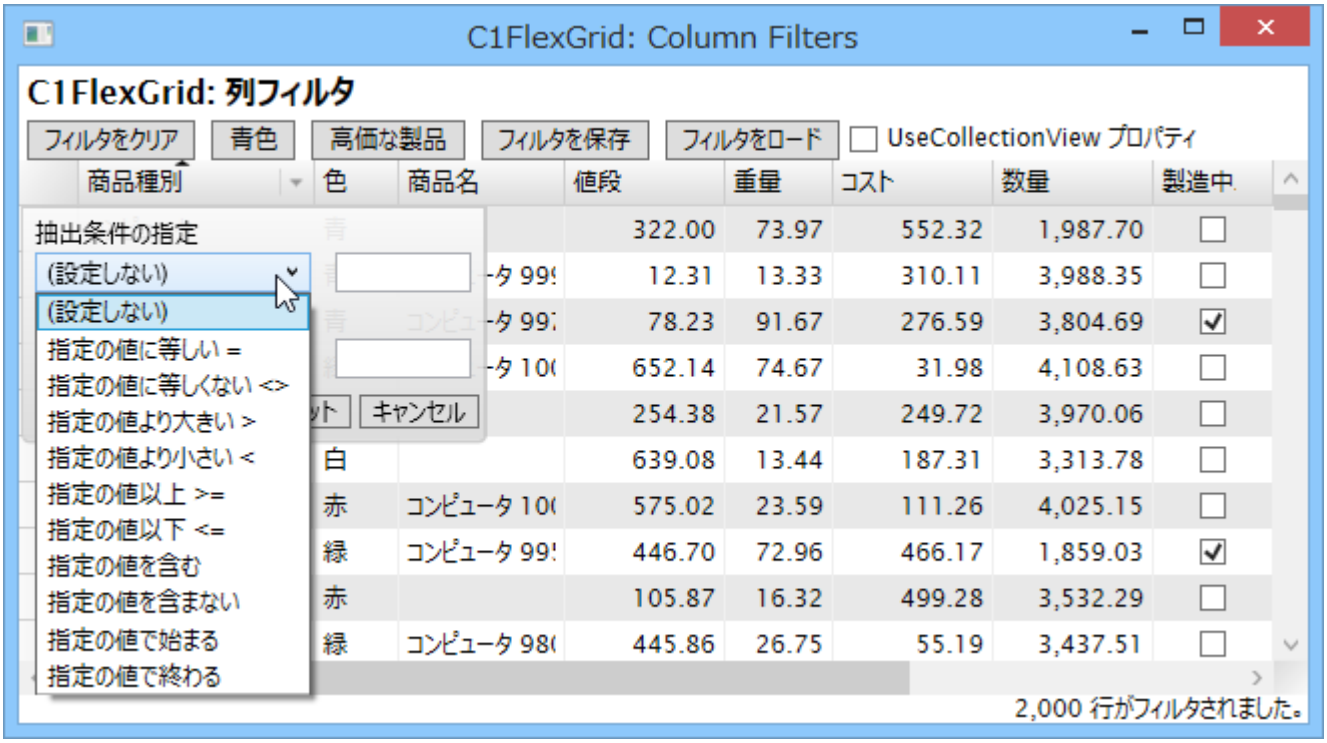
エディタを使用してフィルタ処理の方法を指定できますし、コードで設定することも可能です。

デフォルト設定として、各行に Visible プロパティを設定するとフィルタ処理はグリッドに適用されます。そして、フィルタ基準を満たす行は表示し、満たさない行は非表示になります。このモードでは、フィルタ処理は全行数に影響は及ぼしませんので、簡単な LINQ 表現を用いてフィルタ基準を満たした行の数を計算できます。
なお、グリッドがフィルタ対応のデータソースと連結されている場合、フィルタの **UseCollectionView** プロパティを true に設定すると、フィルタをデータソースに直接適用できます(ICollectionView.Filter プロパティを使用)。そして、このモードを使用する場合はフィルタでの変更は行数に(同じデータソースと連結されている他のコントロールの項目数にも)影響を及ぼします。

【実行例】値フィルタ(Default)



【実行例】条件フィルタ（ByCondition）



フィルタリングを実現するサンプルコードについては、製品付属サンプル「ColumnFilter」をご参照ください。

Excel のようなフィルタ処理

FlexGrid provides Excel-like filtering feature to filter data through drop down/ellipsis icon in column headers. It adds Excel-style filtering UI to each column that allows you to apply different filters to filter data.

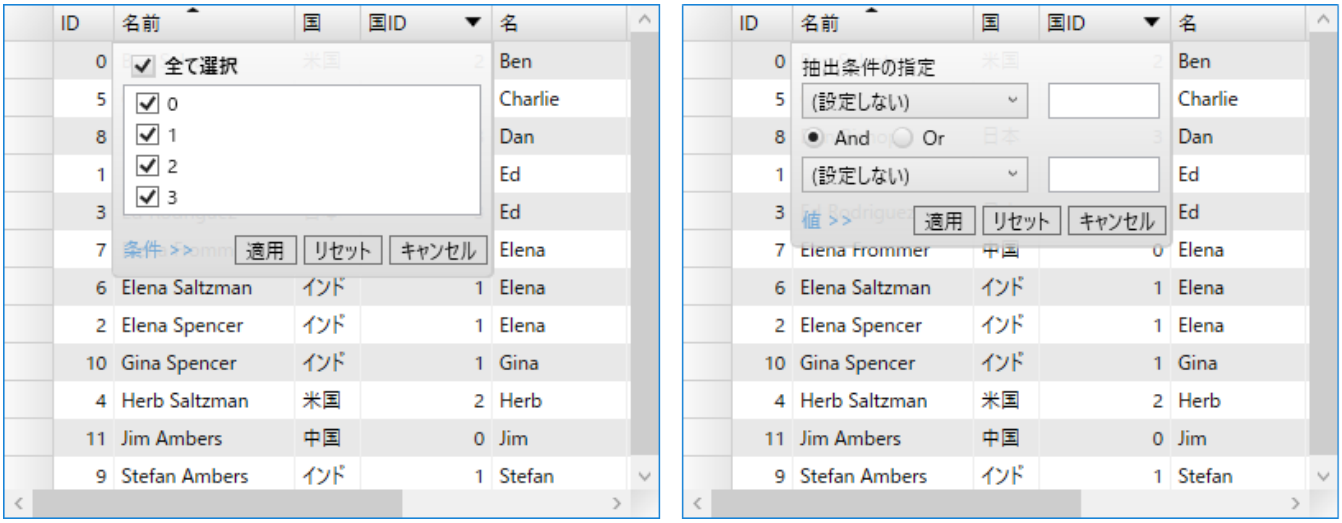
.NET Framework

FlexGrid は、Excel のようなフィルタ処理機能を提供し、列ヘッダーのドロップダウンアイコンでデータをフィルタ処理できます。この機能は、FlexGridFilter という名前の独立したコントロールから FlexGrid に提供されます。このコントロールは、拡張アセンブリC1.WPF.FlexGridFilterとして実装されています。FlexGridFilter コントロールが追加されると、列ヘッダーにマウスを置いたときにグリッドにドロップダウンアイコンが表示されます。ドロップダウンアイコンには、列のフィルタを指定するためのエディタが表示されます。ユーザーは、2 種類のフィルタから選択できます。

- **値フィルタ**:このフィルタでは、列で特定の値をフィルタできます。
- **条件フィルタ**:このフィルタを使用すると、演算子(より大きい、より小さいなど)とパラメータから成る条件を指定できます。**AND** または **OR** 演算子を使用して、条件を結合できます。

Note: Filtering using **FlexGridFilter** is only available in .NET Framework version.

次の図に、ドロップダウンアイコンをクリックして表示されるフィルタを示します。



FlexGridFilter からフィルタ処理を有効にするには

次のコードに示すように、**C1.WPF.FlexGridFilter** アセンブリを手動でプロジェクトに追加し、**C1FlexGridFilter** クラスのオブジェクトを作成し、このオブジェクトを既存の FlexGrid オブジェクトに追加することで、FlexGrid で手動でフィルタ処理を有効にできます。

```
<c1:C1FlexGrid Name="grid" >
  <!-- コントロールにフィルタ処理のサポートを追加します。-->
  <c1:C1FlexGridFilterService.FlexGridFilter>
    <c1:C1FlexGridFilter />
  </c1:C1FlexGridFilterService.FlexGridFilter>
</c1:C1FlexGrid>
```

```
// FlexGrid を作成します
var grid = new C1FlexGrid();
//FlexGridFilter からフィルタ処理を有効にします
var gridFilter = new C1FlexGridFilter(grid);
```

フィルタモードを選択するには

フィルタは、**UseCollectionView** プロパティの値に応じて 2 つのモードで動作します。UseCollectionView プロパティを **false** に設定すると、フィルタを満たさない行が非表示になります(フィルタは **Visible** プロパティを **false** に設定します)。このモードでは、フィルタは行数に影響を及ぼしません。このモードは、連結および非連結のグリッドで使用できます。

フィルタの UseCollectionView プロパティを **true** に設定した場合、フィルタはデータソースに適用されます。このモードでは、フィルタが変更されると、グリッドや同じデータソースに連結されている他のすべてのコントロールに対してデータソースから公開されている項目の数に影響を及ぼします。このフィルタ処理モードは、連結グリッドでのみ使用できます。

```
<c1:C1FlexGrid Name="grid" >
  <!-- コントロールにフィルタ処理のサポートを追加します。-->
  <c1:C1FlexGridFilterService.FlexGridFilter>
    <c1:C1FlexGridFilter UseCollectionView=”True”/>
  </c1:C1FlexGridFilterService.FlexGridFilter>
</c1:C1FlexGrid>

// C1FlexGrid を作成します
var grid = new C1FlexGrid();

// グリッドでフィルタ処理を有効にします
var gridFilter = new C1FlexGridFilter(grid);

// データソースレベルでフィルタ処理します
gridFilter.UseCollectionView = true;
```

列ごとにフィルタタイプをカスタマイズするには

デフォルトでは、すべての列でフィルタが有効です。Boolean または列挙データを含む列は値フィルタを受け取り、他のデータ型を含む列は**値**と**条件**を受け取ります。FilterType プロパティを使用すると、この動作を変更して、各列でフィルタタイプを指定できます。

列に一意の値が多くある場合や、フィルタが機能しない連結が列に含まれる場合は、フィルタタイプの指定が重要です。たとえば、画像を含む列を値フィルタまたは条件フィルタでフィルタ処理することはできません。この場合は、FilterType プロパティを **None** に設定することで、フィルタを無効にします。

また、数千個の項目が含まれるグリッドに一意の ID 列がある場合は、値フィルタに追加される項目が多くなりすぎるため、処理が遅くなり、あまり役に立ちません。この場合は、FilterType プロパティを **Condition** に設定することで、値フィルタを無効にします。

次のコードは、この方法を示します。

C#

```
// C1FlexGrid を作成します
var grid= new C1FlexGrid();

// グリッドでフィルタ処理を有効にします
var gridFilter = new C1FlexGridFilter(grid);

// Image 列のフィルタを無効にします
var columnFilter = gridFilter.GetColumnFilter(grid.Columns["Image"]);
columnFilter.FilterType = FilterType.None;

// ID 列の値フィルタを無効にします
columnFilter = gridFilter.GetColumnFilter(grid.Columns["ID"]);
columnFilter.FilterType = FilterType.Condition;
```

コードでフィルタタイプを指定するには

多くの場合は、ユーザーがフィルタを設定します。ただし、**ColumnFilter** クラスは、開発者がコードでフィルタ条件をカスタマイズできる完全なオブジェクトモデルを公開します。たとえば、次のコードは、フィルタを 2 番目の列に適用します。このフィルタにより、グリッドには、2 番目の列の値に文字 Z が含まれる項目が表示されます。

C#

```
//  C1FlexGrid を作成します
var grid = new C1FlexGrid();

// グリッドでフィルタ処理を有効にします
var gridFilter = new C1FlexGridFilter(grid);
```

```
// 最初の列に対するフィルタを取得します
var columnFilter = gridFilter.GetColumnFilter(grid.Columns[0]);

// 「Z」を含むフィルタ条件を作成します
var condition = columnFilter.ConditionFilter.Condition1;
condition.Operator = ConditionOperator.Contains;
condition.Parameter = "Z";

// フィルタを適用します
gridFilter.Apply();
```

フィルタを永続化するには

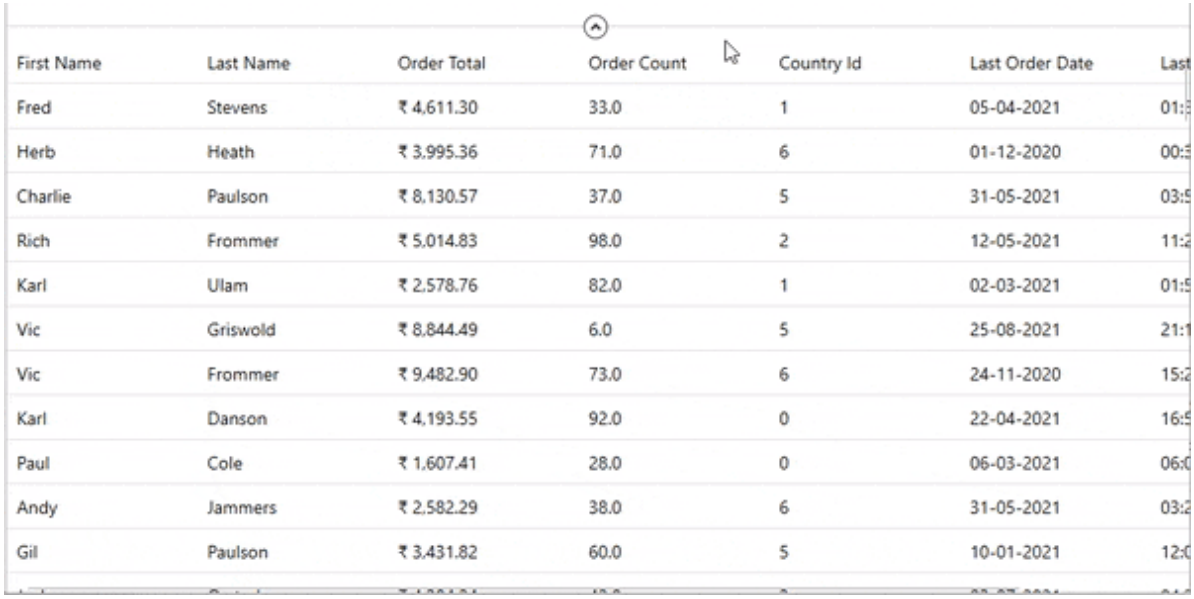
C1FlexGridFilter クラスには、現在のフィルタ状態を XML 文字列として取得または設定する **FilterDefinition** プロパティがあります。この文字列を使用して、ユーザーがアプリケーションを終了したときのフィルタ状態を保存し、後で復元することができます。複数のフィルタ定義を保存することもできます。ユーザーは、これらの設定済みのフィルタを選択してカスタマイズできます。**SaveFilterDefinition** および **LoadFilterDefinition** メソッドを使用して、フィルタ定義をストリームに保存して復元することもできます。

メモ: **C1.WPF.FlexGridFilter** 拡張アセンブリは、次の理由で FlexGrid とは独立して提供されています。

- FlexGrid アセンブリのフットプリントを最小限に抑えます。
- 拡張の選択に完全な柔軟性を提供します。
- カスタムコードで **C1FlexGrid** クラスの機能を拡張します。

Excel-like filtering is available in FlexGrid by default, and the **column FilterLoading event** allows you to customize the filtering behavior and allows you to display conditional filters, by default. It lets you to get the data filter to be displayed for column filtering using the **DataFilter** property. This filter can then be added to the column for filtering values. Contrastingly, you can disable the filtering on any column by setting its **AllowFiltering** property to **false**.

By default, the numeric (conditional) filters are enabled for every column with numeric data. In addition, you may choose to display value filter for the columns with Boolean or enumerated data and display a combination of value and conditional filters for the columns that contain the data of other data types as showcased in the following code. Here, the value filter lets you filter specific values in the column and the conditional filter lets you specify conditions composed of an operator (greater than, less than, etc.) and a parameter. The conditions can be combined using an **AND** or an **OR** operator.



First Name	Last Name	Order Total	Order Count	Country Id	Last Order Date	Last
Fred	Stevens	₹ 4,611.30	33.0	1	05-04-2021	01:
Herb	Heath	₹ 3,995.36	71.0	6	01-12-2020	00:
Charlie	Paulson	₹ 8,130.57	37.0	5	31-05-2021	03:
Rich	Frommer	₹ 5,014.83	98.0	2	12-05-2021	11:
Karl	Ulam	₹ 2,578.76	82.0	1	02-03-2021	01:
Vic	Griswold	₹ 8,844.49	6.0	5	25-08-2021	21:
Vic	Frommer	₹ 9,482.90	73.0	6	24-11-2020	15:
Karl	Danson	₹ 4,193.55	92.0	0	22-04-2021	16:
Paul	Cole	₹ 1,607.41	28.0	0	06-03-2021	06:
Andy	Jammers	₹ 2,582.29	38.0	6	31-05-2021	03:
Gil	Paulson	₹ 3,431.82	60.0	5	10-01-2021	12:

The following code uses the Customer class created in Quick Start. Here, the FlexGrid columns are bound to the customer's first name, last name, order total, order count, country id, last order date, and last order time as shown in the following XAML code.

XAML

```
<c1:FlexGrid x:Name="grid" AutoGenerateColumns="False">
    <c1:FlexGrid.Columns>
        <c1:GridColumn Binding="FirstName"/>
        <c1:GridColumn Binding="LastName"/>
        <c1:GridColumn Binding="OrderTotal" Format="C2"/>
        <c1:GridColumn Binding="OrderCount" Format="N1"
FilterLoading="Order_FilterLoading"/>
        <c1:GridColumn Binding="CountryId" Header="Country Id"
FilterLoading="Country_FilterLoading"/>
        <c1:GridDateTimeColumn Binding="LastOrderDate" Mode="Date"/>
        <c1:GridDateTimeColumn Binding="LastOrderDate" Header="Last Order Time"
Mode="Time"/>
    </c1:FlexGrid.Columns>
</c1:FlexGrid>
```

To display value filter and conditional filters on the grid created above, use the following code:

C#

```
private List<Customer> _data;
```

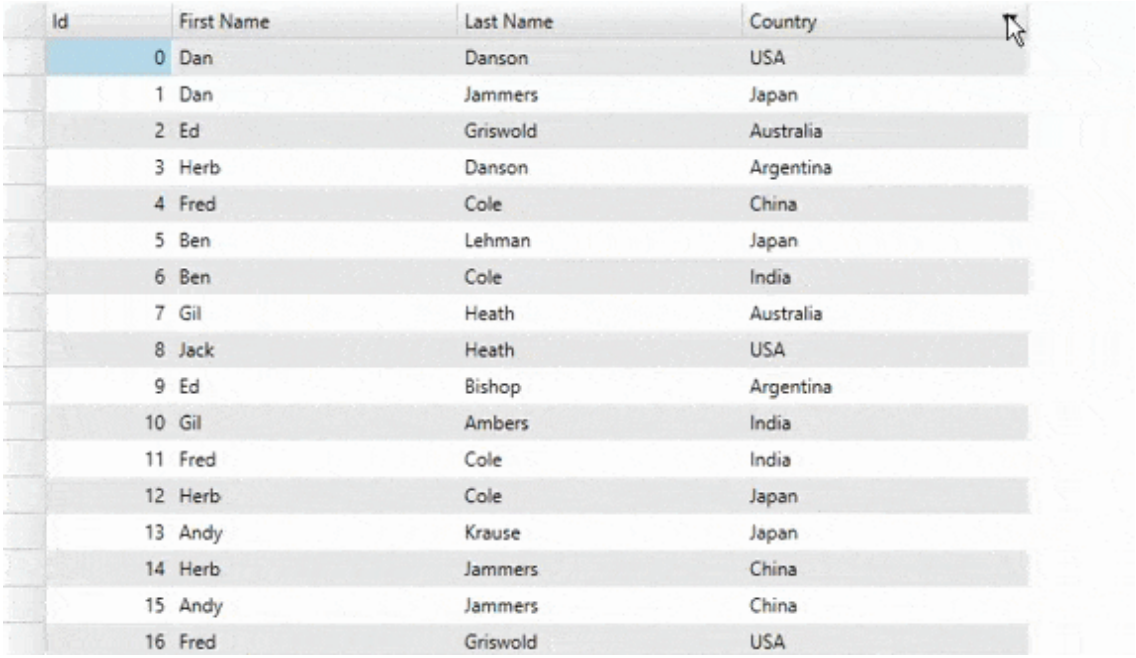
```
public ExcelFiltering()
{
    InitializeComponent();

    _data = Customer.GetCustomerList(100).ToList();
    // 値フィルタを表示します
    grid.Columns[4].DataMap = new C1.WPF.Grid.GridDataMap { ItemsSource =
Customer.GetCountries(), DisplayMemberPath = "Key", SelectedValuePath = "Key" };
    grid.ItemsSource = _data;
}

private void Country_FilterLoading(object sender,
C1.WPF.Grid.GridColumnFilterLoadingEventArgs e)
{
     e.DataFilter.Filters.Add(new NumericFilter() { PropertyName = "CountryId" });
     e.ShowApplyButton = true;
     e.ShowClearButton = true;
}

private void Order_FilterLoading(object sender,
C1.WPF.Grid.GridColumnFilterLoadingEventArgs e)
{
    // 数値（条件付き）フィルタは、数値データのデフォルトで表示されます。
    e.ShowApplyButton = true;
    e.ShowClearButton =true;
}
```

Multi-value filtering

In FlexGrid, you can easily filter a specific value from a column using **Filter** property but, sometimes, you may want to filter multiple values from a single column containing large data. For such situations, you can use the multi-value filtering to explicitly set a specific filter type for a column. FlexGrid allows you to apply multi-value filtering on a column to filter out relevant information from large data. In multi-value filtering, a checked list of possible values is displayed allowing you to select the values you wish to filter out of the view. This is a good option for columns where you have several unique and repeated values (like a category or type field). Let's say, you have a huge list of countries in your database and want to expand your business in specific countries. In such scenarios, you can apply multi-value filtering on your data to filter out the customers and get their contact details. The following example shows how you can achieve this by applying multi-value filtering to the Country column in FlexGrid.

.NET Framework

The following GIF shows how multi-value filtering can be used in FlexGrid to select multiple countries at runtime.



Id	First Name	Last Name	Country
0	Dan	Danson	USA
1	Dan	Jammers	Japan
2	Ed	Griswold	Australia
3	Herb	Danson	Argentina
4	Fred	Cole	China
5	Ben	Lehman	Japan
6	Ben	Cole	India
7	Gil	Heath	Australia
8	Jack	Heath	USA
9	Ed	Bishop	Argentina
10	Gil	Ambers	India
11	Fred	Cole	India
12	Herb	Cole	Japan
13	Andy	Krause	Japan
14	Herb	Jammers	China
15	Andy	Jammers	China
16	Fred	Griswold	USA

You can apply multi-value filtering in FlexGrid by using the **FilterType** property of the **ColumnFilter** class to set the filter type to Value using the **FilerType** enumeration.

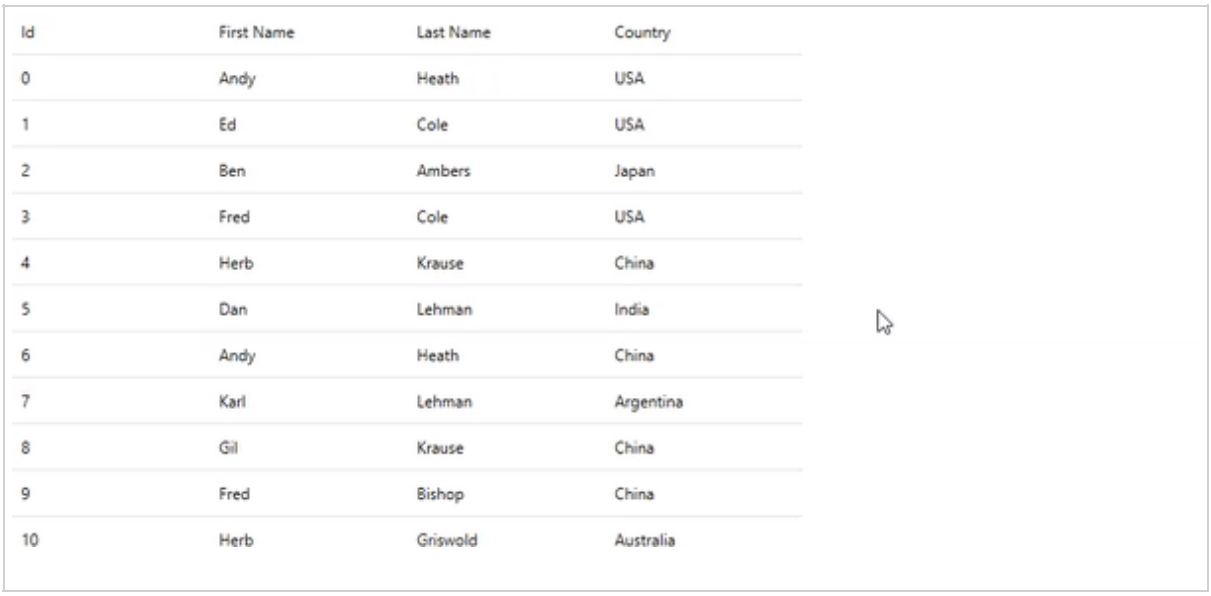
Use the following code snippet to enable multi-value filtering for the **Country** column.

C#

```
//FlexGridFilterによるフィルタリングを有効にします
var gridFilter = new C1FlexGridFilter(grid);
var columnFilter = gridFilter.GetColumnFilter(flex.Columns["Country"]);
columnFilter.FilterType = FilterType.Value;
```

.NET

The following GIF shows how multi-value filtering can be used in FlexGrid to select multiple countries at runtime.



Id	First Name	Last Name	Country
0	Andy	Heath	USA
1	Ed	Cole	USA
2	Ben	Ambers	Japan
3	Fred	Cole	USA
4	Herb	Krause	China
5	Dan	Lehman	India
6	Andy	Heath	China
7	Karl	Lehman	Argentina
8	Gil	Krause	China
9	Fred	Bishop	China
10	Herb	Griswold	Australia

To apply multi-value filtering a column, say Country, you can generate the **FilterLoading** event for it. In the **FilterLoading** event, create an instance of the **ChecklistFilter** class and add it to the FilterCollection to explicitly set the checklist filter type for the Country column as shown in the following code. The following code uses the Customer class created in Quick Start.

C#

```
private void Country_FilterLoading(object sender,
C1.WPF.Grid.GridColumnFilterLoadingEventArgs e)
    {
        e.DataFilter.Filters.Clear();
        C1.WPF.DataFilter.ChecklistFilter filter = new
C1.WPF.DataFilter.ChecklistFilter();
        List<object> filterList = new List<object>();
                foreach (Customer data in grid.ItemsSource)
            filterList.Add(data.Country);
            filter.ItemsSource = filterList;
            filter.PropertyName = "Country";
            e.DataFilter.Filters.Add(filter);
    }
```

フィルタリングを適用する(DataCollection を使用)

IDataCollection インターフェイスでは、**Filter** プロパティを使用したデータのフィルタ処理がサポートされます。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(このタイプの方法は、述語と呼ばれています)。次のセクションでは、FlexGrid .NET 4.5.2および.NET 5バージョンで**DataCollection**を使用してフィルターデータを実装する方法を学習します。

.NET Framework

ユーザーコントロールSearchBox をIDataCollectionフィルタの例とします。このコントロールは、検索する値をユーザーが入力する TextBox コントロールと、1 つのタイマーから構成されます。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って動作し、フィルタを適用します。

C#

```
_view.Filter = null;
_view.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、すべての項目を表示します。
    if (string.IsNullOrEmpty(srch))
        {
            return true;
        }

// 任意の指定したプロパティのテキストを含む項目を表示します。
    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;
};
```

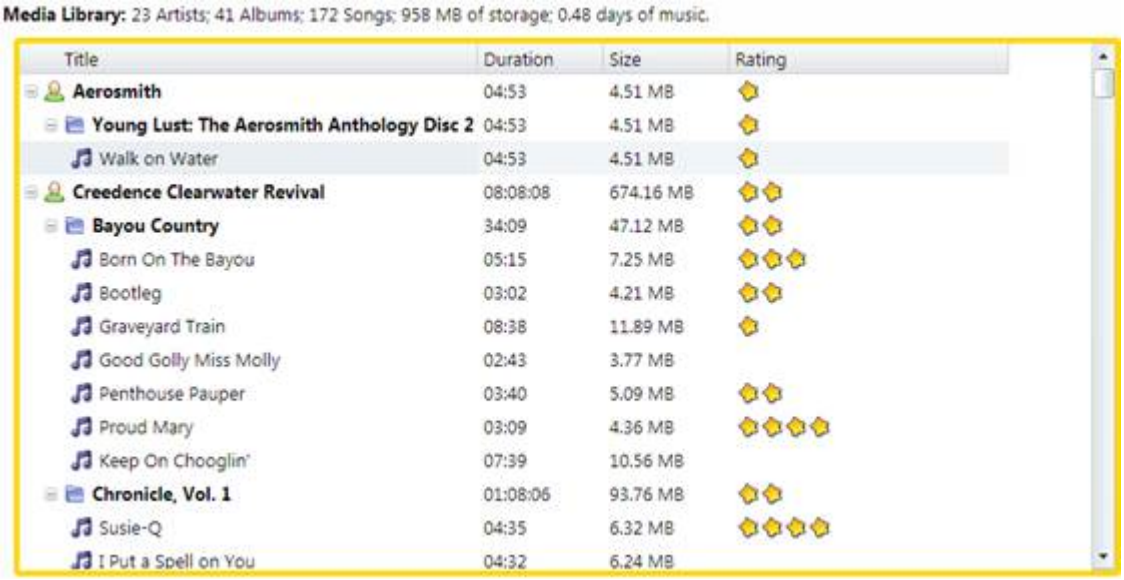
ラムダ関数を使って **Filter** プロパティの値を設定していることに注意してください。独立したメソッドを提供することもできましたが、この表記は簡潔であり、必要に応じてローカル変数を使用できるため、こちらの方が使いやすい場合も数多く存在します。

このラムダ関数は、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。

フィルタが適用されると、グリッド（および DataCollection 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。

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（**MainTestApplication** サンプルより）に、非常に多い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（曲、アルバム、アーティスト）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。

グリッドの上に表示されているステータスラベルに注目してください。リストが変更されるたびに、このラベルは自動的に更新されます。したがって、フィルタが適用されると、このステータスは新しいフィルタを反映して更新されます。ステータスを更新するルーチンは、LINQ を使用して、選択されたアーティスト、アルバム、および曲の数を計算し、合計のストレージと再生時間を計算します。曲のステータス更新ルーチンは、次のように実装されます。

C#

```
// update song status
 void UpdateSongStatus()
 {
   var view = _flexiTunes.ItemsSource as DataCollection;
   var songs = view.OfType<Song>();
   _txtSongs.Text = string.Format(
      "{0:n0} Artists; {1:n0} Albums; {2:n0} Songs; " +
      "{3:n0} MB of storage; {4:n2} days of music.",
     (from s in songs select s.Artist).Distinct().Count(),
     (from s in songs select s.Album).Distinct().Count(),
     (from s in songs select s.Name).Count(),
     (double)(from s in songs select s.Size/1024.0/1024.0).Sum(),
     (double)(from s in songs select s.Duration/3600000.0/24.0).Sum());
 }
```

このルーチンは、グリッドには直接関連しませんが、大きなデータソースに連結されたグリッドを表示する際に、情報の取得が必要になることが多いため、LINQ 機能を活用する方法を示しました。

上の LINQ 文は、**Distinct** および **Count** コマンドを使用して、現在データソースで公開されているアーティスト、アルバム、および曲の数を計算します。また、**Sum** コマンドを使用して、現在の選択範囲の合計のストレージと再生時間を計算します。

ICollectionView のフィルタ述語は子行をフィルタできません。子行をフィルタできるには、子行のタイプはICollectionView に設定されている必要があります。親グリッドと子行の両方がICollectionViewタイプに設定されている場合、両方をフィルタすることが可能です。DataCollection を使用する子行では、ソートが対応されていません。

.NET

ユーザーコントロールSearchBox をDataCollectionフィルタの例とします。このコントロールは、検索する値をユーザーが入力する TextBox コントロールと、1 つのタイマーから構成されます。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って動作し、フィルタを適用します。

ラムダ関数を使って **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。独立したメソッドを提供することもできましたが、この表記は簡潔であり、必要に応じてローカル変数を使用できるため、こちらの方が使いやすい場合も数多く存在します。

このラムダ関数は、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。

C#

```
C1DataCollection<Customer> _view;
public MainWindow()
{
    InitializeComponent();

    //グリッドデータを生成します。
    var customerlist = Customer.GetCustomerList(50);
    _view = new C1.DataCollection.C1DataCollection<Customer>(customerlist);
    //グリッドにデータをバインドします。
    grid.ItemsSource = _view;
}

private void filter_Click(object sender, RoutedEventArgs e)
{
    //検索テキストを取得します。
    if (!string.IsNullOrEmpty(_txtSearch.Text))
    {
        FilterTextExpression filterText = new FilterTextExpression("FirstName",
FilterOperation.Equal, _txtSearch.Text, false, false);
        _view.FilterAsync(filterText);
    }

}
```

フルテキストフィルタを使用してデータの検索

In the the following section learn how to implement Search Data using Full Text Filter in FlexGrid .NET 4.5.2 and .NET 5 versions.

.NET Framework

You can search data in FlexGrid using full text filtering feature wherein the grid's data source is filtered based on the user input to return the search results. The **C1FullTextFilter** class helps to implement this feature. The **FilterEntry** property of this class associates a text box with the FlexGrid control. As soon as the user inputs a value in this text box, the FlexGrid control is filtered. However, you can alter this behavior to filter the text after entering complete value using **Mode** property of the **C1FullTextFilter** class, which accepts values from the **FullTextFilterMode** enumeration.

By default, filtering is applied to the data source of FlexGrid. In case a **DataCollection** is used as **FlexGrid's** data source, filtering is applied to the **DataCollection**, and this impacts all the controls bound to the same **DataCollection**.

Moreover, you can customize the filtering behavior using the **MatchCase**, **MatchNumbers**, **MatchWholeWord** or **TreatSpacesAsAndOperator** properties which allow you to filter data by matching case, whole word, number or treat space as AND operator respectively. In addition, you can also specify the delay time used to filter after the last typed character using the **Delay** property.

In XAML

The following markup binds a text box to search and filter across all columns on the grid:

XAML

```
<TextBox x:Name="filter" Margin="0, 40, 0, 0"/> <c1:C1FlexGrid Name="grid"
Grid.Row="1">
<c1:C1FlexGridFilterService.FlexGridFilter> <c1:C1FlexGridFilter />
</c1:C1FlexGridFilterService.FlexGridFilter>
<c1:C1FlexGridFilterService.FullTextFilterBehavior> <c1:C1FullTextFilter
FilterEntry="{Binding Source={x:Reference filter}}" MatchCase="True"/>
</c1:C1FlexGridFilterService.FullTextFilterBehavior> </c1:C1FlexGrid>
```

In Code

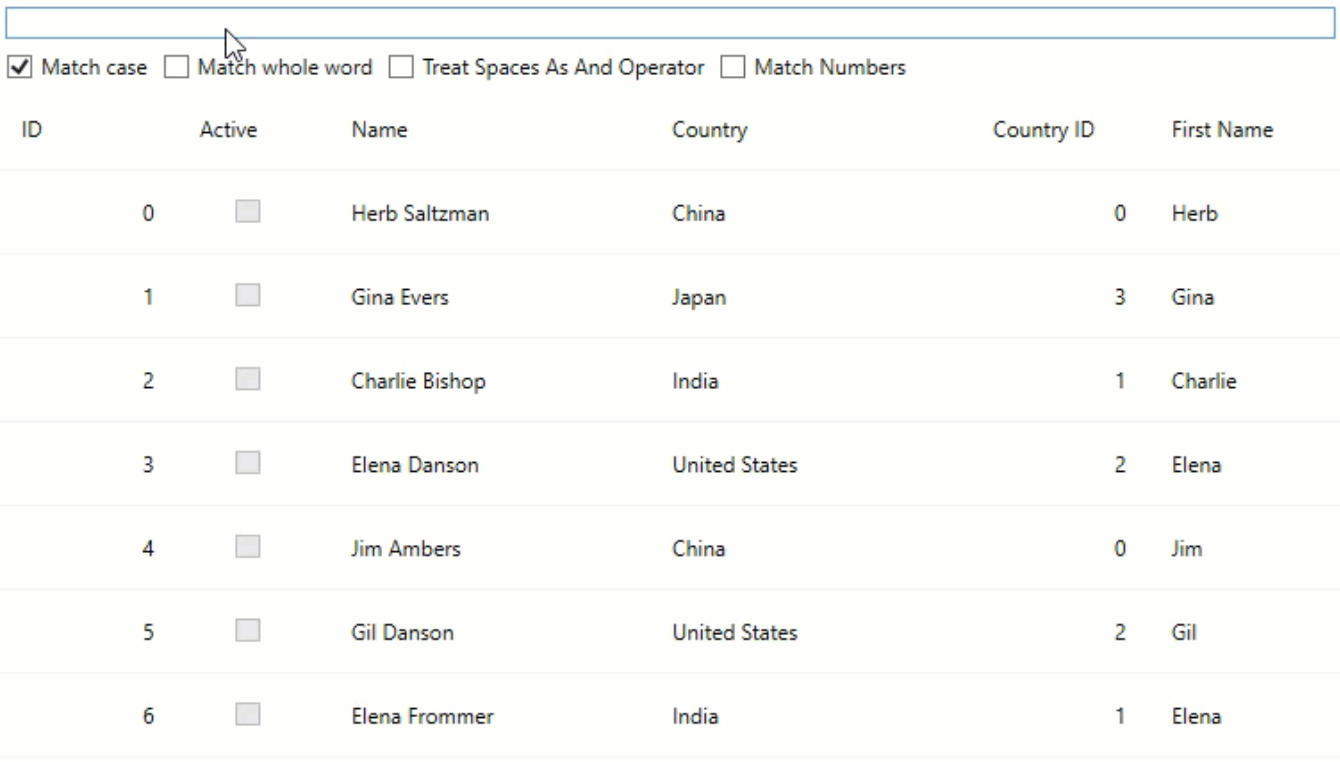
The following code sets the entry field and performs search and filtering across all columns on the grid:

```
Dim ftf As C1FullTextFilter = New C1FullTextFilter(grid)
ftf.FilterEntry = filter
ftf.MatchCase = True
ftf.Mode = C1FullTextFilter.FullTextFilterMode.WhenCompleted
```

```
C1FullTextFilter ftf = new C1FullTextFilter(grid);
ftf.FilterEntry = filter;
ftf.MatchCase = true;
ftf.Mode = C1FullTextFilter.FullTextFilterMode.WhenCompleted;
```

.NET

You can search data in FlexGrid using full text filtering feature wherein the grid's data source is filtered based on the user input to return the search results. The **FullTextFilterBehaviour** class helps to implement this feature. The **FilterEntry** property of this class associates a text box with the FlexGrid control. As soon as the user inputs a value in this text box, the FlexGrid control is filtered as shown in the GIF below.



However, you can alter this behavior to filter the text after entering complete value using **Mode** property of the **FullTextFilterBehaviour** class, which accepts values from the **FullTextFilterMode** enumeration.

By default, filtering is applied to the data source of FlexGrid. In case a **DataCollection** is used as **FlexGrid's** data source, filtering is applied to the **DataCollection**, and this impacts all the controls bound to the same **DataCollection**.

Moreover, you can customize the filtering behavior using the **MatchCase**, **MatchNumbers**, **MatchWholeWord** or **TreatSpacesAsAndOperator** properties which allow you to filter data by matching case, whole word, number or treat space as AND operator respectively. In addition, you can also specify the delay time used to filter after the last typed character using the **Delay** property.

In XAML

The following markup binds a text box to search and filter across all columns on the grid.

XAML

```
<TextBox x:Name="filter" Margin="4" />
<StackPanel Orientation="Horizontal" Grid.Row="1">
    <CheckBox x:Name="matchCaseCheckbox" Content="Match case" Margin="4"/>
    <CheckBox x:Name="matchWholeWordCheckBox" Content="Match whole word" Margin="4"
/>
    <CheckBox x:Name="treatSpacesAsAndOperator" Content="Treat Spaces As And
Operator" Margin="4" />
    <CheckBox x:Name="matchNumbers" Content="Match Numbers" Margin="4" />
</StackPanel>
```

```
<c1:FlexGrid x:Name="grid" IsReadOnly="True" Grid.Row="2">
    <i:Interaction.Behaviors>
        <c1:FullTextFilterBehavior FilterEntry="{Binding Source={x:Reference
filter}}"
                                   MatchWholeWord="{Binding IsChecked, Source={x:Reference
matchWholeWordCheckBox}}"
                                   MatchCase="{Binding IsChecked, Source={x:Reference
matchCaseCheckbox}}"
                                   TreatSpacesAsAndOperator="{Binding IsChecked, Source=
{x:Reference treatSpacesAsAndOperator}}"
                                   MatchNumbers="{Binding IsChecked, Source={x:Reference
matchNumbers}}" />
    </i:Interaction.Behaviors>
</c1:FlexGrid>
```

In Code

The following code sets the entry field and performs search and filtering across all columns on the grid.

C#

```
public FullTextFilter()
{
    InitializeComponent();

    var data = Customer.GetCustomerList(100);
    grid.ItemsSource = data;
    grid.MinColumnWidth = 85;
```

グリッド

パフォーマンスを改善する

WPFアプリケーションではごく一般的に使われる仮想化技術では、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。同じく、C1FlexGrid の場合も仮想化技術を活用してパフォーマンスを改善できます。

rows.DeferNotifications() を使用します

グリッドの RowCollection クラスを取得し、ICollectionView インターフェイスで使用したメカニズムと同じDeferNotifications を実装します。このメカニズムは、Windows フォームアプリケーションでよく使用される BeginUpdate/EndUpdate パターンに似ており、パフォーマンスの大幅な向上をもたらします。

次の例では、連結グリッドおよび非連結グリッドの場合 DeferNotifications メカニズムを使用してパフォーマンスを向上します。

非連結グリッドの場合、コードは次のようになります。

VisualBasic

```
Private Sub InitialUnboundFlexGrid()
    Dim rows = _flex1.Rows
    'グリッドの RowsCollection クラスを取得して
    'ICollectionView インターフェイスで使用したメカニズムと同じDeferNotifications を実装します
    Using rows.DeferNotifications()
        ' 非連結グリッドに行/列を追加します
        For i As Integer = 0 To 19
            _flex1.Columns.Add(New Column())
        Next
        For i As Integer = 0 To 499
            _flex1.Rows.Add(New Row())
        Next

        ' グリッドにデータを挿入します
        For r As Integer = 0 To _flex1.Rows.Count - 1
            For c As Integer = 0 To _flex1.Columns.Count - 1
                _flex1(r, c) = String.Format("cell [{0},{1}]", r, c)
            Next
        Next
    End Using
End Sub
```

C#

```
private void InitialUnboundFlexGrid()
{
    var rows = _flex1.Rows;
    //グリッドの RowsCollection クラスを取得して
    //ICollectionView インターフェイスで使用したメカニズムと同じDeferNotifications を実装します
    using (rows.DeferNotifications())
    {
        // 非連結グリッドに行/列を追加します
        for (int i = 0; i < 20; i++)
        {
            _flex1.Columns.Add(new Column());
        }
        for (int i = 0; i < 500; i++)
        {
            _flex1.Rows.Add(new Row());
        }

        // グリッドにデータを挿入します
        for (int r = 0; r < _flex1.Rows.Count; r++)
        {
            for (int c = 0; c < _flex1.Columns.Count; c++)
            {
                _flex1[r, c] = string.Format("cell [{0},{1}]", r, c);
            }
        }
    }
}
```

連結グリッドの場合、コードは次のようになります。

VisualBasic

```
Private Sub InitialBoundFlexGrid()
    Dim p = Product.GetProducts(100)
    _flex2.ItemsSource = p
    Dim view = _flex2.CollectionView
    ''using (view.DeferRefresh())' 文はオプションです。この文を追加すると、
    'すべてのグループが設定されるまでデータソースからの通知が保留になるため、パフォーマンスが向上します。
    Using view.DeferRefresh()
        view.GroupDescriptions.Clear()
        view.GroupDescriptions.Add(New PropertyGroupDescription("Rating"))
        view.GroupDescriptions.Add(New PropertyGroupDescription("Color"))
    End Using

End Sub
```

C#

```
private void InitialBoundFlexGrid()
{
    var p = Product.GetProducts(100);
    _flex.ItemsSource = p;
    var view = _flex.CollectionView;
    //using (view.DeferRefresh())' 文はオプションです。この文を追加すると、
    //すべてのグループが設定されるまでデータソースからの通知が保留になるため、パフォーマンスが向上します。
    using (view.DeferRefresh())
    {
        view.GroupDescriptions.Clear();
        view.GroupDescriptions.Add(new PropertyGroupDescription("Rating"));
        view.GroupDescriptions.Add(new PropertyGroupDescription("Color"));
    }
    _flex2.ItemsSource = view;
}
```

レイアウトを復元する

C1FlexGrid の **SaveColumnLayout/LoadColumnLayout** メソッドを使用してSystem.Xml.XmlReader から列のレイアウトを保存/ロードできます。このプロパティを使用すると、列のレイアウト文字列は、列の位置、表示/非表示、および幅を定義します。しかし、スタイルなどが維持されませんのでご注意ください。SaveColumnLayout/LoadColumnLayout メソッドを実現する方法は以下に示します。

【実行例】

Visual Basic

```
Partial Public Class MainWindow
    Inherits Window

    Public Sub New()
        InitializeComponent()

        Dim p = Product.GetProducts(50)
        _flex.ItemsSource = p
        'グリッドのスタイルを設定します
        _flex.Rows.Frozen = 1
        _flex.Columns.Frozen = 1
        _flex.Background = Brushes.YellowGreen
        _flex.Rows(0).Foreground = Brushes.Red
        _flex.Rows(0).Background = Brushes.Yellow
    End Sub

    Private Sub Save_Click(sender As Object, e As RoutedEventArgs)
        Dim settings As New XmlWriterSettings()
        settings.Indent = True
        settings.IndentChars = vbTab
        Dim writer As XmlWriter = XmlWriter.Create("products.xml", settings)
        _flex.SaveColumnLayout(writer)
        writer.Close()
        MessageBox.Show("正常に保存しました!")
    End Sub

    Private Sub Load_Click(sender As Object, e As RoutedEventArgs)
        Dim reader As XmlReader = XmlReader.Create("products.xml")
        _flex.LoadColumnLayout(reader)
        reader.Close()
    End Sub

End Class
```

C#

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        var p = Product.GetProducts(50);
        _flex.ItemsSource = p;
        _flex.Rows.Frozen = 1;
        _flex.Columns.Frozen = 1;
        _flex.Background = Brushes.YellowGreen;
        _flex.Rows[0].Foreground = Brushes.Red;
        _flex.Rows[0].Background = Brushes.Yellow;
    }

    private void Save_Click(object sender, RoutedEventArgs e)
    {
        XmlWriterSettings settings = new XmlWriterSettings();
        settings.Indent = true;
        settings.IndentChars = "\t";
        XmlWriter writer = XmlWriter.Create("producs.xml", settings);
        _flex.SaveColumnLayout(writer);
        writer.Close();
        MessageBox.Show("Save completed!");
    }

    private void Load_Click(object sender, RoutedEventArgs e)
    {
        XmlReader reader = XmlReader.Create("producs.xml");
        _flex.LoadColumnLayout(reader);
        reader.Close();
    }
}
```

エクスポート

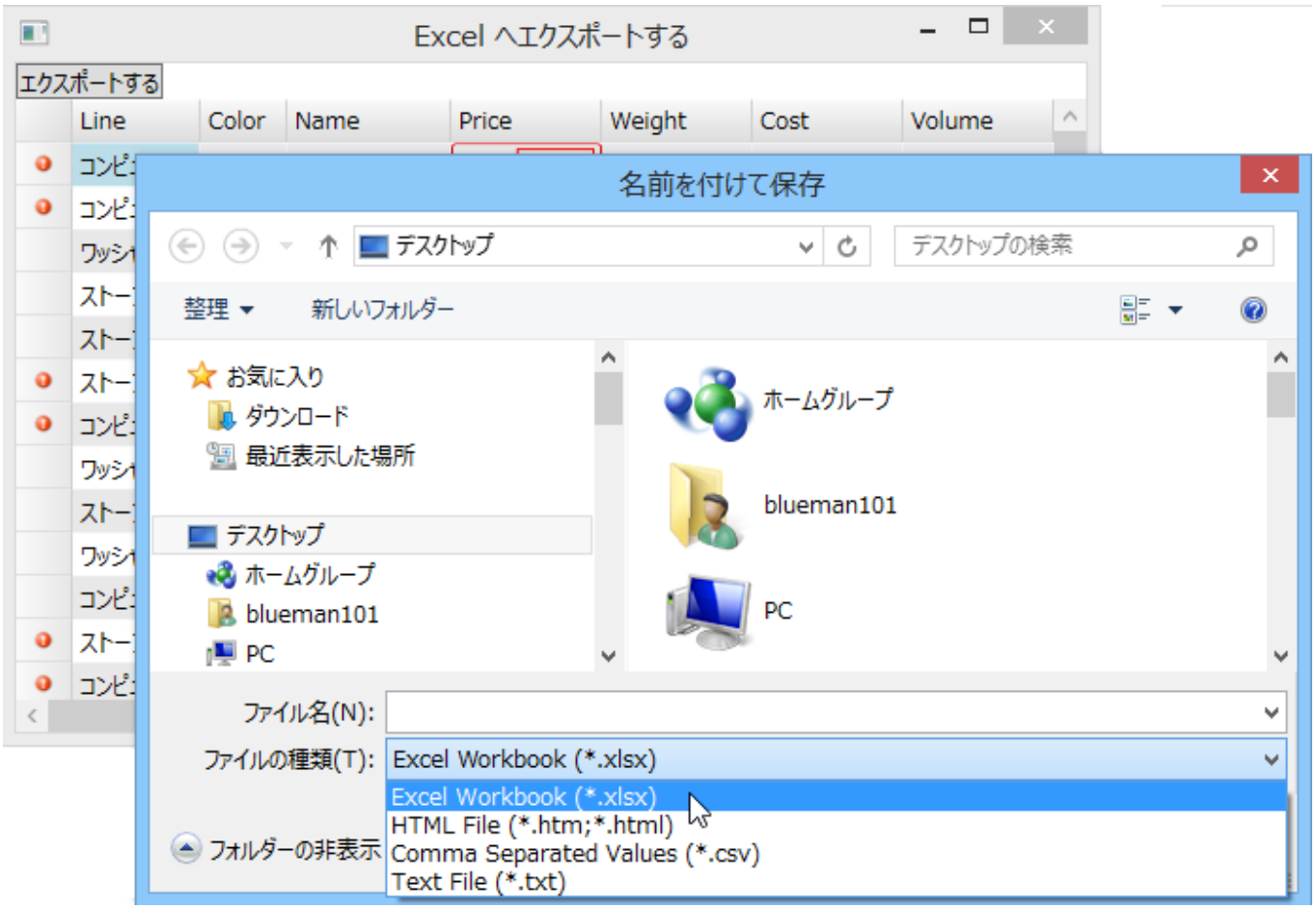
Excelファイルを保存する

C1FlexGrid の内容は、デフォルトで CSV、HTML形式またはプレーンテキストとしてエクスポートできますが、**C1Excel** を使用してグリッドのデータをExcel 形式にエクスポートすることも可能です。また、C1Excel ライブラリを使用してエクセル(XLSX形式)にもエクスポートできます。

次の例では、通常の連結グリッド C1FlexGrid に対して **ExcelFilter** のヘルパークラスを使用してグリッドの内容を XLSX 形式ファイルにエクスポートする方法を示します。基本的に、これらのヘルパークラスをプロジェクトに追加してクラスのメソッドを使用することでデータをエクセルファイルにエクスポートできます。

[実行例]

[Excelファイルを保存する]



サンプルコードは次のようになります。

C# サンプルコード

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        IEnumerable<Product> products = Product.GetProducts(250);
        products.ElementAt(0).Price = -10;
        products.ElementAt(1).Price = -10;

        C1FlexGrid1.ItemsSource = products;


    }

    private void btnExport_Click(object sender, RoutedEventArgs e)
    {
        var dlg = new Microsoft.Win32.SaveFileDialog();
        dlg.DefaultExt = "xlsx";
        dlg.Filter = "Excel Workbook (*.xlsx)|*.xlsx|" + "HTML File
(*.htm;*.html)|*.htm;*.html|" + "Comma Separated Values (*.csv)|*.csv|" + "Text
File (*.txt)|*.txt";
        if (dlg.ShowDialog() == true)
        {
            var ext = System.IO.Path.GetExtension(dlg.SafeFileName).ToLower();
            ext = ext == ".htm" ? "ehtm" : ext == ".html" ? "ehtm" : ext;
            switch (ext)
            {
                case "ehtm":
                    {
                        C1FlexGrid1.Save(dlg.FileName,
C1.WPF.FlexGrid.FileFormat.Html, SaveOptions.Formatted);
                        break;
                    }
                case ".csv":
                    {
                        C1FlexGrid1.Save(dlg.FileName,
C1.WPF.FlexGrid.FileFormat.Csv, SaveOptions.Formatted);
                        break;
                    }
                case ".txt":
                    {
                        C1FlexGrid1.Save(dlg.FileName,
C1.WPF.FlexGrid.FileFormat.Text, SaveOptions.Formatted);
                        break;
                    }
                default:
                    {
                        Save(dlg.FileName,C1FlexGrid1);
                        break;
```

```
                    }
                }
            }
        }

        public void Save(string filename, C1FlexGrid flexgrid)
        {
            // 保存するエクセルブックを作成します
            var book = new C1XLBook();
            book.Sheets.Clear();
            var  xlSheet = book.Sheets.Add("Sheet1");
            ExcelExport.ExcelFilter.Save(flexgrid, xlSheet);


            // エクセルブックを保存します
            book.Save(filename, C1.WPF.Excel.FileFormat.OpenXml);
        }
}
```

ExcelFilter.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Data;
using System.Diagnostics;
using C1.WPF.Excel;
using C1.WPF.FlexGrid;
using C1.WPF;

using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace ExcelExport
{

    /// <summary>
    /// XLSheetとC1FlexGridの間にデータを転送するための方法を提供するクラス
    /// </summary>
    internal sealed class ExcelFilter
    {
        private static C1XLBook _lastBook;
        private static Dictionary<XLStyle, ExcelCellStyle> _cellStyles = new
Dictionary<XLStyle, ExcelCellStyle>();

        private static Dictionary<ExcelCellStyle, XLStyle> _excelStyles = new
Dictionary<ExcelCellStyle, XLStyle>();
        //------------------------------------------------------------------------
--------
        #region "** object model"

        /// <summary>
        /// C1FlexGridのコンテンツをXLSheetに保存します
        /// </summary>
        public static void Save(C1FlexGrid flex, XLSheet sheet)
        {
            // 新しいbookの場合は、スタイルのキャッシュをクリアします
            if (!object.ReferenceEquals(sheet.Book, _lastBook))
            {
                _cellStyles.Clear();
                _excelStyles.Clear();
                _lastBook = sheet.Book;
            }
```

```
            // グローバルパラメーターを保存します
            sheet.DefaultRowHeight = PixelsToTwips(flex.Rows.DefaultSize);
            sheet.DefaultColumnWidth = PixelsToTwips(flex.Columns.DefaultSize);
            sheet.Locked = flex.IsReadOnly;
            sheet.ShowGridLines = flex.GridLinesVisibility !=
GridLinesVisibility.None;
            sheet.ShowHeaders = flex.HeadersVisibility != HeadersVisibility.None;
            sheet.OutlinesBelow = flex.GroupRowPosition ==
GroupRowPosition.BelowData;

            // 列を保存します
            sheet.Columns.Clear();
            foreach (Column col in flex.Columns)
            {
                dynamic c = sheet.Columns.Add();
                if (!col.Width.IsAuto)
                {
                    c.Width = PixelsToTwips(col.ActualWidth);
                }
                c.Visible = col.Visible;
                if (col.CellStyle is ExcelCellStyle)
                {
                    c.Style = GetXLStyle(flex, sheet,
(ExcelCellStyle)col.CellStyle);
                }
            }

            sheet.Rows.Clear();

            // 列ヘッダーを保存します
            XLStyle headerStyle = default(XLStyle);
            headerStyle = new XLStyle(sheet.Book);
            headerStyle.Font = new XLFont("Arial", 10, true, false);

            foreach (Row row in flex.ColumnHeaders.Rows)
            {
                dynamic r = sheet.Rows.Add();
                if (row.Height > -1)
                {
                    r.Height = PixelsToTwips(row.Height);
                }
                if (row.CellStyle is ExcelCellStyle)
                {
                    r.Style = GetXLStyle(flex, sheet,
(ExcelCellStyle)row.CellStyle);
                }
                if (row is ExcelRow)
                {
                    r.OutlineLevel = ((ExcelRow)row).Level;
                }
                for (int c = 0; c <= flex.ColumnHeaders.Columns.Count - 1; c++)
                {
```

```
                    // セル値を保存します
                    dynamic cell = sheet[row.Index, c];
                    string colHeader = flex.ColumnHeaders[row.Index, c] != null ?
flex.ColumnHeaders[row.Index, c].ToString() : flex.Columns[c].ColumnName;
                    cell.Value = colHeader;

                    // 列ヘッダーを太字にします
                    cell.Style = headerStyle;

                }
                r.Visible = row.Visible;
            }



            // 行を保存します
            foreach (Row row in flex.Rows)
            {
                dynamic r = sheet.Rows.Add();
                if (row.Height > -1)
                {
                    r.Height = PixelsToTwips(row.Height);
                }
                if (row.CellStyle is ExcelCellStyle)
                {
                    r.Style = GetXLStyle(flex, sheet,
(ExcelCellStyle)row.CellStyle);
                }
                if (row is ExcelRow)
                {
                    r.OutlineLevel = ((ExcelRow)row).Level;
                }
                r.Visible = row.Visible;
            }

            // セルを保存します
            for (int r = flex.ColumnHeaders.Rows.Count - 1; r <= flex.Rows.Count -
1; r++)
            {
                for (int c = 0; c <= flex.Columns.Count - 1; c++)
                {
                    // セル値を保存します
                    dynamic cell = sheet[r + 1, c];
                    dynamic obj = flex[r, c];
                    cell.Value = obj is FrameworkElement ? 0 : obj;

                    // セルの数式とスタイルを保存します
                    dynamic row = flex.Rows[r] as ExcelRow;
                    if (row != null)
                    {
                        // セルの数式を保存します
                        dynamic col = flex.Columns[c];
```

```
                        // セルのスタイルを保存します
                        dynamic cs = row.GetCellStyle(col) as ExcelCellStyle;
                        if (cs != null)
                        {
                            cell.Style = GetXLStyle(flex, sheet, cs);
                        }
                    }
                }
            }

            // 選択範囲を保存します
            dynamic sel = flex.Selection;
            if (sel.IsValid)
            {
                dynamic xlSel = new XLCellRange(sheet, sel.Row, sel.Row2,
sel.Column, sel.Column2);
                sheet.SelectedCells.Clear();
                sheet.SelectedCells.Add(xlSel);
            }
        }

        #endregion

        //------------------------------------------------------------------------
--------
        #region "** implementation"

        private static double TwipsToPixels(double twips)
        {
            return Convert.ToInt32(twips / 1440.0 * 96.0 * 1.2 + 0.5);
        }
        private static int PixelsToTwips(double pixels)
        {
            return Convert.ToInt32(pixels * 1440.0 / 96.0 / 1.2 + 0.5);
        }
        private static double PointsToPixels(double points)
        {
            return points / 72.0 * 96.0 * 1.2;
        }
        private static double PixelsToPoints(double pixels)
        {
            return pixels * 72.0 / 96.0 / 1.2;
        }

        // Excelスタイルをクリッドスタイルに変更します
        private static ExcelCellStyle GetCellStyle(XLStyle x)
        {
            // キャッシュを検索します
            ExcelCellStyle s = default(ExcelCellStyle);
            if (_cellStyles.TryGetValue(x, out s))
            {
                return s;
```

```
}

// 見つかりません。スタイルを作成します
s = new ExcelCellStyle();

// 配置
switch (x.AlignHorz)
{
    case XLAlignHorzEnum.Left:
        s.HorizontalAlignment = HorizontalAlignment.Left;
        break;
    case XLAlignHorzEnum.Center:
        s.HorizontalAlignment = HorizontalAlignment.Center;
        break;
    case XLAlignHorzEnum.Right:
        s.HorizontalAlignment = HorizontalAlignment.Right;
        break;
}
switch (x.AlignVert)
{
    case XLAlignVertEnum.Top:
        s.VerticalAlignment = VerticalAlignment.Top;
        break;
    case XLAlignVertEnum.Center:
        s.VerticalAlignment = VerticalAlignment.Center;
        break;
    case XLAlignVertEnum.Bottom:
        s.VerticalAlignment = VerticalAlignment.Bottom;
        break;
}
s.TextWrapping = x.WordWrap;

// カラー
if (x.BackPattern == XLPatternEnum.Solid && IsColorValid(x.BackColor))
{
    s.Background = new SolidColorBrush(x.BackColor);
}
if (IsColorValid(x.ForeColor))
{
    s.Foreground = new SolidColorBrush(x.ForeColor);
}

// フォント
dynamic font = x.Font;
if (font != null)
{
    s.FontFamily = new FontFamily(font.FontName);
    s.FontSize = PointsToPixels(font.FontSize);
    if (font.Bold)
    {
        s.FontWeight = FontWeights.Bold;
    }
```

```
                if (font.Italic)
                {
                    s.FontStyle = FontStyles.Italic;
                }
                if (font.Underline != XLUnderlineStyle.None)
                {
                    s.TextDecorations = TextDecorations.Underline;
                }
            }

            // 書式
            if (!string.IsNullOrEmpty(x.Format))
            {
                s.Format = XLStyle.FormatXLToDotNet(x.Format);
            }

            // 境界線
            s.CellBorderThickness = new Thickness(GetBorderThickness(x.BorderLeft),
GetBorderThickness(x.BorderTop), GetBorderThickness(x.BorderRight),
GetBorderThickness(x.BorderBottom));
            s.CellBorderBrushLeft = GetBorderBrush(x.BorderColorLeft);
            s.CellBorderBrushTop = GetBorderBrush(x.BorderColorTop);
            s.CellBorderBrushRight = GetBorderBrush(x.BorderColorRight);
            s.CellBorderBrushBottom = GetBorderBrush(x.BorderColorBottom);

            // キャッシュに保存して戻します
            _cellStyles[x] = s;
            return s;
        }

        // グリッドスタイルをExcelスタイルに変更します
        private static XLStyle GetXLStyle(C1FlexGrid flex, XLSheet sheet,
ExcelCellStyle s)
        {
            // キャッシュで検索します
            XLStyle x = default(XLStyle);
            if (_excelStyles.TryGetValue(s,out  x))
            {
                return x;
            }

            // 見つかりません。スタイルを作成します
            x = new XLStyle(sheet.Book);

            // 配置
            if (s.HorizontalAlignment.HasValue)
            {
                switch (s.HorizontalAlignment.Value)
                {
                    case HorizontalAlignment.Left:
                        x.AlignHorz = XLAlignHorzEnum.Left;
                        break;
                    case HorizontalAlignment.Center:
```

```
                x.AlignHorz = XLAlignHorzEnum.Center;
                break;
            case HorizontalAlignment.Right:
                x.AlignHorz = XLAlignHorzEnum.Right;
                break;
        }
    }
    if (s.VerticalAlignment.HasValue)
    {
        switch (s.VerticalAlignment.Value)
        {
            case VerticalAlignment.Top:
                x.AlignVert = XLAlignVertEnum.Top;
                break;
            case VerticalAlignment.Center:
                x.AlignVert = XLAlignVertEnum.Center;
                break;
            case VerticalAlignment.Bottom:
                x.AlignVert = XLAlignVertEnum.Bottom;
                break;
        }
    }
    if (s.TextWrapping.HasValue)
    {
        x.WordWrap = s.TextWrapping.Value;
    }

    // カラー
    if (s.Background is SolidColorBrush)
    {
        x.BackColor = ((SolidColorBrush)s.Background).Color;
        x.BackPattern = XLPatternEnum.Solid;
    }
    if (s.Foreground is SolidColorBrush)
    {
        x.ForeColor = ((SolidColorBrush)s.Foreground).Color;
    }

    // フォント
    dynamic fontName = flex.FontFamily.Source;
    dynamic fontSize = flex.FontSize;
    dynamic bold = false;
    dynamic italic = false;
    bool underline = false;
    bool hasFont = false;
    if (s.FontFamily != null)
    {
        fontName = s.FontFamily.Source;
        hasFont = true;
    }
    if (s.FontSize.HasValue)
    {
```

```
                fontSize = s.FontSize.Value;
                hasFont = true;
            }
            if (s.FontWeight.HasValue)
            {
                bold = s.FontWeight.Value == FontWeights.Bold || s.FontWeight.Value
== FontWeights.ExtraBold || s.FontWeight.Value == FontWeights.SemiBold;
                hasFont = true;
            }
            if (s.FontStyle.HasValue)
            {
                italic = s.FontStyle.Value == FontStyles.Italic;
                hasFont = true;
            }
            if (s.TextDecorations != null)
            {
                underline = true;
                hasFont = true;
            }
            if (hasFont)
            {
                fontSize = PixelsToPoints(fontSize);
                if (underline)
                {
                    dynamic color = Colors.Black;
                    if (flex.Foreground is SolidColorBrush)
                    {
                        color = ((SolidColorBrush)flex.Foreground).Color;
                    }
                    if (s.Foreground is SolidColorBrush)
                    {
                        color = ((SolidColorBrush)s.Foreground).Color;
                    }
                    x.Font = new XLFont(fontName, Convert.ToSingle(fontSize), bold,
italic, false, XLFontScript.None, XLUnderlineStyle.Single, color);
                }
                else
                {
                    x.Font = new XLFont(fontName, Convert.ToSingle(fontSize), bold,
italic);
                }
            }

            // 書式
            if (!string.IsNullOrEmpty(s.Format))
            {
                x.Format = XLStyle.FormatDotNetToXL(s.Format);
            }

            // 境界線
            if (s.CellBorderThickness.Left > 0 && s.CellBorderBrushLeft is
SolidColorBrush)
            {
```

```
                x.BorderLeft = GetBorderLineStyle(s.CellBorderThickness.Left);
                x.BorderColorLeft = ((SolidColorBrush)s.CellBorderBrushLeft).Color;
            }
            if (s.CellBorderThickness.Top > 0 && s.CellBorderBrushTop is
SolidColorBrush)
            {
                x.BorderTop = GetBorderLineStyle(s.CellBorderThickness.Top);
                x.BorderColorTop = ((SolidColorBrush)s.CellBorderBrushTop).Color;
            }
            if (s.CellBorderThickness.Right > 0 && s.CellBorderBrushRight is
SolidColorBrush)
            {
                x.BorderRight = GetBorderLineStyle(s.CellBorderThickness.Right);
                x.BorderColorRight =
((SolidColorBrush)s.CellBorderBrushRight).Color;
            }
            if (s.CellBorderThickness.Bottom > 0 && s.CellBorderBrushBottom is
SolidColorBrush)
            {
                x.BorderBottom = GetBorderLineStyle(s.CellBorderThickness.Bottom);
                x.BorderColorBottom =
((SolidColorBrush)s.CellBorderBrushBottom).Color;
            }

            // キャッシュに保存して返します
            _excelStyles[s] = x;
            return x;
        }
        private static double GetBorderThickness(XLLineStyleEnum ls)
        {
            switch (ls)
            {
                case XLLineStyleEnum.None:
                    return 0;
                case XLLineStyleEnum.Hair:
                    return 0.5;
                case XLLineStyleEnum.Thin:
                case XLLineStyleEnum.ThinDashDotDotted:
                case XLLineStyleEnum.ThinDashDotted:
                case XLLineStyleEnum.Dashed:
                case XLLineStyleEnum.Dotted:
                    return 1;
                case XLLineStyleEnum.Medium:
                case XLLineStyleEnum.MediumDashDotDotted:
                case XLLineStyleEnum.MediumDashDotted:
                case XLLineStyleEnum.MediumDashed:
                case XLLineStyleEnum.SlantedMediumDashDotted:
                    return 2;
                case XLLineStyleEnum.Double:
                case XLLineStyleEnum.Thick:
                    return 3;
            }
```

```
            return 0;
        }
        private static XLLineStyleEnum GetBorderLineStyle(double t)
        {
            if (t == 0)
            {
                return XLLineStyleEnum.None;
            }
            if (t < 1)
            {
                return XLLineStyleEnum.Hair;
            }
            if (t < 2)
            {
                return XLLineStyleEnum.Thin;
            }
            if (t < 3)
            {
                return XLLineStyleEnum.Medium;
            }
            return XLLineStyleEnum.Thick;
        }
        private static Brush GetBorderBrush(Color color)
        {
            return IsColorValid(color) ? new SolidColorBrush(color) : null;
        }
        private static bool IsColorValid(Color color)
        {
            return color.A > 0;
            // == 0xff;
        }

        #endregion
    }
}
```

ExcelCellStyle.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Data;
using System.Diagnostics;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.ComponentModel;
using C1.WPF.FlexGrid;

namespace ExcelExport
{
    /// <summary>
    /// Excelスタイルのセル境界線と書式指定文字列を提供するためのCellStyleクラスを拡張します
    /// </summary>
    public class ExcelCellStyle : CellStyle
    {
        // ** フィールド
        private string _format;
        private Thickness _bdrThickness;
        private Brush _bdrLeft;
        private Brush _bdrTop;
        private Brush _bdrRight;
        private Brush _bdrBottom;

        private static Thickness _thicknessEmpty = new Thickness(0);
        // ** オブジェクト・モデル
        public string Format
        {
            get { return _format; }
            set
            {
                if (value != _format)
                {
                    _format = value;
                    OnPropertyChanged(new PropertyChangedEventArgs("Format"));
                }
            }
        }
        public Thickness CellBorderThickness
        {
            get { return _bdrThickness; }
            set
            {
```

```
                if (value != _bdrThickness)
                {
                    _bdrThickness = value;
                    OnPropertyChanged(new
PropertyChangedEventArgs("BorderThickness"));
                }
            }
        }
        public Brush CellBorderBrushLeft
        {
            get { return _bdrLeft; }
            set
            {
                if (!object.ReferenceEquals(value, _bdrLeft))
                {
                    _bdrLeft = value;
                    OnPropertyChanged(new
PropertyChangedEventArgs("BorderColorLeft"));
                }
            }
        }
        public Brush CellBorderBrushTop
        {
            get { return _bdrTop; }
            set
            {
                if (!object.ReferenceEquals(value, _bdrTop))
                {
                    _bdrTop = value;
                    OnPropertyChanged(new
PropertyChangedEventArgs("BorderColorTop"));
                }
            }
        }
        public Brush CellBorderBrushRight
        {
            get { return _bdrRight; }
            set
            {
                if (!object.ReferenceEquals(value, _bdrRight))
                {
                    _bdrRight = value;
                    OnPropertyChanged(new
PropertyChangedEventArgs("BorderColorRight"));
                }
            }
        }
        public Brush CellBorderBrushBottom
        {
            get { return _bdrBottom; }
            set
            {
                if (!object.ReferenceEquals(value, _bdrBottom))
```

```
                {
                    _bdrBottom = value;
                    OnPropertyChanged(new
PropertyChangedEventArgs("BorderColorBottom"));
                }
            }
        }

        // ** オーバーライド
        public override void Apply(Border bdr, SelectedState selState)
        {
            base.Apply(bdr, selState);
            ApplyBorder(bdr, _bdrLeft, new Thickness(_bdrThickness.Left, 0, 0, 0));
            ApplyBorder(bdr, _bdrTop, new Thickness(0, _bdrThickness.Top, 0, 0));
            ApplyBorder(bdr, _bdrRight, new Thickness(0, 0, _bdrThickness.Right,
0));
            ApplyBorder(bdr, _bdrBottom, new Thickness(0, 0, 0,
_bdrThickness.Bottom));
        }
        private void ApplyBorder(Border bdr, Brush br, Thickness t)
        {
            if (br != null && t != _thicknessEmpty)
            {
                // 内部の境界線を作成します
                dynamic inner = new Border();
                inner.BorderThickness = t;
                inner.BorderBrush = br;

                // コンテンツに拡張します
                dynamic content = bdr.Child;
                bdr.Child = inner;
                inner.Child = content;

                // パディングします
                inner.Padding = bdr.Padding;
                bdr.Padding = _thicknessEmpty;
            }
        }
    }
}
```

ExcelRow.cs

```
using Microsoft.VisualBasic;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Data;
using System.Diagnostics;
using C1.WPF.FlexGrid;
using C1.WPF.Excel;

using System.Globalization;
using System.Windows;

namespace ExcelExport
{

    /// <summary>
    /// 編集可能、ツリー ノードとして使用でき、各列と関連するセルの
    /// スタイルのコレクションを保守するグリッド行
    /// </summary>
    public class ExcelRow : GroupRow
    {
        // ** フィールド

        private Dictionary<Column, CellStyle> _cellStyles;
        // 既定の有効桁数を6桁に設定します
        private const string DEFAULT_FORMAT = "#,##0.######";

        // ** ctor
        public ExcelRow(ExcelRow styleRow)
        {
            IsReadOnly = false;
            if (styleRow != null && styleRow.Grid != null) {
                foreach (var c in styleRow.Grid.Columns) {
                    dynamic cs = styleRow.GetCellStyle(c);
                    if (cs != null) {
                        this.SetCellStyle(c, cs.Clone());
                    }
                }
            }
        }
        public ExcelRow()
            : this(null)
        {
        }

        // ** オブジェクト・モデル

        /// <summary>
        /// データを取得する場合、書式を適用するためにオーバーライドされる
```

```
        /// </summary>
        public override string GetDataFormatted(Column col)
        {
            // データを取得します
            dynamic data = GetDataRaw(col);

            // 書式を適用します
            dynamic ifmt = data as IFormattable;
            if (ifmt != null)
            {
                // セルの書式を取得します
                dynamic s = GetCellStyle(col) as ExcelCellStyle;
                dynamic fmt = s != null && (!string.IsNullOrEmpty(s.Format)) ?
s.Format : DEFAULT_FORMAT;
                data = ifmt.ToString(fmt, CultureInfo.CurrentUICulture);
            }

            // 完了
            return data != null ? data.ToString() : string.Empty;
        }

        // ** オブジェクト・モデル

        /// <summary>
        /// この行では、セルにスタイルを適用します
        /// </summary>
        public void SetCellStyle(Column col, CellStyle style)
        {
            if (!object.ReferenceEquals(style, GetCellStyle(col)))
            {
                if (_cellStyles == null)
                {
                    _cellStyles = new Dictionary<Column, CellStyle>();
                }
               _cellStyles[col] = style;
                if (Grid != null)
                {
                    Grid.Invalidate(new CellRange(this.Index, col.Index));
                }
            }
        }
        /// <summary>
        /// この行では、セルに適用したスタイルを取得します
        /// </summary>
        public CellStyle GetCellStyle(Column col)
        {
            CellStyle s = null;
            if (_cellStyles != null)
            {
                _cellStyles.TryGetValue(col,out s);
            }
            return s;
        }
```

```
        }
}
```

VB サンプルコード

```
Class MainWindow
    Inherits Window
    Public Sub New()
        InitializeComponent()
        Dim products As ListCollectionView = Product.GetProducts(250)
        TryCast(products.GetItemAt(0), Product).Price = -10
        TryCast(products.GetItemAt(1), Product).Price = -10

        C1FlexGrid1.ItemsSource = products

    End Sub

    Private Sub btnExport_Click(sender As Object, e As RoutedEventArgs)
        Dim dlg = New Microsoft.Win32.SaveFileDialog()
        dlg.DefaultExt = "xlsx"
        dlg.Filter = "Excel Workbook (*.xlsx)|*.xlsx|" + "HTML File
(*.htm;*.html)|*.htm;*.html|" + "Comma Separated Values (*.csv)|*.csv|" + "Text
File (*.txt)|*.txt"
        If dlg.ShowDialog() = True Then
            Dim ext = System.IO.Path.GetExtension(dlg.SafeFileName).ToLower()
            ext = If(ext = ".htm", "ehtm", If(ext = ".html", "ehtm", ext))
            Select Case ext
                Case "ehtm"
                    If True Then
                        C1FlexGrid1.Save(dlg.FileName,
C1.WPF.FlexGrid.FileFormat.Html, C1.WPF.FlexGrid.SaveOptions.Formatted)
                        Exit Select
                    End If
                Case ".csv"
                    If True Then
                        C1FlexGrid1.Save(dlg.FileName,
C1.WPF.FlexGrid.FileFormat.Csv, C1.WPF.FlexGrid.SaveOptions.Formatted)
                        Exit Select
                    End If
                Case ".txt"
                    If True Then
                        C1FlexGrid1.Save(dlg.FileName,
C1.WPF.FlexGrid.FileFormat.Text, C1.WPF.FlexGrid.SaveOptions.Formatted)
                        Exit Select
                    End If
                Case Else
                    If True Then
                        Save(dlg.FileName, C1FlexGrid1)
                        Exit Select
                    End If
            End Select
        End If
    End Sub
```

```
    Public Sub Save(filename As String, flexgrid As C1.WPF.FlexGrid.C1FlexGrid)
        ' 保存するBookを作成します
        Dim book = New C1XLBook()
        book.Sheets.Clear()
        Dim xlSheet = book.Sheets.Add("Sheet1")
        ExcelFilter.Save(flexgrid, xlSheet)


        ' Bookを保存します
        book.Save(filename, C1.WPF.Excel.FileFormat.OpenXml)
    End Sub

End Class
```

ExcelFilter.vb

```
Imports C1.WPF.Excel
Imports C1.WPF.FlexGrid

Friend NotInheritable Class ExcelFilter
    Private Shared _lastBook As C1XLBook
    Private Shared _cellStyles As New Dictionary(Of XLStyle, ExcelCellStyle)()

    Private Shared _excelStyles As New Dictionary(Of ExcelCellStyle, XLStyle)()
    '-------------------------------------------------------------------------------
---
#Region "** object model"

    ''' <summary>
    ''' C1FlexGridのコンテンツをXLSheetに保存します
    ''' </summary>
    Public Shared Sub Save(flex As C1FlexGrid, sheet As XLSheet)
        ' 新しいbookの場合は、スタイルのキャッシュをクリアします
        If Not Object.ReferenceEquals(sheet.Book, _lastBook) Then
            _cellStyles.Clear()
            _excelStyles.Clear()
            _lastBook = sheet.Book
        End If

        ' グローバルパラメーターを保存します
        sheet.DefaultRowHeight = PixelsToTwips(flex.Rows.DefaultSize)
        sheet.DefaultColumnWidth = PixelsToTwips(flex.Columns.DefaultSize)
        sheet.Locked = flex.IsReadOnly
        sheet.ShowGridLines = flex.GridLinesVisibility <> GridLinesVisibility.None
        sheet.ShowHeaders = flex.HeadersVisibility <> HeadersVisibility.None
        sheet.OutlinesBelow = flex.GroupRowPosition = GroupRowPosition.BelowData

        ' 列を保存します
        sheet.Columns.Clear()
        For Each col As Column In flex.Columns
            Dim c As XLColumn = sheet.Columns.Add()
            If Not col.Width.IsAuto Then
                c.Width = PixelsToTwips(col.ActualWidth)
            End If
            c.Visible = col.Visible
            If TypeOf col.CellStyle Is ExcelCellStyle Then
                c.Style = GetXLStyle(flex, sheet, DirectCast(col.CellStyle,
ExcelCellStyle))
            End If
        Next

        sheet.Rows.Clear()

        ' 列ヘッダーを保存します
        Dim headerStyle As XLStyle = Nothing
```

```
        headerStyle = New XLStyle(sheet.Book)
        headerStyle.Font = New XLFont("Arial", 10, True, False)

        For Each row As Row In flex.ColumnHeaders.Rows
            Dim r As XLRow = sheet.Rows.Add()
            If row.Height > -1 Then
                r.Height = PixelsToTwips(row.Height)
            End If
            If TypeOf row.CellStyle Is ExcelCellStyle Then
                r.Style = GetXLStyle(flex, sheet, DirectCast(row.CellStyle,
ExcelCellStyle))
            End If
            If TypeOf row Is ExcelRow Then
                r.OutlineLevel = DirectCast(row, ExcelRow).Level
            End If
            For c As Integer = 0 To flex.ColumnHeaders.Columns.Count - 1
                ' セル値を保存します
                Dim cell As XLCell = sheet(row.Index, c)
                Dim colHeader As String = If(flex.ColumnHeaders(row.Index, c) IsNot
Nothing, flex.ColumnHeaders(row.Index, c).ToString(), flex.Columns(c).ColumnName)
                cell.Value = colHeader

                ' 列ヘッダーを太字にします

                cell.Style = headerStyle
            Next
            r.Visible = row.Visible
        Next



        ' 行を保存します
        For Each row As Row In flex.Rows
            Dim r As XLRow = sheet.Rows.Add()
            If row.Height > -1 Then
                r.Height = PixelsToTwips(row.Height)
            End If
            If TypeOf row.CellStyle Is ExcelCellStyle Then
                r.Style = GetXLStyle(flex, sheet, DirectCast(row.CellStyle,
ExcelCellStyle))
            End If
            If TypeOf row Is ExcelRow Then
                r.OutlineLevel = DirectCast(row, ExcelRow).Level
            End If
            r.Visible = row.Visible
        Next

        ' 選択範囲を保存します
        For r As Integer = flex.ColumnHeaders.Rows.Count To flex.Rows.Count - 1
            For c As Integer = 0 To flex.Columns.Count - 1
                ' セル値を保存します
                Dim cell As XLCell = sheet(r, c)
                Dim obj As Object = flex(r, c)
```

```
                cell.Value = If(TypeOf obj Is FrameworkElement, 0, obj)

                ' セルの数式とスタイルを保存します
                Dim row As ExcelRow = TryCast(flex.Rows(r), ExcelRow)
                If row IsNot Nothing Then
                    ' セルの数式を保存します
                    Dim col As Column = flex.Columns(c)

                    ' セルのスタイルを保存します
                    Dim cs As ExcelCellStyle = TryCast(row.GetCellStyle(col),
ExcelCellStyle)
                    If cs IsNot Nothing Then
                        cell.Style = GetXLStyle(flex, sheet, cs)
                    End If
                End If
            Next
        Next

        ' 選択範囲を保存します
        Dim sel As CellRange = flex.Selection
        If sel.IsValid Then
            Dim xlSel As XLCellRange = New XLCellRange(sheet, sel.Row, sel.Row2,
sel.Column, sel.Column2)
            sheet.SelectedCells.Clear()
            sheet.SelectedCells.Add(xlSel)
        End If
    End Sub

#End Region

    '-------------------------------------------------------------------------
---
#Region "** implementation"

    Private Shared Function TwipsToPixels(twips As Double) As Double
        Return Convert.ToInt32(twips / 1440.0 * 96.0 * 1.2 + 0.5)
    End Function
    Private Shared Function PixelsToTwips(pixels As Double) As Integer
        Return Convert.ToInt32(pixels * 1440.0 / 96.0 / 1.2 + 0.5)
    End Function
    Private Shared Function PointsToPixels(points As Double) As Double
        Return points / 72.0 * 96.0 * 1.2
    End Function
    Private Shared Function PixelsToPoints(pixels As Double) As Double
        Return pixels * 72.0 / 96.0 / 1.2
    End Function

    ' Excelスタイルをクリッドスタイルに変更します
    Private Shared Function GetCellStyle(x As XLStyle) As ExcelCellStyle
        ' キャッシュを検索します
        Dim s As ExcelCellStyle = Nothing
        If _cellStyles.TryGetValue(x, s) Then
```

```
            Return s
        End If

        ' 見つかりません。スタイルを作成します
        s = New ExcelCellStyle()

        ' 配置
        Select Case x.AlignHorz
            Case XLAlignHorzEnum.Left
                s.HorizontalAlignment = HorizontalAlignment.Left
                Exit Select
            Case XLAlignHorzEnum.Center
                s.HorizontalAlignment = HorizontalAlignment.Center
                Exit Select
            Case XLAlignHorzEnum.Right
                s.HorizontalAlignment = HorizontalAlignment.Right
                Exit Select
        End Select
        Select Case x.AlignVert
            Case XLAlignVertEnum.Top
                s.VerticalAlignment = VerticalAlignment.Top
                Exit Select
            Case XLAlignVertEnum.Center
                s.VerticalAlignment = VerticalAlignment.Center
                Exit Select
            Case XLAlignVertEnum.Bottom
                s.VerticalAlignment = VerticalAlignment.Bottom
                Exit Select
        End Select
        s.TextWrapping = x.WordWrap

        ' カラー
        If x.BackPattern = XLPatternEnum.Solid AndAlso IsColorValid(x.BackColor)
Then
            s.Background = New SolidColorBrush(x.BackColor)
        End If
        If IsColorValid(x.ForeColor) Then
            s.Foreground = New SolidColorBrush(x.ForeColor)
        End If

        ' フォント
        Dim font As XLFont = x.Font
        If font IsNot Nothing Then
            s.FontFamily = New FontFamily(font.FontName)
            s.FontSize = PointsToPixels(font.FontSize)
            If font.Bold Then
                s.FontWeight = FontWeights.Bold
            End If
            If font.Italic Then
                s.FontStyle = FontStyles.Italic
            End If
            If font.Underline <> XLUnderlineStyle.None Then
                s.TextDecorations = TextDecorations.Underline
```

```
            End If
        End If

        ' 書式
        If Not String.IsNullOrEmpty(x.Format) Then
            s.Format = XLStyle.FormatXLToDotNet(x.Format)
        End If

        ' 境界線
        s.CellBorderThickness = New Thickness(GetBorderThickness(x.BorderLeft),
GetBorderThickness(x.BorderTop), GetBorderThickness(x.BorderRight),
GetBorderThickness(x.BorderBottom))
        s.CellBorderBrushLeft = GetBorderBrush(x.BorderColorLeft)
        s.CellBorderBrushTop = GetBorderBrush(x.BorderColorTop)
        s.CellBorderBrushRight = GetBorderBrush(x.BorderColorRight)
        s.CellBorderBrushBottom = GetBorderBrush(x.BorderColorBottom)

        ' キャッシュに保存して戻します
        _cellStyles(x) = s
        Return s
    End Function

    ' グリッドスタイルをExcelスタイルに変更します
    Private Shared Function GetXLStyle(flex As C1FlexGrid, sheet As XLSheet, s As
ExcelCellStyle) As XLStyle
        ' キャッシュで検索します
        Dim x As XLStyle = Nothing
        If _excelStyles.TryGetValue(s, x) Then
            Return x
        End If

        ' 見つかりません。スタイルを作成します
        x = New XLStyle(sheet.Book)

        ' 配置
        If s.HorizontalAlignment.HasValue Then
            Select Case s.HorizontalAlignment.Value
                Case HorizontalAlignment.Left
                    x.AlignHorz = XLAlignHorzEnum.Left
                    Exit Select
                Case HorizontalAlignment.Center
                    x.AlignHorz = XLAlignHorzEnum.Center
                    Exit Select
                Case HorizontalAlignment.Right
                    x.AlignHorz = XLAlignHorzEnum.Right
                    Exit Select
            End Select
        End If
        If s.VerticalAlignment.HasValue Then
            Select Case s.VerticalAlignment.Value
                Case VerticalAlignment.Top
                    x.AlignVert = XLAlignVertEnum.Top
```

```
                    Exit Select
                Case VerticalAlignment.Center
                    x.AlignVert = XLAlignVertEnum.Center
                    Exit Select
                Case VerticalAlignment.Bottom
                    x.AlignVert = XLAlignVertEnum.Bottom
                    Exit Select
            End Select
        End If
        If s.TextWrapping.HasValue Then
            x.WordWrap = s.TextWrapping.Value
        End If

        ' カラー
        If TypeOf s.Background Is SolidColorBrush Then
            x.BackColor = DirectCast(s.Background, SolidColorBrush).Color
            x.BackPattern = XLPatternEnum.Solid
        End If
        If TypeOf s.Foreground Is SolidColorBrush Then
            x.ForeColor = DirectCast(s.Foreground, SolidColorBrush).Color
        End If

        ' フォント
        Dim fontName As String = flex.FontFamily.Source
        Dim fontSize As Double = flex.FontSize
        Dim bold As Boolean = False
        Dim italic As Boolean = False
        Dim underline As Boolean = False
        Dim hasFont As Boolean = False
        If s.FontFamily IsNot Nothing Then
            fontName = s.FontFamily.Source
            hasFont = True
        End If
        If s.FontSize.HasValue Then
            fontSize = s.FontSize.Value
            hasFont = True
        End If
        If s.FontWeight.HasValue Then
            bold = s.FontWeight.Value = FontWeights.Bold OrElse s.FontWeight.Value
= FontWeights.ExtraBold OrElse s.FontWeight.Value = FontWeights.SemiBold
            hasFont = True
        End If
        If s.FontStyle.HasValue Then
            italic = s.FontStyle.Value = FontStyles.Italic
            hasFont = True
        End If
        If s.TextDecorations IsNot Nothing Then
            underline = True
            hasFont = True
        End If
        If hasFont Then
            fontSize = PixelsToPoints(fontSize)
            If underline Then
```

```
                Dim color As Color = Colors.Black
                If TypeOf flex.Foreground Is SolidColorBrush Then
                    color = DirectCast(flex.Foreground, SolidColorBrush).Color
                End If
                If TypeOf s.Foreground Is SolidColorBrush Then
                    color = DirectCast(s.Foreground, SolidColorBrush).Color
                End If
                x.Font = New XLFont(fontName, Convert.ToSingle(fontSize), bold,
italic, False, XLFontScript.None, _
                    XLUnderlineStyle.[Single], color)
            Else
                x.Font = New XLFont(fontName, Convert.ToSingle(fontSize), bold,
italic)
            End If
        End If

        ' 書式
        If Not String.IsNullOrEmpty(s.Format) Then
            x.Format = XLStyle.FormatDotNetToXL(s.Format)
        End If

        ' 境界線
        If s.CellBorderThickness.Left > 0 AndAlso TypeOf s.CellBorderBrushLeft Is
SolidColorBrush Then
            x.BorderLeft = GetBorderLineStyle(s.CellBorderThickness.Left)
            x.BorderColorLeft = DirectCast(s.CellBorderBrushLeft,
SolidColorBrush).Color
        End If
        If s.CellBorderThickness.Top > 0 AndAlso TypeOf s.CellBorderBrushTop Is
SolidColorBrush Then
            x.BorderTop = GetBorderLineStyle(s.CellBorderThickness.Top)
            x.BorderColorTop = DirectCast(s.CellBorderBrushTop,
SolidColorBrush).Color
        End If
        If s.CellBorderThickness.Right > 0 AndAlso TypeOf s.CellBorderBrushRight Is
SolidColorBrush Then
            x.BorderRight = GetBorderLineStyle(s.CellBorderThickness.Right)
            x.BorderColorRight = DirectCast(s.CellBorderBrushRight,
SolidColorBrush).Color
        End If
        If s.CellBorderThickness.Bottom > 0 AndAlso TypeOf s.CellBorderBrushBottom
Is SolidColorBrush Then
            x.BorderBottom = GetBorderLineStyle(s.CellBorderThickness.Bottom)
            x.BorderColorBottom = DirectCast(s.CellBorderBrushBottom,
SolidColorBrush).Color
        End If

        ' キャッシュに保存して戻します
        _excelStyles(s) = x
        Return x
    End Function
    Private Shared Function GetBorderThickness(ls As XLLineStyleEnum) As Double
```

```
        Select Case ls
            Case XLLineStyleEnum.None
                Return 0
            Case XLLineStyleEnum.Hair
                Return 0.5
            Case XLLineStyleEnum.Thin, XLLineStyleEnum.ThinDashDotDotted,
XLLineStyleEnum.ThinDashDotted, XLLineStyleEnum.Dashed, XLLineStyleEnum.Dotted
                Return 1
            Case XLLineStyleEnum.Medium, XLLineStyleEnum.MediumDashDotDotted,
XLLineStyleEnum.MediumDashDotted, XLLineStyleEnum.MediumDashed,
XLLineStyleEnum.SlantedMediumDashDotted
                Return 2
            Case XLLineStyleEnum.[Double], XLLineStyleEnum.Thick
                Return 3
        End Select
        Return 0
    End Function
    Private Shared Function GetBorderLineStyle(t As Double) As XLLineStyleEnum
        If t = 0 Then
            Return XLLineStyleEnum.None
        End If
        If t < 1 Then
            Return XLLineStyleEnum.Hair
        End If
        If t < 2 Then
            Return XLLineStyleEnum.Thin
        End If
        If t < 3 Then
            Return XLLineStyleEnum.Medium
        End If
        Return XLLineStyleEnum.Thick
    End Function
    Private Shared Function GetBorderBrush(color As Color) As Brush
        Return If(IsColorValid(color), New SolidColorBrush(color), Nothing)
    End Function
    Private Shared Function IsColorValid(color As Color) As Boolean
        Return color.A > 0
        ' == 0xff;
    End Function

#End Region
End Class
```

ExcelCellStyle.vb

```
Imports C1.WPF.FlexGrid
Imports System.ComponentModel

Public Class ExcelCellStyle
    Inherits CellStyle
    ' ** フィールド
    Private _format As String
    Private _bdrThickness As Thickness
    Private _bdrLeft As Brush
    Private _bdrTop As Brush
    Private _bdrRight As Brush
    Private _bdrBottom As Brush

    Private Shared _thicknessEmpty As New Thickness(0)
    ' ** オブジェクト・モデル
    Public Property Format() As String
        Get
            Return _format
        End Get
        Set(value As String)
            If value <> _format Then
                _format = value
                OnPropertyChanged(New PropertyChangedEventArgs("Format"))
            End If
        End Set
    End Property
    Public Property CellBorderThickness() As Thickness
        Get
            Return _bdrThickness
        End Get
        Set(value As Thickness)
            If value <> _bdrThickness Then
                _bdrThickness = value
                OnPropertyChanged(New PropertyChangedEventArgs("BorderThickness"))
            End If
        End Set
    End Property
    Public Property CellBorderBrushLeft() As Brush
        Get
            Return _bdrLeft
        End Get
        Set(value As Brush)
            If Not Object.ReferenceEquals(value, _bdrLeft) Then
                _bdrLeft = value
                OnPropertyChanged(New PropertyChangedEventArgs("BorderColorLeft"))
            End If
        End Set
    End Property
    Public Property CellBorderBrushTop() As Brush
```

```
        Get
            Return _bdrTop
        End Get
        Set(value As Brush)
            If Not Object.ReferenceEquals(value, _bdrTop) Then
                _bdrTop = value
                OnPropertyChanged(New PropertyChangedEventArgs("BorderColorTop"))
            End If
        End Set
    End Property
    Public Property CellBorderBrushRight() As Brush
        Get
            Return _bdrRight
        End Get
        Set(value As Brush)
            If Not Object.ReferenceEquals(value, _bdrRight) Then
                _bdrRight = value
                OnPropertyChanged(New PropertyChangedEventArgs("BorderColorRight"))
            End If
        End Set
    End Property
    Public Property CellBorderBrushBottom() As Brush
        Get
            Return _bdrBottom
        End Get
        Set(value As Brush)
            If Not Object.ReferenceEquals(value, _bdrBottom) Then
                _bdrBottom = value
                OnPropertyChanged(New
PropertyChangedEventArgs("BorderColorBottom"))
            End If
        End Set
    End Property

    ' ** オーバーライド
    Public Overrides Sub Apply(bdr As Border, selState As SelectedState)
        MyBase.Apply(bdr, selState)
        ApplyBorder(bdr, _bdrLeft, New Thickness(_bdrThickness.Left, 0, 0, 0))
        ApplyBorder(bdr, _bdrTop, New Thickness(0, _bdrThickness.Top, 0, 0))
        ApplyBorder(bdr, _bdrRight, New Thickness(0, 0, _bdrThickness.Right, 0))
        ApplyBorder(bdr, _bdrBottom, New Thickness(0, 0, 0, _bdrThickness.Bottom))
    End Sub
    Private Sub ApplyBorder(bdr As Border, br As Brush, t As Thickness)
        If br IsNot Nothing AndAlso t <> _thicknessEmpty Then
            ' 内部の境界線を作成します
            Dim inner As Border = New Border()
            inner.BorderThickness = t
            inner.BorderBrush = br

            ' コンテンツに拡張します
            Dim content As Border = bdr.Child
            bdr.Child = inner
            inner.Child = content
```

```
            ' パディングします
            inner.Padding = bdr.Padding
            bdr.Padding = _thicknessEmpty
        End If
    End Sub
End Class
```

ExcelRow.vb

```
Imports C1.WPF.FlexGrid
Imports System.Globalization

Public Class ExcelRow
    Inherits GroupRow
    ' ** フィールド

    Private _cellStyles As Dictionary(Of Column, CellStyle)
    ' 既定の有効桁数を6桁に設定します
    Private Const DEFAULT_FORMAT As String = "#,##0.######"

    ' ** ctor
    Public Sub New(styleRow As ExcelRow)
        IsReadOnly = False
        If styleRow IsNot Nothing AndAlso styleRow.Grid IsNot Nothing Then
            For Each c As Column In styleRow.Grid.Columns
                Dim cs As CellStyle = styleRow.GetCellStyle(c)
                If cs IsNot Nothing Then
                    Me.SetCellStyle(c, cs.Clone())
                End If
            Next
        End If
    End Sub
    Public Sub New()
        Me.New(Nothing)
    End Sub

    ' ** オブジェクト・モデル

    ''' <summary>
    ''' データを取得する場合、書式を適用するためにオーバーライドされる
    ''' </summary>
    Public Overrides Function GetDataFormatted(col As Column) As String
        ' データを取得します
        Dim data As Object = GetDataRaw(col)

        ' 書式を適用します
        Dim ifmt As IFormattable = TryCast(data, IFormattable)
        If ifmt IsNot Nothing Then
            ' セルの書式を取得します
            Dim s As ExcelCellStyle = TryCast(GetCellStyle(col), ExcelCellStyle)
            Dim fmt As String = If(s IsNot Nothing AndAlso (Not
String.IsNullOrEmpty(s.Format)), s.Format, DEFAULT_FORMAT)
            data = ifmt.ToString(fmt, CultureInfo.CurrentUICulture)
        End If

        ' 完了
        Return If(data IsNot Nothing, data.ToString(), String.Empty)
    End Function
```

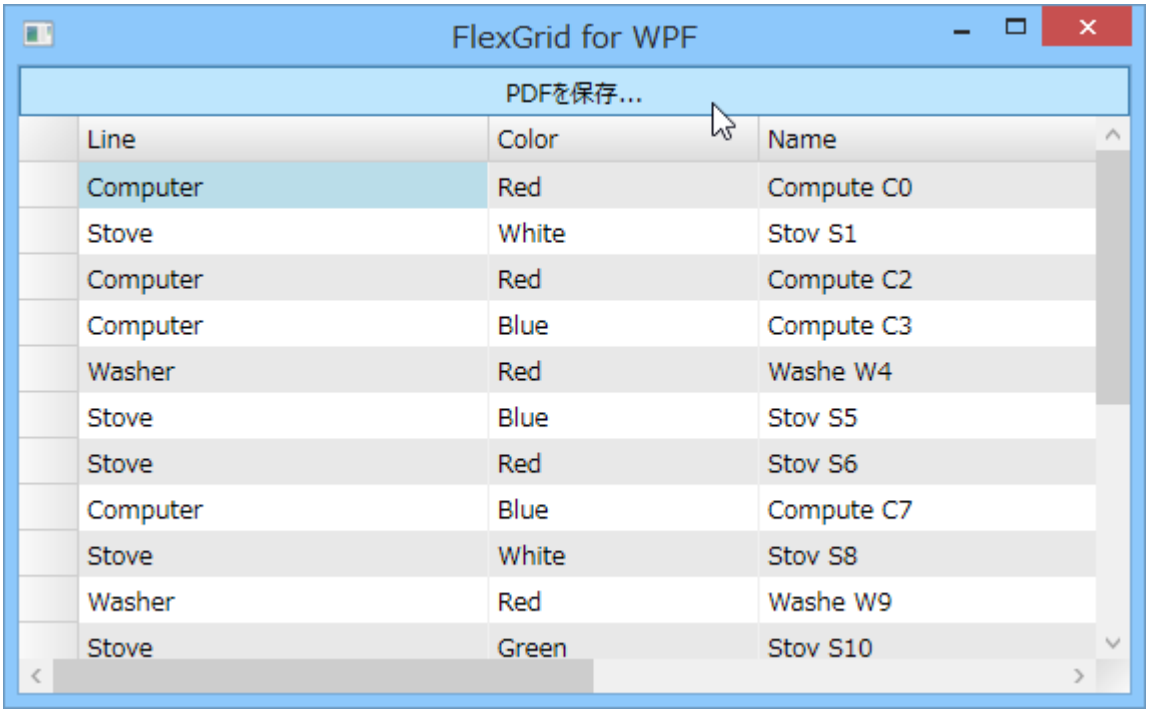
```
    ' ** オブジェクト・モデル

    ''' <summary>
    ''' この行では、セルにスタイルを適用します
    ''' </summary>
    Public Sub SetCellStyle(col As Column, style As CellStyle)
        If Not Object.ReferenceEquals(style, GetCellStyle(col)) Then
            If _cellStyles Is Nothing Then
                _cellStyles = New Dictionary(Of Column, CellStyle)()
            End If
            _cellStyles(col) = style
            If Grid IsNot Nothing Then
                Grid.Invalidate(New CellRange(Me.Index, col.Index))
            End If
        End If
    End Sub
    ''' <summary>
    ''' この行では、セルに適用したスタイルを取得します
    ''' </summary>
    Public Function GetCellStyle(col As Column) As CellStyle
        Dim s As CellStyle = Nothing
        If _cellStyles IsNot Nothing Then
            _cellStyles.TryGetValue(col, s)
        End If
        Return s
    End Function

End Class
```

PDFファイルを保存する

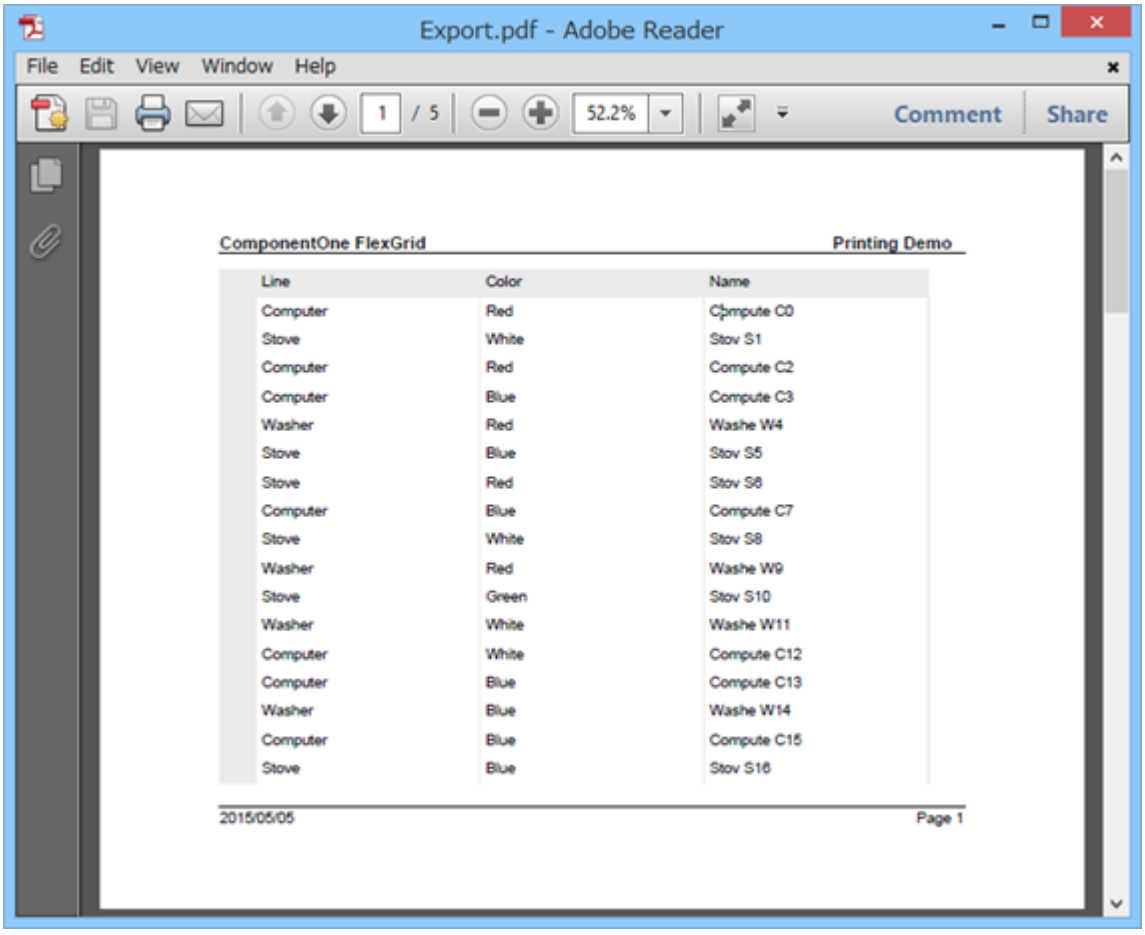
C1FlexGrid のデータをPDFファイルにエクスポートする方法として、Helperクラス、たとえばGridExport、を使用してグリッドの内容をPDF内に描画する方法があります。具体的には、GridExportクラス内に SavePdf というメソッドで FlexGrid を PDFストリーム内に直接描画します。この方法は次のように簡単に実現できます。

また、ここはヘルパークラス内に RenderGrid というメソッドでグリッドを既存の C1PdfDocument 上に描画する方式も含みます。RenderGrid は、複数グリッドを文書に追加してグラフ、テキスト、画像と他のUI要素も組み合わせる場合よく使いこなします。詳細方法については、次に示すコードをご確認ください。以下に示す GridExport のヘルパークラスは必要に応じてカスタマイズできますので活用してみてください。

[実行例]



PDF ファイルを保存します



サンプルコードは次のようになります。

C# サンプルコード

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();

        // グリッドにデータを挿入します
        _flex1.ItemsSource = Product.GetProducts(20);
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
        var dlg = new Microsoft.Win32.SaveFileDialog();
        dlg.Filter = "PDF files (*.pdf)|*.pdf";
        //var t = dlg.ShowDialog();
        if (dlg.ShowDialog().Value)
        {
            // PDF文書を作成します
            var pdf = new C1PdfDocument();
            pdf.Landscape = true;
            pdf.Compression = CompressionLevel.NoCompression;

            // グリッドをPDFに描画します
            var options = new PdfExportOptions();
            options.ScaleMode = ScaleMode.ActualSize;
            GridExport.RenderGrid(pdf, _flex1, options);
            pdf.NewPage();

            // 文書を保存します
            using (var stream = dlg.OpenFile())
            {
                pdf.Save(stream);
            }
        }
    }
}
```

GridExport.cs

```
namespace MultiGridPdf
{
    internal static class GridExport
    {
        public static void SavePdf(C1FlexGrid flex, Stream s)
        {
            var options = new PdfExportOptions();
            SavePdf(flex, s, options);
        }
        public static void SavePdf(C1FlexGrid flex, Stream s, PdfExportOptions
options)
        {
            var pdf = new C1PdfDocument();
            options.KnownPageCount = false;
            RenderGrid(pdf, flex, options);

            // PDF文書を保存してストリームを閉じます
            pdf.Save(s);
            s.Close();
        }

        public static void RenderGrid(C1PdfDocument pdf, C1FlexGrid flex)
        {
            RenderGrid(pdf, flex, null);
        }
        public static void RenderGrid(C1PdfDocument pdf, C1FlexGrid flex,
PdfExportOptions options)
        {
            // 描画オプションを取得します
            if (options == null)
            {
                options = new PdfExportOptions();
            }

            // PDFページをレイアウトするためにルート要素を取得します
            Panel root = null;
            for (var parent = flex.Parent as FrameworkElement; parent != null;
parent = parent.Parent as FrameworkElement)
            {
                if (parent is Panel)
                {
                    root = parent as Panel;
                }
            }

            // ページサイズを取得します
            var rc = pdf.PageRectangle;

            // 描画する間に要素を保持するためにパネルを作成します
```

```
            var pageTemplate = new PageTemplate();
            pageTemplate.Width = rc.Width;
            pageTemplate.Height = rc.Height;
            pageTemplate.SetPageMargin(options.Margin);
            root.Children.Add(pageTemplate);

            // PDF文書にグリッドを描画します
            var m = options.Margin;
            var sz = new Size(rc.Width - m.Left - m.Right, rc.Height - m.Top -
m.Bottom);
            var pages = flex.GetPageImages(options.ScaleMode, sz, 100);
            for (int i = 0; i < pages.Count; i++)
            {
                // 必要な場合、ページをスキップします
                if (i > 0)
                {
                    pdf.NewPage();
                }

                // コンテンツを設定します
                pageTemplate.PageContent.Child = pages[i];
                pageTemplate.PageContent.Stretch = options.ScaleMode ==
ScaleMode.ActualSize
                    ? System.Windows.Media.Stretch.None
                    : System.Windows.Media.Stretch.Uniform;

                // ヘッダー・フッターのテキストを設定します
                pageTemplate.HeaderLeft.Text = options.DocumentTitle;
                if (options.KnownPageCount)
                {
                    pageTemplate.FooterRight.Text = string.Format("Page {0} of
{1}",
                        pdf.CurrentPage + 1, pages.Count);
                }
                else
                {
                    pageTemplate.FooterRight.Text = string.Format("Page {0}",
                        pdf.CurrentPage + 1);
                }

                // ページの要素を測定します
                pageTemplate.UpdateLayout();
                pageTemplate.Arrange(new Rect(0, 0, rc.Width, rc.Height));

                // PDFに追加します
                pdf.DrawElement(pageTemplate, rc);
            }

            // テンプレート完了
            root.Children.Remove(pageTemplate);
        }
```

```
    }
}
```

PdfExportOptions.cs

```
namespace MultiGridPdf
{
    public class PdfExportOptions
    {
        public PdfExportOptions()
        {
            Margin = new Thickness(96);
            ScaleMode = ScaleMode.PageWidth;
            DocumentTitle = "ComponentOne FlexGrid";
        }
        public Thickness Margin { get; set; }
        public ScaleMode ScaleMode { get; set; }
        public string DocumentTitle { get; set; }
        public bool KnownPageCount { get; set; }
    }
}
```

PageTemplate.xaml

```
<!-- ヘッダー -->
<Border Grid.Column="1" HorizontalAlignment="Stretch" VerticalAlignment="Bottom"
        Margin="0 12"
        BorderBrush="Black"
        BorderThickness="0 0 0 1" >
    <Grid>
        <TextBlock x:Name="HeaderLeft" Text="ComponentOne FlexGrid"
FontWeight="Bold" FontSize="14"
            VerticalAlignment="Bottom" HorizontalAlignment="Left" />
        <TextBlock x:Name="HeaderRight" Text="Printing Demo" FontWeight="Bold"
FontSize="14" HorizontalAlignment="Right" Height="18" VerticalAlignment="Bottom" />
    </Grid>
</Border>

<!-- フッター -->
<Border Grid.Column="1" Grid.Row="2" HorizontalAlignment="Stretch"
VerticalAlignment="Top"
        Margin="0 12"
        BorderBrush="Black"
        BorderThickness="0 1 0 0" >
    <Grid>
        <TextBlock x:Name="FooterLeft" Text="Today"
            VerticalAlignment="Bottom" HorizontalAlignment="Left" />
        <TextBlock x:Name="FooterRight" Text="Page {0} of {1}"
            VerticalAlignment="Bottom" HorizontalAlignment="Right" />
    </Grid>
</Border>

<!-- コンテンツ -->
<Viewbox Name="PageContent" Grid.Row="1" Grid.Column="1"
    VerticalAlignment="Top" HorizontalAlignment="Left" />
```

VB サンプルコード

```
Class MainWindow
    Inherits Window
    Public Sub New()
        InitializeComponent()
        ' グリッドにデータを挿入します
        _flex1.ItemsSource = Product.GetProducts(10)
        _flex1.CollectionView.Filter = AddressOf FilterComputers

    End Sub

    Private Function FilterComputers(item As Object) As Boolean
        Dim p = TryCast(item, Product)
        Return p.Line = "Computers"
    End Function
    Private Function FilterStoves(item As Object) As Boolean
        Dim p = TryCast(item, Product)
        Return p.Line = "Stoves"
    End Function
    Private Function FilterWashers(item As Object) As Boolean
        Dim p = TryCast(item, Product)
        Return p.Line = "Washers"
    End Function
    Private Sub Button_Click(sender As Object, e As RoutedEventArgs)
        Dim dlg = New Microsoft.Win32.SaveFileDialog()
        dlg.Filter = "PDF files (*.pdf)|*.pdf"
        'var t = dlg.ShowDialog();
        If dlg.ShowDialog().Value Then
            ' PDF文書を作成します
            Dim pdf = New C1PdfDocument()
            pdf.Landscape = True
            pdf.Compression = CompressionLevel.NoCompression

            ' グリッドをPDFに描画します
            Dim options = New PdfExportOptions()
            options.ScaleMode = ScaleMode.ActualSize
            GridExport.RenderGrid(pdf, _flex1, options)

            ' 文書を保存します
            Using stream = dlg.OpenFile()
                pdf.Save(stream)
            End Using
        End If
    End Sub

End Class
```

GridExport.vb

```
Public Class GridExport
    Public Shared Sub SavePdf(flex As C1FlexGrid, s As Stream)
        Dim options = New PdfExportOptions()
        SavePdf(flex, s, options)
    End Sub
    Public Shared Sub SavePdf(flex As C1FlexGrid, s As Stream, options As
PdfExportOptions)
        Dim pdf = New C1PdfDocument()
        options.KnownPageCount = False
        RenderGrid(pdf, flex, options)

        ' PDF文書を保存してストリームを閉じます
        pdf.Save(s)
        s.Close()
    End Sub

    Public Shared Sub RenderGrid(pdf As C1PdfDocument, flex As C1FlexGrid)
        RenderGrid(pdf, flex, Nothing)
    End Sub
    Public Shared Sub RenderGrid(pdf As C1PdfDocument, flex As C1FlexGrid, options
As PdfExportOptions)
        ' 描画オプションを取得します
        If options Is Nothing Then
            options = New PdfExportOptions()
        End If

        ' PDFページをレイアウトするためにルート要素を取得します
        Dim root As Panel = Nothing
        Dim parent = TryCast(flex.Parent, FrameworkElement)
        While parent IsNot Nothing
            If TypeOf parent Is Panel Then
                root = TryCast(parent, Panel)
            End If
            parent = TryCast(parent.Parent, FrameworkElement)
        End While

        ' ページサイズを取得します
        Dim rc = pdf.PageRectangle

        ' 描画する間に要素を保持するためにパネルを作成します
        Dim pageTemplate = New PageTemplate()
        pageTemplate.Width = rc.Width
        pageTemplate.Height = rc.Height
        pageTemplate.SetPageMargin(options.Margin)
        root.Children.Add(pageTemplate)

        ' PDF文書にグリッドを描画します
        Dim m = options.Margin
        Dim sz = New Size(rc.Width - m.Left - m.Right, rc.Height - m.Top -
```

```
m.Bottom)
        Dim pages = flex.GetPageImages(options.ScaleMode, sz, 100)
        For i As Integer = 0 To pages.Count - 1
            ' 必要な場合、ページをスキップします
            If i > 0 Then
                pdf.NewPage()
            End If

            ' コンテンツを設定します
            pageTemplate.PageContent.Child = pages(i)
            pageTemplate.PageContent.Stretch = If(options.ScaleMode =
ScaleMode.ActualSize, System.Windows.Media.Stretch.None,
System.Windows.Media.Stretch.Uniform)

            ' ヘッダー・フッターのテキストを設定します
            pageTemplate.HeaderLeft.Text = options.DocumentTitle
            If options.KnownPageCount Then
                pageTemplate.FooterRight.Text = String.Format("Page {0} of {1}",
pdf.CurrentPage + 1, pages.Count)
            Else
                pageTemplate.FooterRight.Text = String.Format("Page {0}",
pdf.CurrentPage + 1)
            End If

            ' ページの要素を測定します
            pageTemplate.UpdateLayout()
            pageTemplate.Arrange(New Rect(0, 0, rc.Width, rc.Height))

            ' PDFに追加します
            pdf.DrawElement(pageTemplate, rc)
        Next

        ' テンプレート完了
        root.Children.Remove(pageTemplate)
    End Sub
End Class
```

PdfExportOptions.vb

```
Public Class PdfExportOptions
    Public Sub New()
        Margin = New Thickness(96)
        ScaleMode = ScaleMode.PageWidth
        DocumentTitle = "ComponentOne FlexGrid"
    End Sub
    Public Property Margin() As Thickness
        Get
            Return m_Margin
        End Get
        Set(value As Thickness)
            m_Margin = value
        End Set
    End Property
    Private m_Margin As Thickness
    Public Property ScaleMode() As ScaleMode
        Get
            Return m_ScaleMode
        End Get
        Set(value As ScaleMode)
            m_ScaleMode = value
        End Set
    End Property
    Private m_ScaleMode As ScaleMode
    Public Property DocumentTitle() As String
        Get
            Return m_DocumentTitle
        End Get
        Set(value As String)
            m_DocumentTitle = value
        End Set
    End Property
    Private m_DocumentTitle As String
    Public Property KnownPageCount() As Boolean
        Get
            Return m_KnownPageCount
        End Get
        Set(value As Boolean)
            m_KnownPageCount = value
        End Set
    End Property
    Private m_KnownPageCount As Boolean
End Class
```

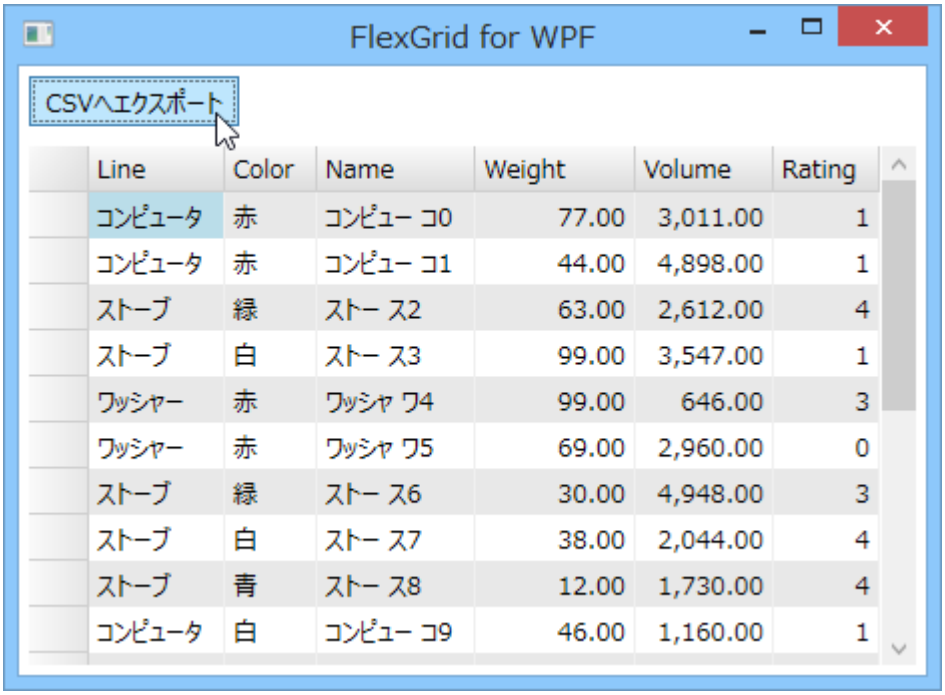
CSVファイルを保存する

グリッドの内容をCSV形式ファイルにエクスポートするには、C1FlexGrid の **Save** メソッドを使用します。

次の例では、グリッドはデータソースと連結されていることを前提として、ボタンクリックでグリッドの内容をCSV形式ファイルに

エクスポートする方法を示します。

[実行例]



コードは次のようになります。

VisualBasic

```
' ボタンクリックでCSVファイルを保存します
Private Sub exportBtn_Click(sender As Object, e As RoutedEventArgs)
    Dim dlg = New Microsoft.Win32.SaveFileDialog()
    dlg.Filter = "CSV files (*.csv)|*.csv"
    If dlg.ShowDialog().Value Then
        _flex.Save(dlg.FileName, C1.WPF.FlexGrid.FileFormat.Csv)
    End If

End Sub
```

C#

```
//ボタンクリックでCSVファイルを保存します
private void exportBtn_Click(object sender, RoutedEventArgs e)
{
    var dlg = new Microsoft.Win32.SaveFileDialog();
    dlg.Filter = "CSV files (*.csv)|*.csv";
    if (dlg.ShowDialog().Value)
    {
        _flex.Save(dlg.FileName, C1.WPF.FlexGrid.FileFormat.Csv);
    }

}
```

固定およびピン留め

Freezing Rows and Columns

This feature in **WPF FlexGrid** allows you to freeze the rows and columns using mouse drag during runtime following a particular sequence. You can set the **AllowFreezing** Enum to **Columns** to freeze only columns, **Rows** to freeze only rows, or **Both** to freeze both columns and rows. Conversely, to disable freezing, set the AllowFreezing Enum to None, which is the default setting. This Enum can be set either in the designer or in code.

Implementation

In Designer

Locate the **AllowFreezing** Enum in the Properties window and set it to Rows, Columns, Both or None.

In Code

Add the following code to set the **AllowFreezing** Enum to **Both**:

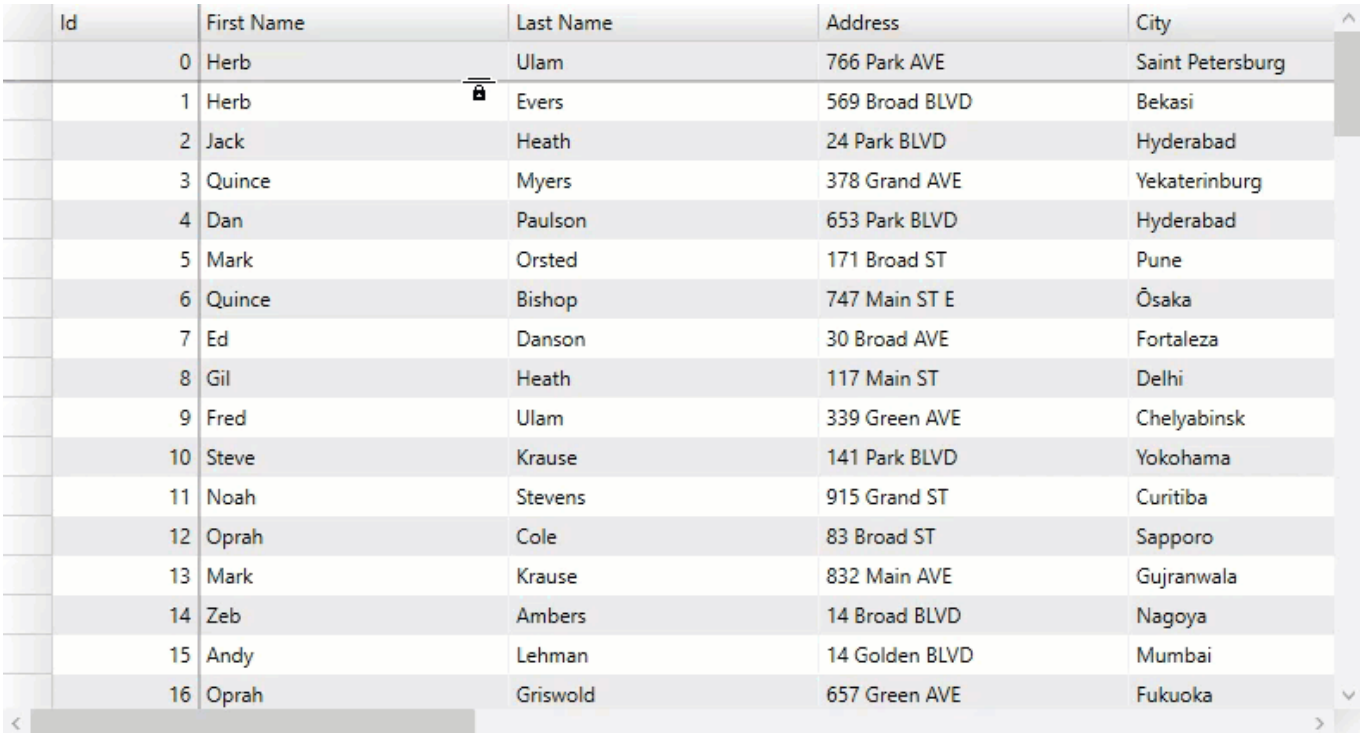
CS

```
c1.FlexGrid.AllowFreezing = C1.WPF.FlexGrid.AllowFreezing.Both
```

Now, you can manually use the mouse drag to adjust the number of frozen columns and rows as needed.

Feature Illustration

When the mouse pointer becomes the lock rows or the lock columns icon, click and drag the mouse over the rows or columns to freeze in a sequence. Setting the AllowFreezing Enum to **Both** allows both rows and columns to be frozen at the same time, as seen in the following GIF.



Id	First Name	Last Name	Address	City
0	Herb	Ulam	766 Park AVE	Saint Petersburg
1	Herb	Evers	569 Broad BLVD	Bekasi
2	Jack	Heath	24 Park BLVD	Hyderabad
3	Quince	Myers	378 Grand AVE	Yekaterinburg
4	Dan	Paulson	653 Park BLVD	Hyderabad
5	Mark	Orsted	171 Broad ST	Pune
6	Quince	Bishop	747 Main ST E	Ōsaka
7	Ed	Danson	30 Broad AVE	Fortaleza
8	Gil	Heath	117 Main ST	Delhi
9	Fred	Ulam	339 Green AVE	Chelyabinsk
10	Steve	Krause	141 Park BLVD	Yokohama
11	Noah	Stevens	915 Grand ST	Curitiba
12	Oprah	Cole	83 Broad ST	Sapporo
13	Mark	Krause	832 Main AVE	Gujranwala
14	Zeb	Ambers	14 Broad BLVD	Nagoya
15	Andy	Lehman	14 Golden BLVD	Mumbai
16	Oprah	Griswold	657 Green AVE	Fukuoka

Column Pinning

Column Pinning in **WPF FlexGrid** allows the you to freeze any column in the grid and pin it to the start, this will allow you to see it while horizontally scrolling the Grid. A column or multiple columns can be pinned to the left-hand side of the FlexGrid.

Each column can be pinned, as long as the pinned area does not become wider than the Grid itself. Pinned columns are always rendered on the left side of the Grid and stay fixed through horizontal scrolling of the unpinned columns in the Grid body.

Implementation

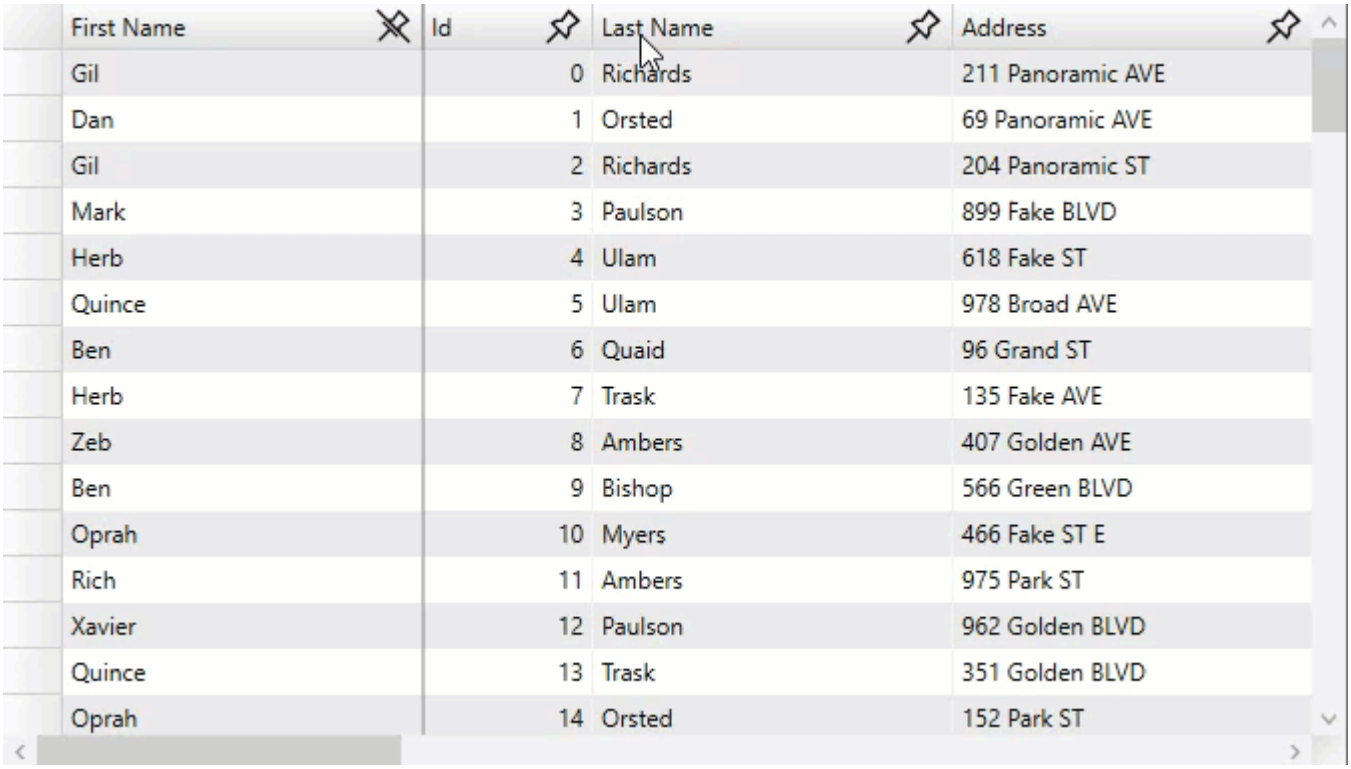
Use the following code to implement Column Pinning in your FlexGrid application through **ColumnPinningCellFactory** class which is a customized **CellFactory** class that will add a pinning icon to each column header. For more information about CellFactory, refer to the CellFactory class topic.

CS

```
private void GridColumnPinning_Loaded(object sender, RoutedEventArgs e)
        {
            grid.CellFactory = new ColumnPinningCellFactory();

        }
```

Feature Illustration

The following GIF demonstrates column pinning feature in the FlexGrid control. Any coloumn from the grid can be pinned where pinned columns make a particular sequence.



First Name	Id	Last Name	Address
Gil	0	Richards	211 Panoramic AVE
Dan	1	Orsted	69 Panoramic AVE
Gil	2	Richards	204 Panoramic ST
Mark	3	Paulson	899 Fake BLVD
Herb	4	Ulam	618 Fake ST
Quince	5	Ulam	978 Broad AVE
Ben	6	Quaid	96 Grand ST
Herb	7	Trask	135 Fake AVE
Zeb	8	Ambers	407 Golden AVE
Ben	9	Bishop	566 Green BLVD
Oprah	10	Myers	466 Fake ST E
Rich	11	Ambers	975 Park ST
Xavier	12	Paulson	962 Golden BLVD
Quince	13	Trask	351 Golden BLVD
Oprah	14	Orsted	152 Park ST

Note: ColumnPinning is not a part of the control, it is only available as a code sample. You can find the complete source code of *ColumnPinningCellFactory* in the ColumnPinning sample available at following location:

Documents\ComponentOne Samples\WPF\C1.WPF.FlexGrid\CS\ColumnPinning

Excelへのエクスポート

FlexGrid allows you to export files to CSV, PDF, Xlsx/Xlsm, and HTML file formats with the help of the **C1.WPF.Grid.Excel** library. This library provides **SaveAsync** method in the **Extensions** class which can be used to export FlexGrid and save it to either the file system or a Stream. The SaveAsync method has two overloads depending on whether you need to save to a stream or the file system:

Overloads	Description
SaveAsync(C1.WPF.Grid.FlexGrid,System.String,System.String, GrapeCity.Documents.Excel.SaveFileFormat, C1.WPF.Grid.GridRowColRanges,C1.WPF.Grid.GridRowColRanges, C1.WPF.Grid.GridHeadersVisibility,System.Boolean,System.Boolean, System.Boolean,System.Drawing.Printing.PrinterSettings)	Saves the contents of the grid in a desired file format using GcExcel.
SaveAsync(C1.WPF.Grid.FlexGrid,System.IO.Stream,System.String, GrapeCity.Documents.Excel.SaveFileFormat, C1.WPF.Grid.GridRowColRanges,C1.WPF.Grid.GridRowColRanges, C1.WPF.Grid.GridHeadersVisibility,System.Boolean,System.Boolean, System.Boolean,System.Drawing.Printing.PrinterSettings)	Saves the contents of the grid in a stream in desired format using GcExcel.

To export FlexGrid to any other format, you need to add the following dependencies to your application:

- using **C1.WPF.Grid**;
- using **GrapeCity.Documents.Excel**;

The following example shows how to use the SaveAsync method to export FlexGrid to Excel files with XLSX/XSLM format. In this example, we used the **Name** property to name the FlexGrid control as "flex".

C#

```
private void Save_Click(object sender, RoutedEventArgs e)
{
    flex.SaveAsync(@"..\..\..\..\FlexGridSheet_overload2.xlsx","flex
Sheet",SaveFileFormat.Xlsx, new GridRowColRanges("1-3",
GridRowColRangesOptions.VisibleOnly), new GridRowColRanges("1-3",
GridRowColRangesOptions.VisibleOnly), GridHeadersVisibility.All, false,false, false,
null);
}
```

カスタムセル

Microsoft のデータグリッドコントロール（Windows フォーム、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、コードを使ってカスタム **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。この方法は十分に実用的です（ComponentOne に含まれる DataGrid for WPF とSilverlight も、主に Microsoft のグリッドとの互換性を保つために、このモデルに準拠しています）。

コードでのカスタムセル:CellFactory クラス

次のセクションでは、FlexGrid .NET 4.5.2および.NET 5バージョンでCellFactoryを使用してカスタムセルを実装する方法を学習します。

.NET Framework

C1FlexGrid 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 インターフェイ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 クラスは直接セルを操作するため、カスタム列より簡潔です。カスタム ICellFactory クラスは、C1FlexGrid クラスに含まれるデフォルトの **CellFactory** クラスを継承できるため、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。

次の図に、FlexGrid で CellFactory を使用して作成されたカスタムセルを示します。

	Symbol	Name	Bid	Ask	Last Sale
	A	Agilent Technologies	765.95 (2.9%)	808.60 (4.5%)	787.27 (3.7%)
	AA	Alcoa Inc.	940.02 (-1.4%)	856.95 (3.2%)	898.49 (0.7%)
	AACC	Asset Acceptance Capital Corp.	712.93 (-3.0%)	678.27 (1.7%)	695.60 (-0.8%)
	AAME	Atlantic American Corporation	842.38 (4.9%)	980.37 (-2.6%)	911.38 (0.7%)
	AANB	Abigail Adams National Bancorp, Inc.	784.71 (0.3%)	749.68 (-0.9%)	767.19 (-0.3%)
	AAON	AAON, Inc.	187.56 (6.0%)	198.61 (5.1%)	193.08 (5.5%)
	AAPL	Apple Inc.	31.14 (-2.0%)	30.88 (-1.6%)	31.01 (-1.8%)
	AATI	Advanced Analogic Technologies, Inc.	136.74 (2.5%)	144.40 (2.9%)	140.57 (2.7%)
	AAUK	Anglo American plc	168.06 (0.8%)	138.89 (-3.7%)	153.47 (-1.3%)
	AAWW	Atlas Air Worldwide Holdings	259.02 (5.6%)	197.86 (3.6%)	228.44 (4.7%)
	ABAX	ABAXIS, Inc.	758.44 (5.6%)	687.37 (-2.1%)	722.91 (1.8%)
	ABBC	Abington Bancorp, Inc.	215.18 (-0.4%)	180.83 (-3.7%)	198.01 (-1.9%)

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、WPF 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、パフォーマンスに影響を与えます。

次のコードは、**CellFactory** インタフェースを示しています。

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);
```

```
}
```

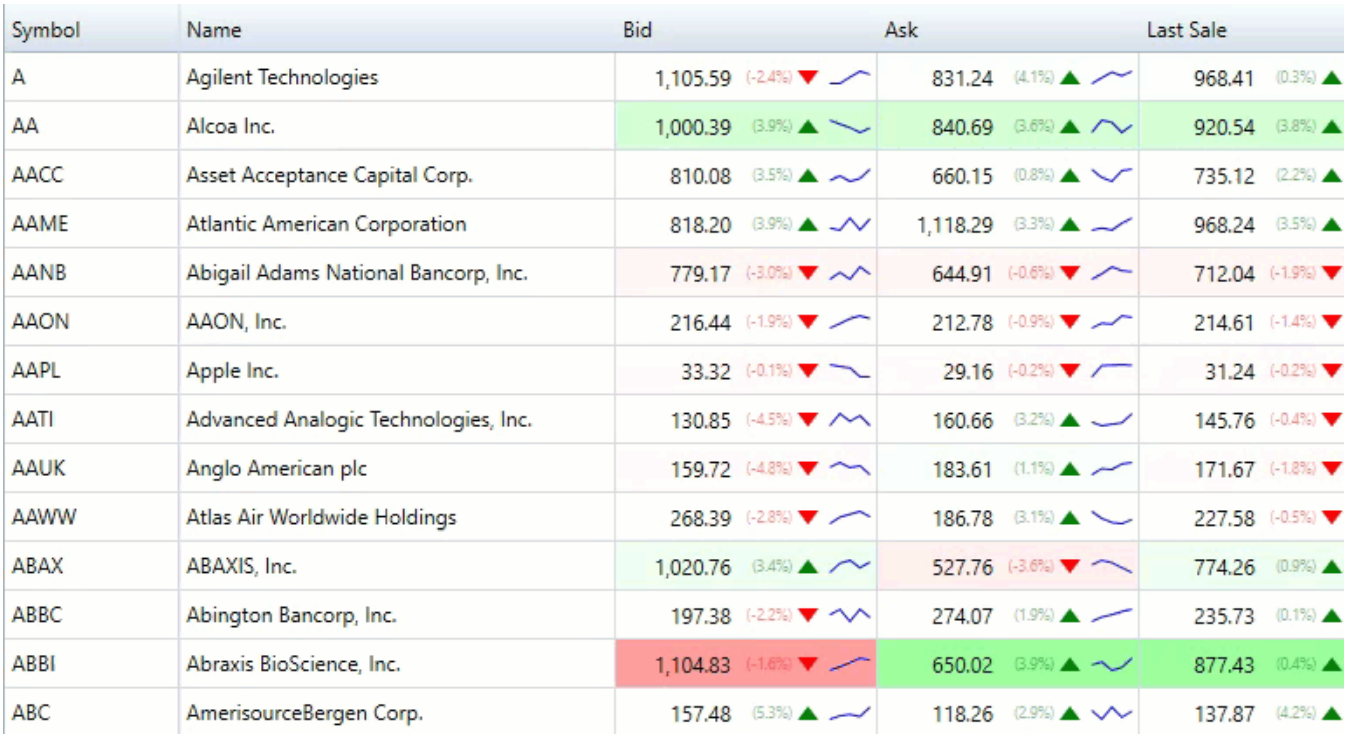
最初のメソッド **CreateCell** は、セルを表す **FrameworkElement** オブジェクトを作成します。このパラメータには、セルを所有するグリッド、作成するセルの種類、および表される CellRange が含まれます。CellType パラメータは、作成されるセルが通常のデータセル、行ヘッダーまたは列ヘッダー、グリッドの左上および右下の固定セルのどれなのかを指定します。**CreateCellEditor** メソッドは、最初のメソッドに似ていますが、セルを編集モードで作成します。最後の **DisposeCell** メソッドは、グリッドからセルが削除された後で呼び出されます。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。

.NET

FlexGrid には、グリッドに表示されるすべてのセルを作成する **GridCellFactory** クラスが含まれます。

カスタム列と同様に、カスタム GridCellFactory 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 GridCellFactory クラスは直接セルを操作するため、カスタム列より簡潔です。カスタム ICellFactory クラスは、FlexGrid クラスに含まれるデフォルトの GridCellFactory クラスを継承できるため、比較的簡単に実装することができます。デフォルトの GridCellFactory 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。

次の図に、FlexGrid で GridCellFactory を使用して作成されたカスタムセルを示します。



Symbol	Name	Bid	Ask	Last Sale
A	Agilent Technologies	1,105.59 (-2.4%)	831.24 (4.1%)	968.41 (0.3%)
AA	Alcoa Inc.	1,000.39 (3.9%)	840.69 (3.6%)	920.54 (3.8%)
AACC	Asset Acceptance Capital Corp.	810.08 (3.5%)	660.15 (0.8%)	735.12 (2.2%)
AAME	Atlantic American Corporation	818.20 (3.9%)	1,118.29 (3.3%)	968.24 (3.5%)
AANB	Abigail Adams National Bancorp, Inc.	779.17 (-3.0%)	644.91 (-0.6%)	712.04 (-1.9%)
AAON	AAON, Inc.	216.44 (-1.9%)	212.78 (-0.9%)	214.61 (-1.4%)
AAPL	Apple Inc.	33.32 (-0.1%)	29.16 (-0.2%)	31.24 (-0.2%)
AATI	Advanced Analogic Technologies, Inc.	130.85 (-4.5%)	160.66 (3.2%)	145.76 (-0.4%)
AAUK	Anglo American plc	159.72 (-4.8%)	183.61 (1.1%)	171.67 (-1.8%)
AAWW	Atlas Air Worldwide Holdings	268.39 (-2.8%)	186.78 (3.1%)	227.58 (-0.5%)
ABAX	ABAXIS, Inc.	1,020.76 (3.4%)	527.76 (-3.6%)	774.26 (0.9%)
ABBC	Abington Bancorp, Inc.	197.38 (-2.2%)	274.07 (1.9%)	235.73 (0.1%)
ABBI	Abraxis BioScience, Inc.	1,104.83 (-1.6%)	650.02 (3.9%)	877.43 (0.4%)
ABC	AmerisourceBergen Corp.	157.48 (5.3%)	118.26 (2.9%)	137.87 (4.2%)

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、WPF 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、パフォーマンスに影響を与えます。

次のコードは、**GridCellFactory** インタフェースを示しています。

C#

```
public class FinancialCellFactory : GridCellFactory
{
    public override object GetCellContentType(GridCellType cellType, GridCellRange
range)
    {
        if (cellType == GridCellType.Cell)
        {
            var c = base.Grid.Columns[range.Column];
            if (c.Binding == "LastSale" || c.Binding == "Bid" || c.Binding == "Ask")
            {
                return typeof(StockTicker);
            }
        }
```

```
        return base.GetCellContentType(cellType, range);
    }

    public override FrameworkElement CreateCellContent(GridCellType cellType,
GridCellRange range, object cellContentType)
    {
        if (cellContentType as Type == typeof(StockTicker))
            return new StockTicker();
        return base.CreateCellContent(cellType, range, cellContentType);
    }

    public override void BindCellContent(GridCellType cellType, GridCellRange range,
FrameworkElement cellContent)
    {
        var stockTicker = cellContent as StockTicker;
        if (stockTicker != null)
        {
            stockTicker.Tag = Grid.Rows[range.Row].DataItem;
            stockTicker.BindingSource = Grid.Columns[range.Column].Binding;
            stockTicker.Value = (double)(decimal)Grid[range.Row, range.Column];
        }
        else
        {
            base.BindCellContent(cellType, range, cellContent);
        }
    }
}
```

最初のメソッド **CreateCell** は、セルを表す **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、および表される CellRange が含まれます。CellType パラメータは、作成されるセルが通常のデータセル、行ヘッダーまたは列ヘッダー、グリッドの左上および右下の固定セルのどれなのかを指定します。**CreateCellEditor** メソッドは、最初のメソッドに似ていますが、セルを編集モードで作成します。最後の **DisposeCell** 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。

XAMLテンプレートを使用するカスタムセル

次のセクションでは、FlexGrid .NET 4.5.2および.NET 5バージョンでXAMLテンプレートを使用してカスタムセルを実装する方法を説明します。

.NET Framework

コードを記述するのではなく、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って編集します。

XAML

```
<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="テンプレート" 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>
```

.NET

コードを記述するのではなく、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 **GridColumn** クラスには、**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って編集します。

XAML

```
<c1:FlexGrid x:Name="_fgTemplated">
    <c1:FlexGrid.Columns>
        <!-- add a templated column -->
        <c1:GridColumn ColumnName="_colTemplated" Header="Template" Width="200">
            <!-- template for cells in display mode -->
            <c1:GridColumn.CellTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding FirstName}" Foreground="Green"
FontWeight="Bold" VerticalAlignment="Center"/>
                </DataTemplate>
            </c1:GridColumn.CellTemplate>
            <!-- template for cells in edit mode -->
            <c1:GridColumn.CellEditingTemplate>
                <DataTemplate>
                    <Grid>
                        <Grid.ColumnDefinitions>
                            <ColumnDefinition Width="Auto" />
                            <ColumnDefinition Width="*" />
                        </Grid.ColumnDefinitions>
                        <Image Source="edit_icon.png" Grid.Column="0" />
                        <TextBox Text="{Binding FirstName, Mode=TwoWay}"
Grid.Column="1" />
                    </Grid>
                </DataTemplate>
            </c1:GridColumn.CellEditingTemplate>
        </c1:GridColumn>
    </c1:FlexGrid.Columns>
</c1:FlexGrid>
```

印刷のサポート

FlexGrid は、基本的な機能と高度な印刷機能をサポートしています。印刷時にスケール、マージン、ページ数を指定して、スタイル、フォント、グラデーションを損なわずに正しくデータをレンダリングできます。

基本的な印刷
: FlexGrid で基本的な印刷を実装する方法を説明します。

高度な印刷
: FlexGrid で高度な印刷を実装する方法を説明します。

基本的な印刷

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.

Flexgrid also allows you to display the print preview of the grid content using the PrintPreview method. This method has two overloads, one of which accepts printing parameters as its parameters and the other accepts document name, page margins, scaling, and maximum number of pages to print as its parameters.

高度な印刷

印刷処理をさらに細かく制御する場合は、**GetPageImages** メソッドを使用してグリッドを自動的にいくつかの画像に分割し、各画像を個別のページにレンダリングできます。各画像はグリッドの一部を 100% 正確に表現し、スタイル、カスタム要素、各ページに繰り返し表示される行ヘッダーと列ヘッダーなどを含みます。

GetPageImages メソッドを使用して、呼び出し元は、画像を拡大縮小することもできます。これにより、グリッド全体を実際のサイズでレンダリングしたり、1 ページに収まるように、または 1 ページの幅に合わせて拡大縮小することができます。

ページ画像を取得したら、WPF の印刷のサポートを使用して、それらを柔軟にドキュメントにレンダリングできます。たとえば、複数のグリッド、チャートなどのコンテンツを含むドキュメントを作成できます。また、ヘッダーやフッターをカスタマイズしたり、レターヘッドを追加することもできます。

WPF の印刷フレームワークは異なっています。以下のセクションでは、GetPageImages を使用して印刷ドキュメントに **C1FlexGrid** をレンダリングする方法を示します。

WPF での C1FlexGrid の印刷

WPF でドキュメントを印刷する場合は、Silverlight の場合と多少異なる手順を実行する必要があります。

1. **PrintDialog** オブジェクトを作成します。
2. ダイアログボックスの **ShowDialog** メソッドが true を返す場合は、次の手順を実行します。
3. ドキュメントのコンテンツを提供する **Paginator** オブジェクトを作成します。
4. ダイアログの **PrintDocument** メソッドを呼び出します。

次のコードは、このメカニズムのサンプル実装です。また、このコードは製品サンプルのPrintingWPFを参考にしてください。

C#

C#

```
// 詳細な印刷操作
void _btnAdvancedPrint_Click(object sender, RoutedEventArgs e)
{
  var pd = new PrintDialog();
  if (pd.ShowDialog().Value)
  {
    // スケールモードと余白を取得します。
   var margin =
     _cmbMargins.SelectedIndex == 0 ? 96.0 / 4 :
     _cmbMargins.SelectedIndex == 1 ? 96.0 / 2 :
     96.0;
   var scaleMode =
     _cmbZoom.SelectedIndex == 0 ? ScaleMode.ActualSize :
     _cmbZoom.SelectedIndex == 1 ? ScaleMode.PageWidth :
     ScaleMode.SinglePage;
    // ページサイズを計算します。
    var sz = new Size(pd.PrintableAreaWidth,
                          pd.PrintableAreaHeight);
    // ページングを設定します。
    var paginator = new FlexPaginator(
      _flex, ScaleMode, sz, new  Thickness(margin), 100);
    // ドキュメントを印刷します
    pd.PrintDocument(paginator, "C1FlexGrid 印刷デモ");
  }
}
```

FlexPaginator クラスは、ページ画像を提供し、概念的には Silverlight で使用される **PrintPage** イベントハンドラに似ています。このクラスは次のように実装されます。

C#

C#

```
/// <summary>
/// C1FlexGrid コントロールをレンダリングするために使用される DocumentPaginator クラス。
/// </summary>
public class FlexPaginator : DocumentPaginator
{
  Thickness _margin;
  Size _pageSize;
  ScaleMode _scaleMode;
  List<FrameworkElement> _pages;
  public FlexPaginator(C1FlexGrid flex,
    ScaleMode scaleMode,
    Size pageSize,
    Thickness margin, int maxPages)
  {
    // 引数を保存します。
    _margin = margin;
    _scaleMode = scaleMode;
    _pageSize = pageSize;
    // グリッド画像を作成する前に、ページサイズで余白を調整します。
    pageSize.Width -= (margin.Left + margin.Right);
    pageSize.Height -= (margin.Top + margin.Bottom);
    // 各ページのグリッド画像を取得します。
    _pages = flex.GetPageImages(scaleMode, pageSize, maxPages);
  }
```

このコンストラクタは、ページ画像を作成します。これらの画像は、後で印刷フレームワークがページネータの **GetPage** メソッドを呼び出すときに、ページにレンダリングされます。

C#

C#

```
public override DocumentPage GetPage(int pageNumber)
  {
    // ページ要素を作成します。
    var pageTemplate = new PageTemplate();
    // 余白を設定します。
    pageTemplate.SetPageMargin(_margin);
    // 内容を設定します。
    pageTemplate.PageContent.Child = _pages[pageNumber];
    pageTemplate.PageContent.Stretch =
      _scaleMode == ScaleMode.ActualSize
        ? System.Windows.Media.Stretch.None
        : System.Windows.Media.Stretch.Uniform;
    // フッタテキストを設定します。
    pageTemplate.FooterRight.Text = string.Format("ページ {0} / {1}",
        pageNumber + 1, _pages.Count);
    // ページ要素を整理します。
    pageTemplate.Arrange(
        new Rect(0, 0, _pageSize.Width, _pageSize.Height));
    // 新しいドキュメントを作成します。
    return new DocumentPage(pageTemplate);
  }
```

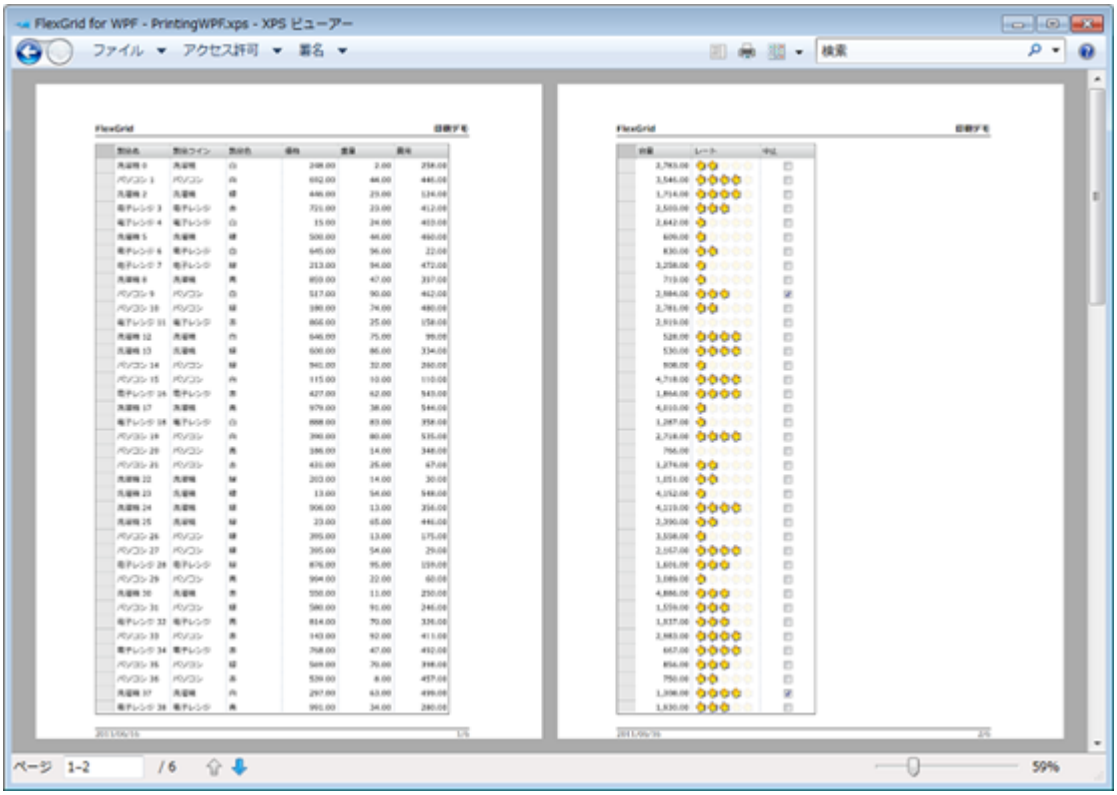
前の例と同様に、**PageTemplate** ヘルパークラスを使用して、グリッド画像を保持し、マージン、ヘッダ、およびフッタを提供します。

残りのページネータメソッドは、細かい実装を行います。

C#

```
public override int PageCount
  {
    get { return _pages.Count; }
  }
  public override IDocumentPaginatorSource Source
  {
    get { return null; }
  }
  public override Size PageSize
  {
    get { return _pageSize; }
    set { throw new NotImplementedException(); }
  }
  public override bool IsPageCountValid
  {
    get { return true; }
  }
}
```

次の図は、グリッドが XPS ファイルにレンダリングされた場合のドキュメントです。このサンプルで使用したカスタムの評価セルも含めて、正確に示されています。行および列ヘッダ、単純なページヘッダ、および標準の "ページ n/m" ページフッタは、自動的にすべてのページに含まれます。



レイアウトと外観

ComponentOne ClearStyle 技術

ComponentOne ClearStyle は、WPF コントロールのスタイル設定をすばやく簡単に実行できる新技術です。ClearStyle を使用すると、面倒な XAML テンプレートやスタイルリソースを操作しなくても、コントロールのカスタムスタイルを作成できます。

現在のところ、すべての標準 WPF コントロールにテーマを追加するには、スタイルリソーステンプレートを作成する必要があります。Microsoft Visual Studio ではこの処理は困難であるため、Microsoft は、このタスクを簡単に実行できるように Expression Blend を導入しました。ただし、Blend に不慣れであったり、十分な学習時間を取れない開発者にとっては、この 2 つの環境を行き来することはかなり困難な作業です。デザイナに作業を任せることも考えられますが、デザイナと開発者が XAML ファイルを共有すると、かえって煩雑になる可能性があります。

このような場合に、ClearStyle を使用します。ClearStyle は、Visual Studio を使用して直感的な方法でスタイル設定を実行できるようにします。ほとんどの場合は、アプリケーション内のコントロールに対して単純なスタイル変更を行うだけなので、この処理は簡単に行えるべきです。たとえば、データグリッドの行の色を変更するだけであれば、1 つのプロパティを設定するだけで簡単に行えるようにする必要があります。一部の色を変更するためだけに、完全で複雑なテンプレートを作成する必要はありません。

ClearStyle の仕組み

コントロールのスタイルの主な要素は、それぞれ単純な色プロパティとして表されます。これが集まって、コントロール固有のスタイルプロパティセットを形成します。たとえば、**Gauge** には **PointerFill** プロパティや **PointerStroke** プロパティがあり、**DataGrid** の行には **SelectedBrush** や **MouseOverBrush** があります。

たとえば、フォーム上に ClearStyle をサポートしていないコントロールがあるとします。その場合は、ClearStyle によって作成された XAML リソースを使用して、フォーム上の他のコントロールを調整して合わせることができます（正確な色合わせなど）。また、スタイルセットの一部を ClearStyle（カスタムスクロールバーなど）で上書きしたいとします。ClearStyle は拡張可能なのでこれも可能です。必要な場所でスタイルを上書きできます。

ClearStyle は、すばやく簡単にスタイルを変更することを意図したソリューションですが、ComponentOne のコントロールには引き続き従来の方法を使用して、必要なスタイルを細かく指定して作成できます。完全なカスタム設計が必要になる特別な状況で ClearStyle が邪魔になることはありません。

ClearStyle プロパティ

FlexGrid for WPF は、コントロールのテンプレートを変更することなくコントロールの色を簡単に変更できる ComponentOne の新しい ClearStyle™ 技術をサポートします。色のプロパティをいくつか設定するだけで、コントロールのスタイルを速く設定できます。

次の表に、C1FlexGrid コントロールのブラシのプロパティの概要を示します。

ブラシ	説明
Background	コントロールの背景のブラシを取得または設定します。
AlternatingRowBackground	奇数行の背景の描画に使用する Brush を取得または設定します。
BottomRightCellBackground	グリッドの右下隅にあるセルの背景の描画に使用する Brush を取得または設定します。
ColumnHeaderBackground	列ヘッダーの背景の描画に使用する Brush を取得または設定します。
ColumnHeaderForeground	列ヘッダーのコンテンツの描画に使用する Brush を取得または設定します。
ColumnHeaderSelectedBackground	選択されているセルの列ヘッダーの背景の描画に使用する Brush を取得または設定します。
CursorBackground	カーソルセルの背景の描画に使用する Brush を取得または設定します。
CursorForeground	カーソルセルのコンテンツの描画に使用する Brush を取得または設定します。

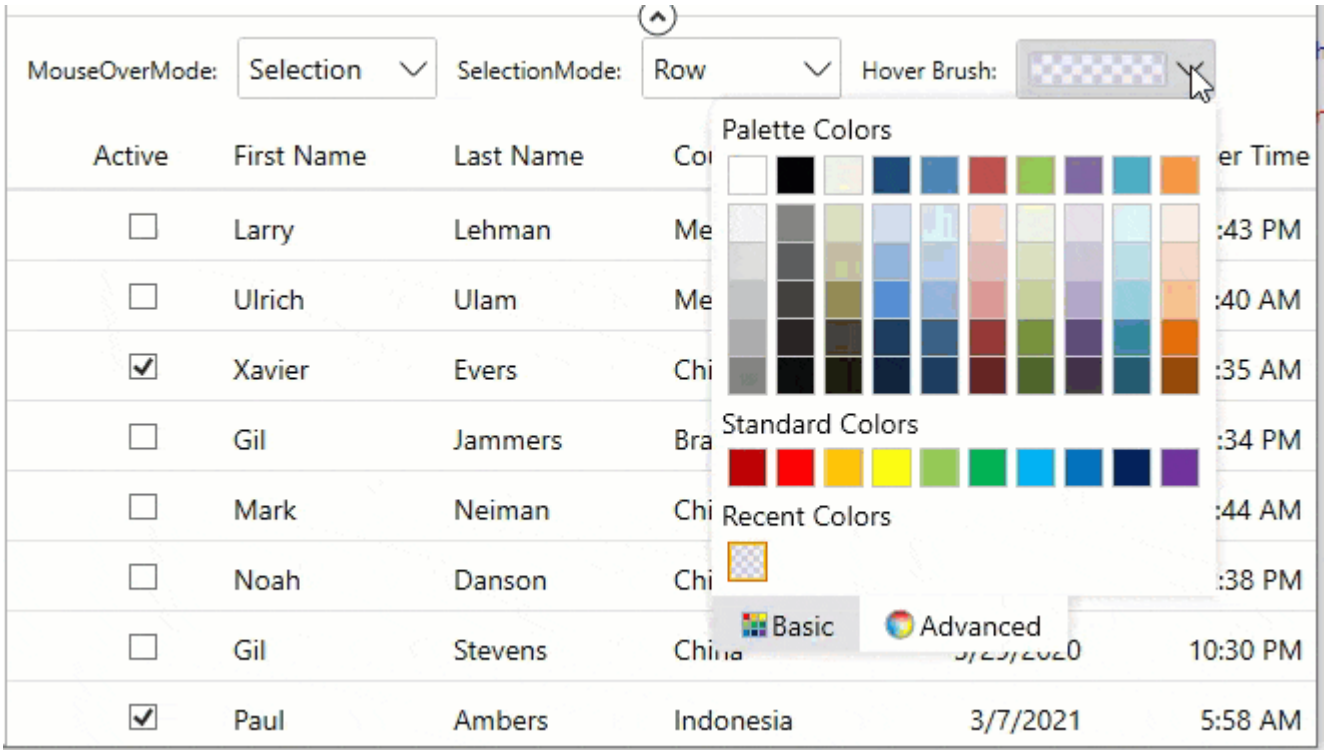
EditorBackground	編集モードのセルの背景の描画に使用する Brush を取得または設定します。
EditorForeground	編集モードのセルのコンテンツの描画に使用する Brush を取得または設定します。
FrozenLinesBrush	グリッドの固定領域とスクロール可能領域の間にある線の描画に使用する Brush を取得または設定します。
GridLinesBrush	セル間にある線の描画に使用する Brush を取得または設定します。
GroupRowBackground	グループ行の背景の描画に使用する Brush を取得または設定します。
GroupRowForeground	グループ行のコンテンツの描画に使用する Brush を取得または設定します。
HeaderGridLinesBrush	行ヘッダーセルと列ヘッダーセルの間にある線の描画に使用する Brush を取得または設定します。
RowBackground	行の背景の描画に使用する Brush を取得または設定します。
RowHeaderBackground	行ヘッダーの背景の描画に使用する Brush を取得または設定します。
RowHeaderForeground	行ヘッダーのコンテンツの描画に使用する Brush を取得または設定します。
RowHeaderSelectedBackground	選択されているセルの行ヘッダーの背景の描画に使用する Brush を取得または設定します。
SelectionBackground	カーソルセル以外の選択されているセルの背景の描画に使用する Brush を取得または設定します。
SelectionForeground	カーソルセル以外の選択されているセルのコンテンツの描画に使用する Brush を取得または設定します。
TopLeftCellBackground	グリッドの左上隅にあるセルの背景の描画に使用する Brush を取得または設定します。

いくつかのプロパティを設定することで、C1FlexGrid コントロールの外観を完全に変更できます。たとえば、**AlternatingRowBackground** プロパティを "#FFC3F2F2" に設定すると、C1FlexGrid コントロールは次のように展開されない状態になります。

ホバースタイル

Hover styles are applied to cells when the mouse hovers over the grid. It provides visual cues so that the user can apply styling to a cell, row, column or to a specific cell according to the **GridSelectionMode** enumeration when they are hovered on prior to selection or editing.

The hover styles can make the FlexGrid appear more "alive" and interactive.



There are two essential properties to consider while performing hover styles, the **MouseOverMode** property and **MouseOverBrush** property in **GridBase** class. The MouseOverMode property helps to set how the mouse hovers over the grid, while the MouseOverBrush property sets a brush to paint the background of the selected cells. Also, **SelectionMode** property is equally important as it helps to set how the cells or rows are selected in the grid, since styling after all is applied based on cell hovering before selection.

Moreover, the SelectionMode property calls the **GridSelectionMode** enumeration, the MouseOverMode property calls the **GridMouseOverMode** enumeration and the MouseOverBrush property calls the **Brush** class.

Let us see how we can set the hover styling in the designer XAML using these properties.

XAML

```
<Grid>
      <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition />
        </Grid.RowDefinitions>
        <StackPanel Orientation="Horizontal">
            <TextBlock VerticalAlignment="Center" Text="MouseOverMode:" Margin="10 0"
/>
            <c1:C1ComboBox x:Name="cbMouseOverMode" Width="100" SelectedItem="
{Binding MouseOverMode,ElementName=grid,Mode=TwoWay}">
                <c1:GridMouseOverMode>None</c1:GridMouseOverMode>
                <c1:GridMouseOverMode>Selection</c1:GridMouseOverMode>
                <c1:GridMouseOverMode>Cell</c1:GridMouseOverMode>
                <c1:GridMouseOverMode>Row</c1:GridMouseOverMode>
                <c1:GridMouseOverMode>Column</c1:GridMouseOverMode>
            </c1:C1ComboBox>
            <TextBlock VerticalAlignment="Center" Text="SelectionMode:" Margin="10 0"
/>
            <c1:C1ComboBox x:Name="cbSelectionMode" Width="100" SelectedItem="
```

```
{Binding SelectionMode,ElementName=grid,Mode=TwoWay}">
                <c1:GridSelectionMode>None</c1:GridSelectionMode>
                <c1:GridSelectionMode>Cell</c1:GridSelectionMode>
                <c1:GridSelectionMode>CellRange</c1:GridSelectionMode>
                <c1:GridSelectionMode>Row</c1:GridSelectionMode>
                <c1:GridSelectionMode>RowRange</c1:GridSelectionMode>
                <c1:GridSelectionMode>Column</c1:GridSelectionMode>
                <c1:GridSelectionMode>ColumnRange</c1:GridSelectionMode>
            </c1:C1ComboBox>
            <TextBlock VerticalAlignment="Center" Text="Hover Brush:" Margin="10 0"
/>
            <c1:C1ColorPicker Width="100" Height="30" Name="indicatorBrush"
SelectedBrush="{Binding MouseOverBrush, ElementName=grid, Mode=TwoWay}" />
        </StackPanel>
        <c1:FlexGrid Grid.Row="1" x:Name="grid" Margin="0 10 0 0"
HeadersVisibility="All" MouseOverBrush="#111700FF"
                     AutoGenerateColumns="False">
            <c1:FlexGrid.Columns>
                <c1:GridColumn Binding="Active" MinWidth="70" Width="0.5*"/>
                <c1:GridColumn Binding="FirstName" MinWidth="110" Width="*"/>
                <c1:GridColumn Binding="LastName" MinWidth="110" Width="*"/>
                <c1:GridColumn Binding="CountryId" Header="Country" MinWidth="110"
Width="*"/>
                <c1:GridDateTimeColumn Binding="LastOrderDate" Mode="Date"
MinWidth="110" Width="*" HorizontalAlignment="Right"
HeaderHorizontalAlignment="Right"/>
                <c1:GridDateTimeColumn Binding="LastOrderDate" Mode="Time"
Header="Last Order Time" MinWidth="110" Width="*" HorizontalAlignment="Right"
HeaderHorizontalAlignment="Right"/>
                <c1:GridColumn Binding="OrderTotal" Format="C" MinWidth="110"
Width="*" HorizontalAlignment="Right" HeaderHorizontalAlignment="Right"/>
            </c1:FlexGrid.Columns>
        </c1:FlexGrid>
</Grid>
```

As you can observe from the GIF, we have used two ComboBoxes, 'cbMouseOverMode' (**SelectedItem** property data-bound to MouseOverMode property of GridBase class) and 'cbSelectionMode' (**SelectedItem** property data-bound to **SelectionMode** property of GridBase class) and ColorPicker' indicatorBrush' (**SelectedBrush** data-bound to MouseOverBrush property of GridBase class).

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.
ActiveFilterIconTemplate	Allows you to set the template which is used to present the active filter icon when the column is not filtered.

InactiveFilterIconTemplate	Allows you to set the template which is used to present the inactive filter icon when the column is filtered.
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
24	Zeb Stevens	Italy	☑	Zeb	Stevens
86	Zeb Quaid	Mexico	☑	Zeb	Quaid
37	Zeb Lehman	Italy	☐	Zeb	Lehman
39	Zeb Frommer	Italy	☐	Zeb	Frommer
71	Zeb Danson	Turkey	☑	Zeb	Danson
10	Xavier Neiman	Mexico	☐	Xavier	Neiman
60	Xavier Lehman	France	☐	Xavier	Lehman
38	Xavier Krause	Egypt	☑	Xavier	Krause
49	Xavier Frommer	Pakistan	☐	Xavier	Frommer
78	Xavier Frommer	Philippines	☑	Xavier	Frommer
96	Xavier Cole	Italy	☑	Xavier	Cole
9	Vic Stevens	Brazil	☐	Vic	Stevens
19	Vic Richards	Pakistan	☐	Vic	Richards
20	Vic Griswold	Mexico	☑	Vic	Griswold
23	Vic Bishop	Ethiopia	☐	Vic	Bishop
36	Ulrich Richards	United States	☑	Ulrich	Richards
28	Ulrich Griswold	United Kingdom	☑	Ulrich	Griswold
5	Ulrich Evers	Mexico	☐	Ulrich	Evers
14	Ulrich Cole	Thailand	☐	Ulrich	Cole
48	Ted Stevens	Iran	☑	Ted	Stevens
53	Ted Richards	Nigeria	☐	Ted	Richards
30	Ted Neiman	Japan	☑	Ted	Neiman
4	Ted Myers	Nigeria	☐	Ted	Myers

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
Path	C1PolygonIcon class (child class of C1VectorIcon class)

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

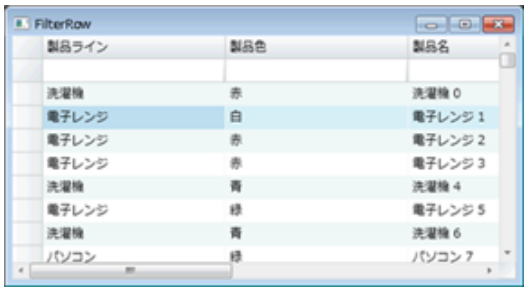
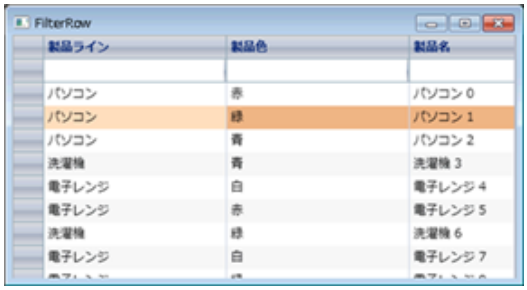
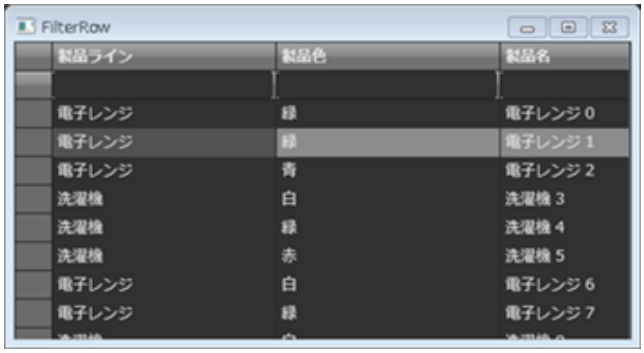
C#

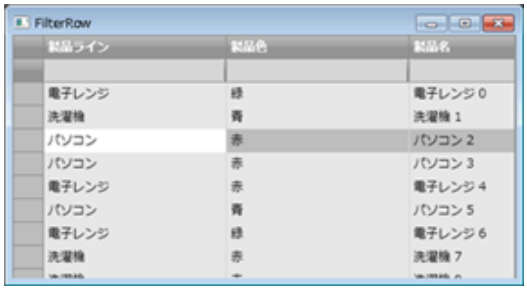
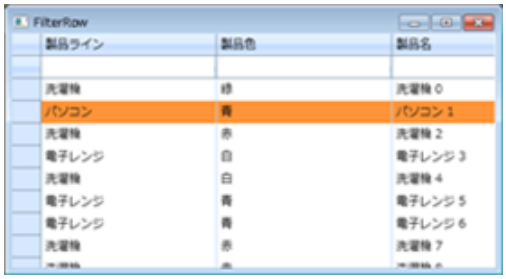
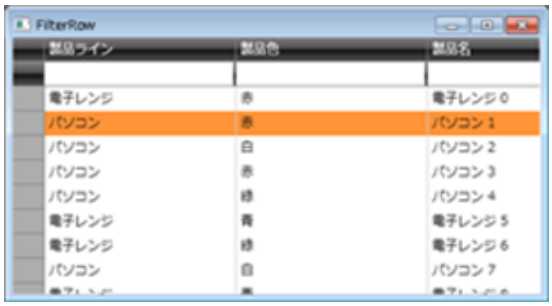
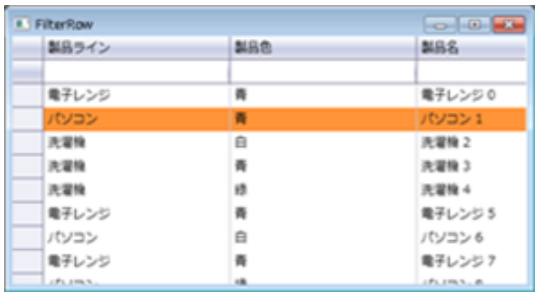
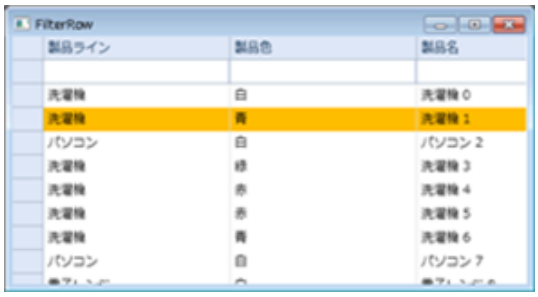
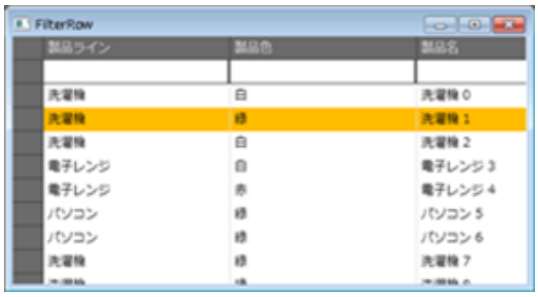
```
BitmapImage imgDown = new BitmapImage();
 imgDown.BeginInit();
 imgDown.UriSource = new Uri("icons/arrow_down.png", UriKind.Relative);
 imgDown.EndInit();

 _flexGrid.SortDescendingIconTemplate = new C1IconTemplate(() => new C1BitmapIcon()
 {
      Source = imgDown,
      Width = 20,
      Height = 20
 });
```

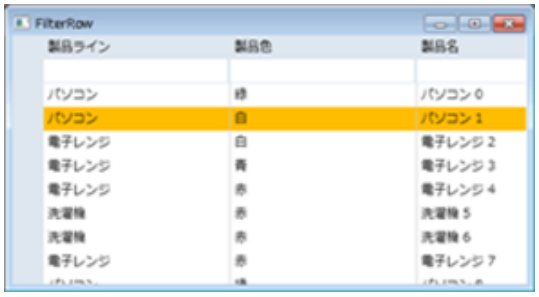
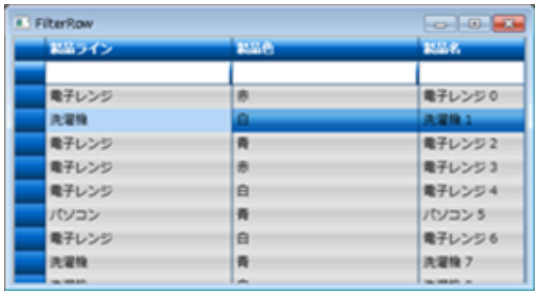
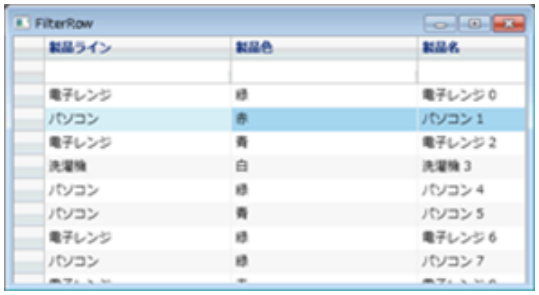
テーマ

FlexGrid for WPF には、グリッドの外観をカスタマイズできるいくつかのテーマが組み込まれています。C1FlexGrid コントロールを初めてページに追加すると、次の図のように表示されます。

これは、このコントロールのデフォルトの外観です。この外観は、組み込みテーマの 1 つを使用したり、独自のカスタムテーマを作成することで変更できます。すべての組み込みテーマは、WPF Toolkit テーマに基づいています。以下に、組み込みテーマの説明と図を示します。以下の図では、選択状態のスタイルを示すために 1 つの行が選択されています。

テーマ名	テーマのプレビュー
C1Blue	
C1ThemeBureauBlack	
C1ThemeExpressionDark	

C1ThemeExpressionLight	
C1ThemeOffice2007Blue	
C1ThemeOffice2007Black	
C1ThemeOffice2007Silver	
C1ThemeOffice2010Blue	
C1ThemeOffice2010Black	

C1ThemeOffice2010Silver	
C1ThemeShinyBlue	
C1ThemeWhistlerBlue	

要素のテーマを設定するには、**ApplyTheme** メソッドを使用します。最初に、テーマアセンブリへの参照をプロジェクトに追加し、次のようにコードでテーマを設定します。

Visual Basic

```
Private Sub Window_Loaded(sender As System.Object, e As System.Windows.RoutedEventArgs)
Handles MyBase.Loaded
    Dim theme As New C1ThemeExpressionDark
    ' ApplyTheme の使用
    C1Theme.ApplyTheme(LayoutRoot, theme)
```

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
    {
        C1ThemeExpressionDark theme = new C1ThemeExpressionDark();
        //ApplyTheme の使用
        C1Theme.ApplyTheme(LayoutRoot, theme);
    }
```

アプリケーション全体にテーマを適用するには、**System.Windows.ResourceDictionary.MergedDictionaries** プロパティを使用します。最初に、テーマアセンブリへの参照をプロジェクトに追加し、次のようにコードでテーマを設定します。

Visual Basic

```
Private Sub Window_Loaded(sender As System.Object, e As System.Windows.RoutedEventArgs)
Handles MyBase.Loaded
   Dim theme As New C1ThemeExpressionDark
   ' MergedDictionaries の使用
Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThemeResources(theme))
End Sub
```

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
   {
      C1ThemeExpressionDark theme = new C1ThemeExpressionDark();
      //MergedDictionaries の使用
Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThemeResources(theme));
   }
```

この方法は、初めてテーマを適用する場合にのみ使用できることに注意してください。別の ComponentOne テーマに切り替える場合は、最初に、**Application.Current.Resources.MergedDictionaries** から前のテーマを削除します。

ミュージックライブラリサンプル

製品に付属の **FlexGridSamples** サンプルにある「ミュージックライブラリ」は、約 10,000 曲のライブラリをアーティストとアルバムによってグループ化して表示します。アルバムとアーティストは、折りたたみ可能なノード行によって示されます。このアプリケーションは、ユーザーがすばやく簡単に曲、アルバム、またはアーティストを探すための検索ボックスを備えています。フィルタを適用すると、ステータスインジケータには、選択されたアーティスト、アルバム、および曲と共に、選択内容のサイズと再生時間の合計が表示されます。

グリッドの各項目は、アーティスト、アルバム、または曲を表し、タイトル、再生時間、サイズ、およびレーティングが含まれます。0 ～ 5 の間の整数で表されるレーティングデータは、0 ～ 5 個の星を使ってグラフィカル表示されます。グループ行では、展開/折りたたみのボタンに（標準の三角形のアイコンの代わりに）従来のプラスとマイナスのアイコンが使用されます。アーティストとアルバムについて表示される再生時間とサイズは、対応するグループ（アルバムまたはアーティスト）のすべての曲を足し合わせることによって動的に計算されます。アーティストとアルバムのレーティングは、対応するグループ（アルバムまたはアーティスト）の曲のレーティングを平均することによって計算されます。

グリッドの上には、すべてのグループ行を折りたたんでアーティストまたはアルバムのみを表示するためのボタンと、展開してグリッド全体を表示するためのボタンがあります。

グリッドの作成

次に、C1FlexGrid を定義する XAML を示します。

XAML for WPF

```
<!-- FlexGrid に曲を表示します -->
<c1:C1FlexGrid x:Name="_flexiTunes" Grid.Row="2"
    Style="{StaticResource gridStyle}"
    IsReadOnly="true"
    AreRowGroupHeadersFrozen="False"
    HeadersVisibility="Column"
    GridLinesVisibility="None"
    Background="White"
    RowBackground="White"
    AlternatingRowBackground="White"
    GroupRowBackground="White"
    MinColumnWidth="30"
    SelectionMode="ListBox"
    SelectionBackground="#a0eaeff4"
    CursorBackground="#ffeaeff4"
    AutoGenerateColumns="False" >
    <c1:C1FlexGrid.Columns>
        <c1:Column Binding="{Binding Name}" Header="Title" AllowDragging="False"
Width="300" />
        <c1:Column Binding="{Binding Duration}" HorizontalAlignment="Right" />
        <c1:Column Binding="{Binding Size}" HorizontalAlignment="Right" />
        <c1:Column Binding="{Binding Rating, Mode=TwoWay}"
HorizontalAlignment="Left" Width="200" />
    </c1:C1FlexGrid.Columns>
</c1:C1FlexGrid>
```

XAML でスタイルに関するプロパティをいくつか設定します。グリッドではいくつかのプロパティが公開されているため、カスタムセルや XAML テンプレートを使用しなくても外観のかなりの部分をカスタマイズできることを認識してください。

コードでは、**AutoGenerateColumns** プロパティを False に設定し、列を明示的に定義していることにも注意してください。これにより、列の外観と動作をさらに制御することができます。

最後に、この XAML コードは **AreRowGroupHeadersFrozen** プロパティを False に設定します。通常の場合、グループ行は水平方向にスクロールしません。そのため、グループ情報は常に表示されています。しかし、このミュージックライブラリアプリケーションでは、グループ行のすべての列に関する情報が表示されるので、他のすべての行と同様にこの行も水平方向にスクロールできるようにする必要があります。それには、**AreRowGroupHeadersFrozen** プロパティを False に設定します。

データの読み込み

グリッドのデータは、**ICollectionView** オブジェクトに簡単に読み込むことができます。コードは次のようになります。

WPF

```
var songs = MediaLibrary.Load();
var view = new MyCollectionView(songs);
using (view.DeferRefresh())
{
  view.GroupDescriptions.Clear();
  view.GroupDescriptions.Add(new PropertyGroupDescription("Artist"));
  view.GroupDescriptions.Add(new PropertyGroupDescription("Album"));
}
var fg = _flexiTunes;
fg.CellFactory = new MusicCellFactory();
fg.MergeManager = null;
fg.Columns["Duration"].ValueConverter = new SongDurationConverter();
fg.Columns["Size"].ValueConverter = new SongSizeConverter();
fg.ItemsSource = view;
```

このコードは、リソースとして格納されている XML ファイルから曲を読み込む **Load** ヘルパーメソッドを使用しています。実際のアプリケーションでは、この部分はサーバーから曲目カタログを読み込む Web サービスに置き換えると考えられます。次に、**Load** メソッドの実装を示します。

WPF

```
public static List<Song> Load()
{
    var asm = Assembly.GetExecutingAssembly();
    foreach (var resName in asm.GetManifestResourceNames())
    {
        if (resName.EndsWith("data.zip"))
        {
            var zip = new
C1.C1Zip.C1ZipFile(asm.GetManifestResourceStream(resName));
            using (var stream = zip.Entries["songs.xml"].OpenReader())
            {
                var xmls = new XmlSerializer(typeof(List<Song>));
                return (List<Song>)xmls.Deserialize(stream);
            }
        }
    }
    throw new Exception("組み込みリソース「data.zip」が見つかりません。");
}
```

Song クラスには、曲の再生時間（ミリ秒）とサイズ（バイト）が格納されます。これはユーザーへの表示に適した書式ではなく、また、ここでの目的に適した形式に値を変換できるシンプルな .NET 書式文字列はありません。上のコードでは、**Column.Format** プロパティを設定する代わりに、**Duration** 列と **Size** 列に **Column.ValueConverter** プロパティを設定します。

Column.ValueConverter プロパティには、未加工データ値を列に表示する値に変換するために使用する **IValueConverter** オブジェクトを指定します。次に、曲の再生時間（ミリ秒）とサイズ（バイト）のコンバータの実装のサンプルを示します。

WPF

```
// 曲の期間のコンバーター(ミリ秒)。
class SongDurationConverter : IValueConverter
{
  public object Convert(object value, Type targetType,
    object parameter,
    System.Globalization.CultureInfo culture)
  {
    var ts = TimeSpan.FromMilliseconds((long)value);
    return ts.Hours == 0
      ? string.Format("{0:00}:{1:00}", ts.Minutes, ts.Seconds)
      : string.Format("{0:00}:{1:00}:{2:00}",
                ts.Hours, ts.Minutes, ts.Seconds);
  }
  public object ConvertBack(object value, Type targetType,
    object parameter,
    System.Globalization.CultureInfo culture)
  {
    throw new NotImplementedException();
  }
}
// 曲のサイズのコンバーター(x.xx MB を返す)。
class SongSizeConverter : IValueConverter
{
  public object Convert(object value, Type targetType,
    object parameter,
    System.Globalization.CultureInfo culture)
  {
    return string.Format("{0:n2} MB", (long)value / 1024.0 / 1024.0);
  }
  public object ConvertBack(object value, Type targetType,
    object parameter,
    System.Globalization.CultureInfo culture)
  {
    throw new NotImplementedException();
  }
}
```

最初の値コンバータは、1 時間を超える再生時間に異なる書式を使用します。2 番目の値コンバータは、バイトをメガバイトに変換します。**IValueConverter** オブジェクトは柔軟でシンプルなので、WPF プログラミングではよく使用されます。

データ連結に関して実行する必要があるのは、これですべてです。以上で、適切な **ICollectionView** データソース、便利な書式を使って曲名、再生時間、およびサイズを表示する列の準備ができました。

グループ化

データ連結コードは、データソースの **GroupDescriptions** プロパティに値を挿入することにより、既に基本的なグループ化機能をカバーしています。これによってグリッドは、アルバムとアーティストによって曲をグループ化し、グループに関する基本情報を含む折りたたみ可能なグループ行を表示することができます。

ここでは、カタログを自動的に折りたたみ/展開するボタンを追加して、アーティストのみ(完全に折りたたまれたグループ)、アーティストとアルバム(中間状態)、またはアーティスト、アルバム、および曲(完全に展開されたグループ)を表示することに

より、グループ化機能をさらに強化します。

そのために、グリッド上にアーティスト、アルバム、および曲の 3 つのボタンを追加しました。これらのボタンのイベントハンドラは、以下のように実装されます。

WPF

```
// グループの展開・折りたたみ
void _btnShowArtists_Click(object sender, RoutedEventArgs e)
{
  ShowOutline(0);
}
void _btnShowAlbums_Click(object sender, RoutedEventArgs e)
{
  ShowOutline(1);
}
void _btnShowSongs_Click(object sender, RoutedEventArgs e)
{
  ShowOutline(int.MaxValue);
}
```

これによってわかるように、すべてのイベントハンドラは同じ **ShowOutline** ヘルパーメソッドを使用します。最初のボタンはレベルゼロのすべてのグループ行(アーティスト)を折りたたみ、2 つ目のボタンはレベルゼロのグループ(アーティスト)を展開してレベル 1(アルバム)を折りたたみます。3 つ目のボタンはすべてのグループ行を展開します。次に、**ShowOutline** メソッドの実装を示します。

WPF

```
void ShowOutline(int level)
{
  var rows = _flexiTunes.Rows;
  using (rows.DeferNotifications())
  {
    foreach (var gr in rows.OfType<GroupRow>())
    {
      gr.IsCollapsed = gr.Level >= level;
    }
  }
}
```

このメソッドは簡単です。まず、グリッドの **RowCollection** クラスを取得し、**ICollectionView** インターフェイスで使用したメカニズムと同じ **DeferNotifications** を実装します。このメカニズムは、Windows フォームアプリケーションでよく使用される **BeginUpdate/EndUpdate** パターンに似ており、パフォーマンスの大幅な向上をもたらします。

次に、このコードは LINQ **OfType** 演算子を使用して、**Rows** コレクションからすべての **GroupRow** オブジェクトを取得します。これは通常の行が自動的に除外されるので、すべてのグループ行のレベルをチェックし、各グループ行のレベルに基づいてその **IsCollapsed** 状態を更新できます。

検索とフィルタ処理

このサンプルには数千曲が含まれています。曲はアーティストとアルバムによって分類されていますが、スクロールして特定の曲を探すことは実用的ではありません。

このサンプルでは、検索機能を実装することにより、これに対処します。検索ボックスに文字列を入力すると、コンテンツが自動的にフィルタされて、指定した文字列を含むタイトルの曲、アーティスト、またはアルバムのみが表示されます。ここでは、**SearchBox** コントロールの外観と動作について簡単に説明します。

このコントロールには、次の 2 つのプロパティがあります。

- **View**:フィルタ処理するデータを含む **ICollectionView** オブジェクト。
- **FilterProperties**:フィルタが適用されたときに、使用するデータ項目のプロパティを指定する **PropertyInfo** オブジェクトのリスト。

ユーザーが **SearchBox** コントロールに入力すると、タイマーが動き始めます。入力が短時間(800 ミリ秒)中断すると、フィルタがデータソースに適用されます。すべての文字について入力の直後にフィルタ処理すると、効率が悪く、ユーザーにとっては煩わしいので、行われません。

この処理に使用されるタイマーのクラスは、**C1FlexGrid** アセンブリで定義されます。

フィルタを適用するために、**SearchBox** コントロールは **View.Filter** プロパティに対して、各データ項目をチェックするメソッドを設定し、**FilterProperties** メンバによって指定されたプロパティの少なくとも 1 つに、ユーザーが入力した検索文字列が含まれている項目のみを保持します。

コードは、次のようになります。

WPF

```
void _timer_Tick(object sender, EventArgs e)
{
    _timer.Stop();
  if (View != null && _propertyInfo.Count > 0)
  {
    View.Filter = null;
    View.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、すべての項目を表示します。
      if (string.IsNullOrEmpty(srch))
      {
        return true;
      }
      // 任意の指定したプロパティのテキストを含む項目を表示します。
      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;
    };
  }
}
```

このコードで注目する部分は、フィルタを適用するブロックです。そこでは、通常のメソッドの代わりにラムダ関数を使用していますが、効果は同じです。ラムダ関数は、リフレクションを使用することにより、指定された各プロパティの値を取得し、検索文字列を含むかどうかをチェックします。一致が見つかると、その項目が表示されます。見つからなければ、フィルタで除外されます。

SearchBox コントロールをアプリケーションに接続するには、**View** プロパティと **FilterProperties** プロパティを設定する必要があります。このサンプルでは、**View** プロパティを曲のデータソースに設定し、*Artist*、*Album*、*Name* の各プロパティを **FilterProperties** コレクションに追加して、ユーザーがこれらの要素を検索できるようにします。**SearchBox** をアプリケーションに接続するコードは次のとおりです。

WPF

```
//「検索」ボックスを設定します。
_srchTunes.View = view;
foreach (string name in "Artist|Album|Name".Split('|'))
{
  _srchTunes.FilterProperties.Add(typeof(Song).GetProperty(name));
}
```

カスタムセル

これで、アプリケーションで最も注目する部分を作成する準備ができました。以下の要素を表示するためのカスタムセルを作成する **ICellFactory** オブジェクトを作成します。

- アーティスト、アルバム、および曲の横に表示する画像
- グループの折りたたみと展開のためのカスタム画像
- 曲、アルバム、およびアーティストのレーティングを表示するためのカスタム画像

これらすべての作業は、**ICellFactory** インターフェイスを実装するカスタム **MusicCellFactory** クラスによって実行されます。カスタムセルファクトリは、それをグリッドの **CellFactory** プロパティに割り当てるだけで使用できます。

WPF

```
_flexiTunes.CellFactory = new MusicCellFactory()
```

MusicCellFactory クラスはデフォルトの **CellFactory** クラスから継承され、**CreateCellContent** メソッドをオーバーライドすることにより、各セルのコンテンツを表示するために使用する要素を作成します。

CreateCellContent メソッドは、パラメータとして、親グリッド、基本クラスによって作成されてセルの背景と境界を提供する **Border** 要素、および作成する必要があるセル（単一のセル、またはマージされている場合は複数のセル）を指定する **CellRange** オブジェクトを受け取ります。

CreateCellContent は、セルコンテンツのみを作成します。境界と背景も含めてセル全体を作成する場合は、代わりに **CreateCell** メソッドをオーバーライドします。

この例では、**CreateCellContent** メソッドは、通常のセルとグループ行内のセルという 2 つの主要なケースを処理します。

通常のセルの処理は簡単です。メソッドは、作成される列に基づいて新しい **SongCell** 要素または **RatingCell** 要素を返すだけです。これらは **StackPanel** から派生するカスタム要素で、セルが表すデータ項目に連結される画像とテキストを含みます。

グループ行のセルは、多少複雑です。このアプリケーションでは、グループ行はアーティストとアルバムを表すために使用されます。これらの項目は、ソースコレクションのデータ項目に対応していません。ここで使用する **CreateCellContent** メソッドは、フェイクの **Song** オブジェクトを作成してアーティストとアルバムを表します。このフェイクの **Song** オブジェクトには、グループに含まれる曲に基づいて計算されたプロパティが含まれます。アルバムの **Duration** は、アルバムの各曲の **Duration** の合計として計算され、**Rating** は、アルバムの各曲の **Rating** の平均として計算されます。このフェイクの Song オブジェクトが作成されると、**SongCell** 要素と **RatingCell** 要素を通常どおりに使用できます。

次に、**MusicCellFactory** クラスとその **CreateCellContent** の実装を多少簡略化して示します。

WPF

```
// ミュージックライブラリ セルを作成するために使用されるセルファクトリー。
public class MusicCellFactory : CellFactory
{
  static Thickness _emptyThickness = new Thickness(0);
  public List<RatingCell> _ratings = new List<RatingCell>();
  // セルを範囲に連結します。
  public override void CreateCellContent(
         C1FlexGrid grid, Border bdr, CellRange range)
  {
    // 行・列を取得します。
    var row = grid.Rows[range.Row];
    var col = grid.Columns[range.Column];
    var gr = row as GroupRow;
    // グループ行に罫線を表示しません。
    if (gr != null)
    {
      bdr.BorderThickness = _emptyThickness;
    }
    // ツリーのセルを連結します。
    if (gr != null && range.Column == 0)
    {
    BindGroupRowCell(grid, bdr, range);
    return;
    }
    // 標準のデータ行のセルを連結します。
    var colName = col.ColumnName;
    if (colName == "Name")
    {
      bdr.Child = new SongCell(row);
      return;
    }
    if (colName == "Rating")
    {
      var song = row.DataItem as Song;
      if (song != null)
      {
       // このセルを表すレートコントロールを作成します。
       // 注:
       // - データコンテキストとして罫線要素を使用します。
       // - 連結するために列を使用します。
       var cell = new RatingCell();
       cell.SetBinding(RatingCell.RatingProperty, col.Binding);
       bdr.Child = Cell;
       return;
      }
    }
    // デフォルトの連結
    base.CreateCellContent(grid, bdr, range);
  }
```

BindGroupRowCell メソッドは、前述のフェイクの **Song** オブジェクトを作成し、それらを行の **DataItem** プロパティに割り当てて **CreateCellContent** メソッドで使用できるようにし、グループ行の最初のセルに特別な処理を行います。各グループ行の最初のセルは他と異なり、通常のセルのコンテンツに加えてグループの折りたたみ/展開のボタンが含まれています。

次に、各グループ行の最初の項目を処理するコードを示します。

WPF

```
// グループ行にセルを連結します。
void BindGroupRowCell(C1FlexGrid grid, Border bdr, CellRange range)
{
  // 行とグループ行を取得します。
  var row = grid.Rows[range.Row];
  var gr = row as GroupRow;
  // グループのキャプション・画像を最初列に表示します。
  if (range.Column == 0)
  {
    // 必要であれば、カスタムデータアイテムを作成します。
    if (gr.DataItem == null)
    {
      gr.DataItem = BuildGroupDataItem(gr);
    }
    // 必要なセル型を取得します。
    Type cellType = gr.Level == 0 ? typeof(ArtistCell) : typeof(AlbumCell);
    // セルを作成します。
    bdr.Child = gr.Level == 0
      ? (ImageCell)new ArtistCell(row)
      : (ImageCell)new AlbumCell(row);
  }
}
```

最後に、アーティストとアルバムを表す **Song** オブジェクトを作成するコードを示します。このメソッドでは、**GroupRow.GetDataItems** メソッドを使用して、グループに含まれるすべてのデータ項目のリストを取得します。次に、LINQ ステートメントを使用して、グループ内の曲の合計サイズ、再生時間、およびレーティングの平均を計算します。

WPF

```
// グループを示す曲を作成します。
// GetChildDataItems メソッドはこのノードに付属するすべての曲を返します。
// 以下の LINQ ステートメントは、アルバム・アーティスト別にの合計のサイズ、長さおよび
// 平均レートを計算します。
Song BuildGroupDataItem(GroupRow gr)
{
  var gs = gr.GetDataItems().OfType<Song>();
  return new Song()
  {
    Name = gr.Group.Name.ToString(),
    Size = (long)gs.Sum(s => s.Size),
    Duration = (long)gs.Sum(s => s.Duration),
    Rating = (int)(gs.Average(s => s.Rating) + 0.5)
  };
}
```

Silverlight

```
// グループを示す曲を作成します。

// GetChildDataItems メソッドはこのノードに付属するすべての曲を返します。

// 以下の LINQ ステートメントは、アルバム・アーティスト別にの合計のサイズ、長さおよび

// 平均レートを計算します。

Song BuildGroupDataItem(GroupRow gr)

{

  var gs = gr.GetDataItems().OfType<Song>();

  return new Song()

  {

    Name = gr.Group.Name.ToString(),

    Size = (long)gs.Sum(s => s.Size),

    Duration = (long)gs.Sum(s => s.Duration),

    Rating = (int)(gs.Average(s => s.Rating) + 0.5)

  };

}
```

これが必要な作業の大部分です。残っている部分は、個々のセルを表すために使用するカスタム要素の定義だけです。次の要素があります。

- **SongCell**:「曲のアイコン」および曲名を表示します。
- **ArtistCell**:折りたたみ/展開のアイコン、「アーティストのアイコン」、およびアーティスト名を表示します。
- **AlbumCell**:折りたたみ/展開のアイコン、「アルバムのアイコン」、およびアーティスト名を表示します。
- **RatingCell**:レーティング(0 ～ 5 の整数)をグラフィカル要素として表示します(1 つの星が 1 つのレーティングポイントを表します)。

次の図に、グリッドにどのように要素が表示されるかを示します。

これらはすべて通常のWPF Framework の要素で、Microsoft Blend またはコードで作成できます。

次に、**RatingCell** 要素を実装するコードを示します。他の要素も同様で、サンプルソースコードを確認すれば実装の詳細がわかります。

WPF

```
///
/// 星付き画像として表示されるレートを示すセル。
///
public class RatingCell : StackPanel
{
  static ImageSource _star;
  const int MAXRATING = 5;
  const double OFF = 0.2;
  const double ON = 1.0;
  ///
  /// <see cref="ItemsSource"/> 依存プロパティを示します。
  ///
  public static readonly DependencyProperty RatingProperty =
      DependencyProperty.Register(
        "Rating",
          typeof(int),
          typeof(RatingCell),
          new PropertyMetadata(0, OnRatingChanged));
  public RatingCell()
  {
    if (_star == null)
    {
      _star = ImageCell.GetImageSource("star.png");
    }
    Orientation = Orientation.Horizontal;
    for (int i = 0; i < 5; i++)
    {
     var img = GetStarImage();
     img.Opacity = OFF;
     img.MouseLeftButtonDown += img_MouseLeftButtonDown;
     Children.Add(img);
    }
  }
  public int Rating
  {
      get { return (int)GetValue(RatingProperty); }
      set { SetValue(RatingProperty, value); }
   }
   void img_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
   {
      // 星レーティングのインデックスに基づいたレートを計算します。
      Image img = sender as Image;
      RatingCell cell = img.Parent as RatingCell;
      int index = cell.Children.IndexOf(img);
      if (index > 0 || e.GetPosition(img).X > img.Width / 3)
      {
           index++;
      }
```

```
        // 新しいレートを適用します。
        cell.Rating = index;
        Animate(img);
    }
    static Image GetStarImage()
    {
      var img = new Image();
      img.Source = _star;
      img.Width = img.Height = 17;
      img.Stretch = Stretch.None;
      return img;
    }
}
```

Silverlight

```
///
/// 星付き画像として表示されるレートを示すセル。
///
public class RatingCell : StackPanel
{
  static ImageSource _star;
  const int MAXRATING = 5;
  const double OFF = 0.2;
  const double ON = 1.0;

  //
  // 依存プロパティを示します。
  //
  public static readonly DependencyProperty RatingProperty =
     DependencyProperty.Register(
          "Rating",
           typeof(int),
           typeof(RatingCell),
           new PropertyMetadata(0, OnRatingChanged));

  public RatingCell()
  {
    if (_star == null)
    {
      _star = ImageCell.GetImageSource("star.png");
```

```
    }
    Orientation = Orientation.Horizontal;
    for (int i = 0; i < 5; i++)
    {
      var img = GetStarImage();
      img.Opacity = OFF;
      img.MouseLeftButtonDown += img_MouseLeftButtonDown;
      Children.Add(img);
    }
  }
  static Image GetStarImage()
  {
    var img = new Image();
    img.Source = _star;
    img.Width = img.Height = 17;
    img.Stretch = Stretch.None;
    return img;
  }
}
```

RatingCell 要素は簡単です。これは、いくつかの **Image** 要素を含む **StackPanel** で構成されます。**Image** 要素の数は、表示するレーティング（コンストラクタに渡される値）によって定義されます。各 **Image** 要素によって 1 つの星アイコンが表示されます。これは、レーティングが変化しないことを前提とした静的な処理なので、動的な連結は必要ありません。

株価情報サンプル

このセクションでは、株価情報に特化した Financial サンプルアプリケーションについて説明します。株価情報アプリケーションは、通常、大量のデータに依存します。データは、多くの場合、強力な専用サーバーからリアルタイムで取得されます。

この種のアプリケーションは、その特性からリアルタイム情報を処理するので、迅速な更新（アプリケーションはサーバーから取得されるデータストリームに追従する必要があります）、およびクリアで効率的な方法で変化をユーザーに伝達するメカニズムが必要です。

変更をリアルタイムで強調するために一般的に使用される方法の 1 つに、要素の点滅があります。たとえば、値が変化したときにグリッドセルを別の色で点滅させます。点滅は、短時間だけでユーザーの注意を変更に対して向けさせることができます。

コンパクトな形式で迅速に豊富な内容の情報を伝達するもう 1 つのメカニズムとして、最近さらに普及しているスパークラインがあります。スパークラインは、長い数値の行より明確で効率的に、トレンドやサマリー情報を表示するミニチャートです。

このセクションでは、リアルタイムのデータ更新、点滅セル、およびスパークラインを使用する株価情報アプリケーションについて説明します。このサンプルアプリケーションでは、WPF における **C1FlexGrid** のパフォーマンスに焦点を合わせます。

次の図に、動作中の株価情報アプリケーションを示します。図では、値が絶えず変化するアプリケーションの動的な動作を伝えることはできないので、機会があるときにサンプルアプリケーションを実行することをお勧めします。

データの生成

この株価情報アプリケーションは、常に更新される動的なデータを提供して実際のサーバーをシミュレートするデータソースを使用します。

動的な特性を持つデータを取得するために、このデータソースオブジェクトは **FinancialDataList** に対して、約 4,000 の **FinancialData** オブジェクト、および指定されたスケジュールに基づいてオブジェクトを変更するタイマーを提供します。呼び出し元は、値の更新頻度および一度に更新する数を指定できます。

FinancialDataList をグリッドに連結し、プログラムの実行中に更新パラメータを変更することにより、データ更新に対してグリッドが対応するパフォーマンスをチェックできます。

データソースの実装の詳細を確認するには、サンプルソースの **FinancialData.cs** ファイルを参照してください。

グリッドは、株価情報データソースに簡単に連結できます。グリッドを **FinancialDataList** に直接連結する代わりに、ここでは、仲介として機能して通常の通貨、ソート、グループ化、フィルタ処理のサービスを提供する **ListCollectionView** を作成します。このコードを次に示します。

WPF

```
// データソースを作成します。
FinancialDataList list = FinancialData.GetFinancialData();
var view = new MyCollectionView(list);
// グリッドにデータソースを連結します。
_flexFinancial.ItemsSource = view;
```

前のサンプルと同様に、**AutoGenerateColumns** プロパティを False に設定し、XAML を使ってグリッド列を作成します。

XAML for WPF

```
<fg:C1FlexGrid x:Name="_flexFinancial"
  MinColumnWidth="10"
  MaxColumnWidth="300"
  AutoGenerateColumns="False" >
  <fg:C1FlexGrid.Columns>
    <fg:Column Binding="{Binding Symbol}"    Width="100" />
    <fg:Column Binding="{Binding Name}"      Width="250" />
    <fg:Column Binding="{Binding Bid}"       Width="150"
        Format="n2" HorizontalAlignment="Right" />
    <fg:Column Binding="{Binding Ask}"       Width="150"
        Format="n2" HorizontalAlignment="Right" />
    <fg:Column Binding="{Binding LastSale}"  Width="150"
        Format="n2" HorizontalAlignment="Right" />
    <fg:Column Binding="{Binding BidSize}"   Width="100"
        Format="n0" HorizontalAlignment="Right" />
    <fg:Column Binding="{Binding AskSize}"   Width="100"
        Format="n0" HorizontalAlignment="Right" />
    <fg:Column Binding="{Binding LastSize}"  Width="100"
        Format="n0" HorizontalAlignment="Right" />
    <fg:Column Binding="{Binding Volume}"    Width="100"
        Format="n0" HorizontalAlignment="Right" />
    <fg:Column Binding="{Binding QuoteTime}" Width="100"
        Format="hh:mm:ss" HorizontalAlignment="Center" />
    <fg:Column Binding="{Binding TradeTime}" Width="100"
        Format="hh:mm:ss" HorizontalAlignment="Center" />
  </fg:C1FlexGrid.Columns>
</fg:C1FlexGrid>
```

検索とフィルタ処理

このサンプルでは、前述のミュージックライブラリサンプルと同様に SearchBox コントロールを再利用して、簡単な方法で検索を行います。ユーザーは、特定の銘柄を選択する代わりに、検索ボックスに「銀行」、「保険」などを入力してデータをフィルタ処理します。

WPF

```
FinancialDataList list = FinancialData.GetFinancialData();
var view = new MyCollectionView(list);
_flexFinancial.ItemsSource = view;
//「検索」ボックスを設定します。
_srchCompanies.View = view;
var props = _srchCompanies.FilterProperties;
props.Add(typeof(FinancialData).GetProperty("Name"));
props.Add(typeof(FinancialData).GetProperty("Symbol"));
```

カスタムセル

サンプルを実行すると、既にグリッドが機能して、予期したとおりにデータが更新されることがわかります。更新パラメータを変更すると、更新の頻度を調節し、更新中にグリッドをスクロールできるようになります。

ただし、更新はされますが確認は困難です。画面上の至る所で数値が点滅し、その数が多すぎるためです。

そこで、カスタムセルを使用して、点滅とスパークラインによってユーザーエクスペリエンスを高めます。

セルに含まれる値が変化すると、点滅によってセルの背景が一時的に変更されます。株価が上昇するとセルは瞬時に緑に変化し、ゆっくりと白に戻ります。

スパークラインは、各セルに表示されるマイクロチャートです。このマイクロチャートにはセルの直近 5 つの値が表示されるので、ユーザーは即座にトレンド（株価が上昇しているか、下落しているか、または安定しているか）を識別できます。

カスタムセルを使用するには、前のサンプルと同様の手順に従います。**FinancialCellFactory** のクラスの作成から開始し、そのクラスのインスタンスをグリッドの **CellFactory** プロパティに割り当てます。

WPF

```
// カスタムセルファクトリーを使用します。
_flexFinancial.CellFactory = new FinancialCellFactory();
public class FinancialCellFactory : CellFactory
{
  static Thickness _thicknessEmpty = new Thickness(0);
  // セルをティッカーに連結します。
  public override void CreateCellContent(
              C1FlexGrid grid, Border bdr, CellRange range)
  {
    // セルに連結します。
    var r = grid.Rows[range.Row];
    var c = grid.Columns[range.Column];
    var pi = c.PropertyInfo;
    if (r.DataItem is FinancialData &&
      (pi.Name == "LastSale" || pi.Name == "Bid" || pi.Name == "Ask"))
    {
      // ティッカーセルを作成します。
      StockTicker ticker = new StockTicker();
      bdr.Child = ticker;
      bdr.Padding = _thicknessEmpty;
      // スパークラインを表示します。
      ticker.Tag = r.DataItem;
      ticker.BindingSource = pi.Name;
      // 標準の連結方法
      var binding = new Binding(pi.Name);
      binding.Source = r.DataItem;
      binding.Mode = BindingMode.OneWay;
      ticker.SetBinding(StockTicker.ValueProperty, binding);
    }
    else
    {
      // デフォルトの実装方法を使用します。
      base.CreateCellContent(grid, bdr, range);
    }
  }
}
```

このカスタムセルファクトリは、行データが **FinancialData** 型かどうかと、列がデータオブジェクトの **LastSale**、**Bid**、または **Ask** プロパティに連結されているかどうかをチェックします。以上の条件がすべて満たされる場合、セルファクトリは、新しい **StockTicker** 要素を作成してデータに連結します。

StockTicker 要素は、ユーザーにデータを表示するために使用します。これは、次の子要素を含む 4 列の **Grid** 要素で構成されます。

要素の説明	タイプ	名前
現在の値	TextBlock	_txtValue
最新の変化率(%	TextBloc	_txtChange
アップ/ダウンアイコン	Polygon	_arrow

スパークライン	Polyline	_sparkLine

これらの要素は **StockTicker.xaml** ファイルで定義されますが、ここでは、そのリストは示しません。

StockTicker.xaml ファイルで最も注目する部分は、点滅を実装するために使用する **Storyboard** の定義です。**Storyboard** は、**Background** コントロールを現在の値から透明まで徐々に変化させるために使用します。

XAML for WPF

```
<UserControl.Resources>
  <Storyboard x:Key="_sbFlash" >
    <ColorAnimation
      Storyboard.TargetName="_root"
      Storyboard.TargetProperty=
                "(Grid.Background).(SolidColorBrush.Color)"
      To="Transparent"
      Duration="0:0:1"
    />
  </Storyboard>
</UserControl.Resources>
```

StockTicker コントロールの実装は、**StockTicker.cs** ファイルにあります。ここで、注目する部分にはコメントが付けられています。

WPF

```
/// <summary>
/// StockTicker.xaml の相互作用ロジック
/// </summary>
public partial class StockTicker : UserControl
{
  public static readonly DependencyProperty ValueProperty =
    DependencyProperty.Register(
      "Value",
      typeof(double),
      typeof(StockTicker),
      new PropertyMetadata(0.0, ValueChanged));
```

ここでは、コントロールをその基底のデータ値に連結するために使用する **ValueProperty** という **DependencyProperty** の定義から開始します。**Value** プロパティは次のように実装されます。

WPF

```
public double Value
{
  get { return (double)GetValue(ValueProperty); }
  set { SetValue(ValueProperty, value); }
}
private static void ValueChanged(DependencyObject d,
            DependencyPropertyChangedEventArgs e)
{
  var ticker = d as StockTicker;
  var value = (double)e.NewValue;
  var oldValue = (double)e.OldValue;
```

スパークラインを実装するために、コントロールは現在の値と前の値以外にもアクセスする必要があります。これは、コントロールの **Tag** プロパティに **FinancialData** オブジェクトを格納し、**FinancialData.GetHistory** メソッドを呼び出して実行します。

この場合、イベントの **OldValue** プロパティによって提供される前の値は信頼できません。グリッドによってセルが仮想化され、コントロールはセルがビューに表示されるようにスクロールされたときに作成されたので、**StockTicker.Value** プロパティが変更されている可能性があります。その場合、コントロールに前の値はありません。この問題は、**FinancialData** オブジェクトから前の値を取得しても解決できます。

WPF

```
// 履歴を取得します。
  var data = ticker.Tag as FinancialData;
  var list = data.GetHistory(ticker.BindingSource);
  if (list != null && list.Count > 1)
  {
    oldValue = (double)list[list.Count - 2];
  }
```

値が使用できるようになると、コントロールは最新の変化をパーセンテージとして計算し、コントロールのテキストを更新します。

WPF

```
  // 変化率を計算します。
  var change = oldValue == 0 || double.IsNaN(oldValue)
    ? 0
    : (value - oldValue) / oldValue;
  // テキストを更新します。
  ticker._txtValue.Text = value.ToString(ticker._format);
  ticker._txtChange.Text = string.Format("({0:0.0}%)", change * 100);
```

変化率(%)は、アップ/ダウンの記号、テキスト、および点滅の色の更新にも使用されます。変化がない場合、アップ/ダウンの記号は表示されず、テキストはデフォルト色に設定されます。

変化が負の場合は、**ScaleY** を -1 に変更してアップ/ダウンの記号が下を示すようにし、記号、テキスト、点滅のアニメーションの色を赤に設定します。

変化が正の場合は、**ScaleY** を +1 に変更してアップ/ダウンの記号が上を示すようにし、記号、テキスト、点滅のアニメーションの色を緑に設定します。

WPF

```
// 点滅色を更新します。
  var ca = ticker._flash.Children[0] as ColorAnimation;
  // シンボルを更新します。
  if (change == 0)
  {
    ticker._arrow.Fill = null;
    ticker._txtChange.Foreground = ticker._txtValue.Foreground;
  }
  else if (change < 0)
  {
    ticker._stArrow.ScaleY = -1;
    ticker._txtChange.Foreground = ticker._arrow.Fill = _brNegative;
    ca.From = _clrNegative;
  }
  else
  {
    ticker._stArrow.ScaleY = +1;
    ticker._txtChange.Foreground = ticker._arrow.Fill = _brPositive;
    ca.From = _clrPositive;
  }
```

次に、以前の **FinancialData.GetHistory** メソッドの呼び出しによって提供される値履歴配列によってスパークライン多角形の **Points** プロパティに値を設定して、スパークラインを更新します。スパークライン多角形の **Stretch** プロパティが **Fill** に設定されるので、ラインは使用できるスペースに合わせて自動的にスケールします。

WPF

```
  // スパークラインを更新します。
  if (list != null)
  {
    var points = ticker._sparkLine.Points;
    points.Clear();
    for (int x = 0; x < list.Count; x++)
    {
      points.Add(new Point(x, (double)list[x]));
    }
  }
```

最後に、値が実際に変化し、コントロールがまだ作成されていない場合は、**StoryBoard.Begin** メソッドを呼び出すことによってセルを点滅させます。

WPF

```
// 新規値を点滅します (コントロール作成直後に点滅しません)。
  if (!ticker._firstTime && change != 0)
  {
    ticker._flash.Begin();
  }
  ticker._firstTime = false;
}
```

これで **StockTicker** コントロールは完成しました。ここでサンプルアプリケーションを実行し、[カスタムセル]チェックボックスを確認すると、今までよりもはるかに有用な情報が表示されていることがすぐにわかります。セルは値が変化すると点滅し、スパークラインには値のトレンドが迅速に表示されます。