InputPanel for UWP

2018.04.10 更新

グレープシティ株式会社

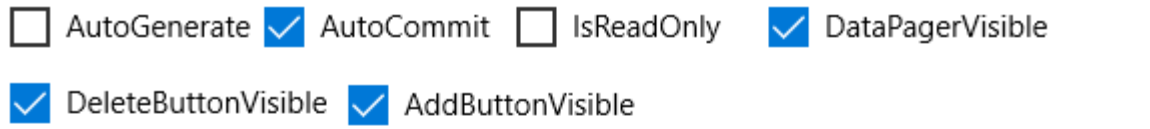
InputPanel for UWP

目次

InputPanel for UWP

ComponentOne には、強力なデータ入力アプリケーションを最小限の作業で作成できるように設計されたコンテナコントロール **InputPanel for UWP** が導入されています。このコントロールは、データレコードの表示、追加、編集、削除、レコード間の移動などを簡単に行うことができるデータフォームを作成および管理するための新しい枠組みです。

InputPanel は、モデルオブジェクト、データテーブル、コレクションなどに簡単に連結することができ、それによって自動的に UI に組み込みエディタが挿入されます。入力コンポーネントを明示的に追加する必要がありません。これらのエディタは基本レベルの検証を行うため、開発の労力をほとんどかけずに、ユーザー入力の不整合を処理することができます。さらに、InputPanel コントロールは、レイアウト、デザイン、外観をカスタマイズでき、容易で一貫性のあるスタイル設定を実現しています。



主な特長

- **フィールドの自動生成**
  InputPanel は、コントロールに提供された入力に基づいて自動的にフィールドを生成します。フィールドが生成されると、それらのデータ型に応じてさまざまなコンポーネントが自動的に InputPanel コントロールの UI に挿入されます。

- **追加、編集、および削除操作**
  InputPanel では、新しいレコードを追加できるほか、データソース内にある現在のレコードを編集および削除することができます。現在表示されているレコードを削除することもできます。

- **データの自動コミット**
  InputPanel では、ユーザーが入力を編集するたびに変更を自動的にコミット/保存できます。この機能をカスタマイズして、確認後に編集がコミットされるようにすることもできます。

- **データ連結**
  InputPanel は、モデルオブジェクトデータ、CollectionView、ObservableCollection などの任意の種類のコレクションデータソースに最小限のコードで連結することができ、完全なナビゲーション機能を持つデータベースブラウザを短時間で作成できます。

- **検証とエラー処理**
  InputPanel では、ユーザー入力に対するデータ検証機能が提供されています。検証エラーは、マークアップまたはイベントのいずれかで定義されたルールに従って表示されます。

- **レコードナビゲーション**
  InputPanel では、レコード間を移動するためのボタンが UI 自体に提供されています。コントロールにページャを実装するための追加コードは必要ありません。

- **データテンプレートのサポート**
  InputPanel では、ユーザー定義のデータテンプレートを作成することで、コントロールのレイアウトを変更することができます。デフォルトでは、フィールドは縦方向に並べられますが、必要に応じて独自のデータテンプレートを作成してレイアウトを定義できます。

- **カスタムフィールドのサポート**
  InputPanel では、コントロール上にフィールドをレンダリングするための 6 種類のデフォルトのエディタが提供されています。ただし、必要に応じて、標準のフィールドを別の入力フィールドに置き換えることもできます。

- **アクセラレータキー**
  InputPanel では、マウスを使用せずに、キーボードを使用して項目を編集したり項目間を移動することができます。InputPanel コントロールにデータソースからフィールドが挿入されると、キーボードアクセラレータキーが自動的に生成されて割り当てられます。

オブジェクトモデルの概要

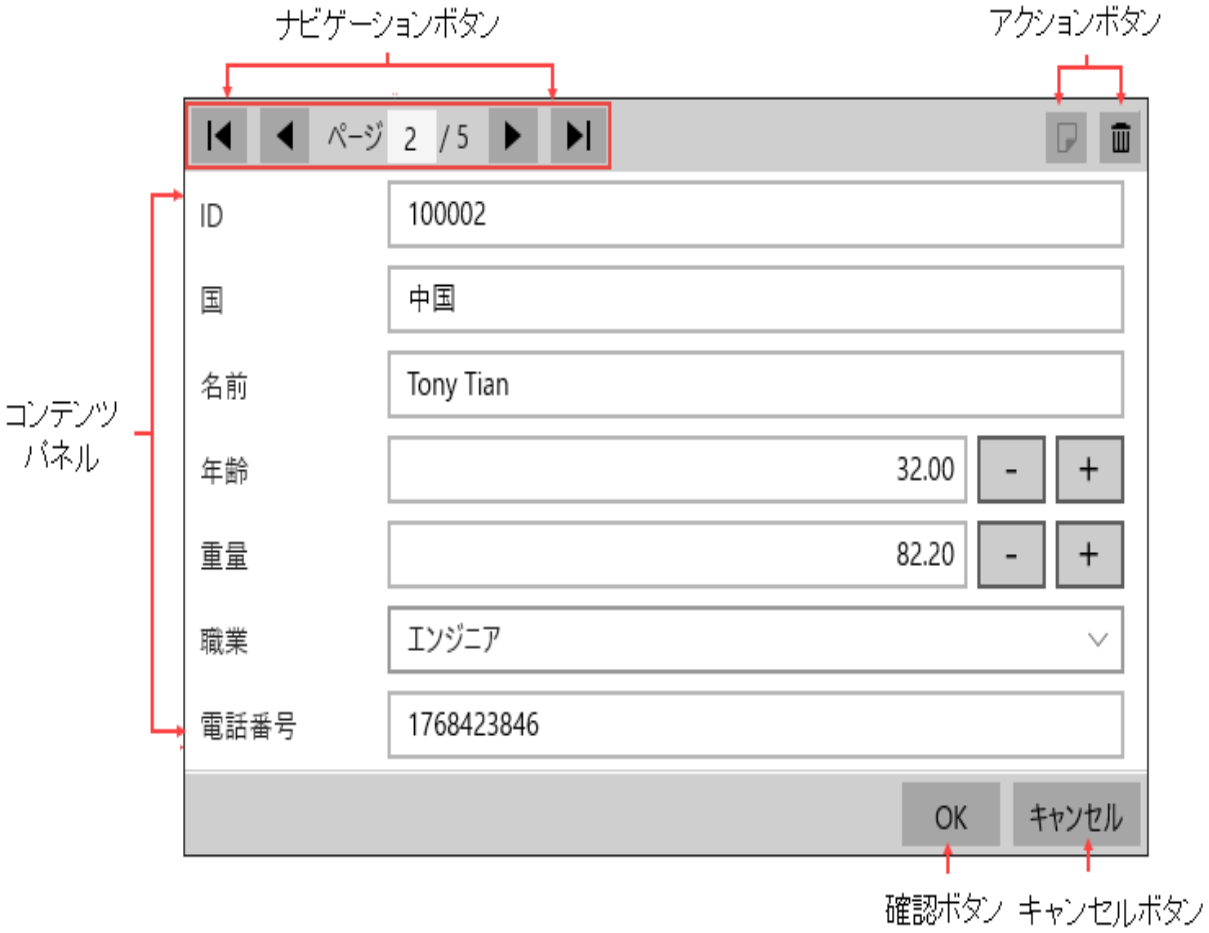
InputPanel には、データ入力アプリケーションを作成するためのさまざまなクラス、オブジェクト、コレクション、関連するメソッドおよびプロパティを提供するリッチオブジェクトモデルが用意されています。これらのオブジェクトの一部とそのプロパティを次の表に一覧します。

C1DataPager
プロパティ: CanMoveItem、CanMoveToFirstItem、CanMoveToLastItem、CanMoveToNextItem、CanMoveToPreviousItem、DisplayMode、NumericButtonStyle、Source
C1InputBase
プロパティ: DataBinding、Header、IsReadOnly、LabelForeground
C1InputCheckBox
プロパティ: IsThreeState
C1InputComboBox
プロパティ: EnumType、ItemsSource
C1InputDateTimePicker
プロパティ: AllowNull、EditMode
C1InputMaskedTextBox
プロパティ: Mask
C1InputNumericBox
プロパティ: AllowNull、Format
C1InputPanel
プロパティ: AddButtonVisibility、AutoCommit、AutoGenerate、CurrentItem、DataPagerVisibility、DeleteButtonVisibility、Header、HeaderBackground、HeaderForeground、ItemsPanelTemplate、ItemsSource、ItemsTemplate、NavigationBackground、ValidationBackground、ValidationErrors
C1InputPanelMaskAttribute
プロパティ: Mask
C1InputPanelPresenter
プロパティ: AutoGenerate、CurrentItem、InputControls、ItemsPanelTemplate、ItemsTemplate
C1InputPanelValidationSummary
プロパティ: Errors、HasErrors

InputPanel の要素

InputPanel コントロールは、ナビゲーションボタン、アクションボタン、コンテンツパネル、確認/キャンセルボタンの 4 種類の要素から成ります。

次の図に、InputPanel コントロールの要素を示します。



次の表で、InputPanel コントロールの要素について説明します。

要素	名前	説明
	ナビゲーションボタン	次へ/前へ/最初/最後ボタンを使用して特定のレコードに移動できます。これらのボタンは、InputPanel コントロールの左上にあります。
	アクションボタン	レコードを追加するための[追加]ボタンと、レコードを削除するための[削除]ボタンがあります。これらのボタンは、InputPanel コントロールの右上にあります。

	コンテンツパネル	フォームレイアウトでフィールドが表示される領域です。ユーザーは、実行時、ここでレコードを表示/編集できます。
	確認/キャンセルボタン	[OK]ボタンと[キャンセル]ボタンがあります。[OK]ボタンを使用すると編集が保存/コミットされ、[キャンセル]ボタンを使用すると編集がキャンセルされます。これらのボタンは、InputPanel コントロールの右下にあります。

InputPanel のエディタ

InputPanel では、さまざまなタイプのデータをサポートするために、以下の 6 種類のエディタが提供されています。AutoGenerate プロパティが true に設定されている場合は、これらのエディタがサポートしているデータ型に応じて、InputPanel コントロールのコンテンツパネルに自動的にエディタがレンダリングされます。

- **InputTextBox**
  InputTextBox エディタは、データ型が文字列または文字に設定されているフィールドに使用されます。

- **InputDateTimePicker**
  InputDateTimePicker エディタは、データ型が日付、時刻、または日時に設定されているフィールドに使用されます。ユーザーは、エディタをクリックすると表示されるカレンダーから日付を選択することで日付を、また[+]ボタンまたは[-]ボタンを使用することで時刻を編集/入力できます。**C1InputDateTimePicker** クラスの **EditMode** プロパティを使用してフィールドの動作を変更することができます。

- **InputCheckBox**
  InputCheckBox エディタは、データ型がブール値に設定されているフィールドに使用されます。このエディタでは、**C1InputCheckBox** クラスの **IsThreeState** プロパティを設定して、オン、オフ、不確定の 3 つのチェックボックス状態をサポートすることができます。

- **InputComboBox**
  InputComboBox エディタは、データ型が列挙に設定されているフィールドに使用されます。このエディタは、ドロップダウンメニューに項目のリストを表示します。このエディタは、**ItemsSource** プロパティを使用してデータソースに連結することができます。

- **InputNumbericBox**
  InputNumbericBox エディタは、データ型が数値に設定されているフィールドに使用されます。コントロールの[+]ボタンと[-]ボタンをクリックすることで、値を増減できます。このエディタでは、**C1InputNumbericBox** クラスの **Format** プロパティを使用して表示書式を設定できます。

- **InputMaskedTextBox**
  InputMaskedTextBox エディタは、各文字位置が特殊なプレースホルダまたはリテラル文字のいずれかにマップされるフィールドに使用されます。リテラル文字（リテラル）は、使用されているデータの型を視覚的に示します。このエディタでは、**C1InputMaskedTextBox** クラスの **Mask** プロパティを使用してマスク付きテキストの書式を設定できます。

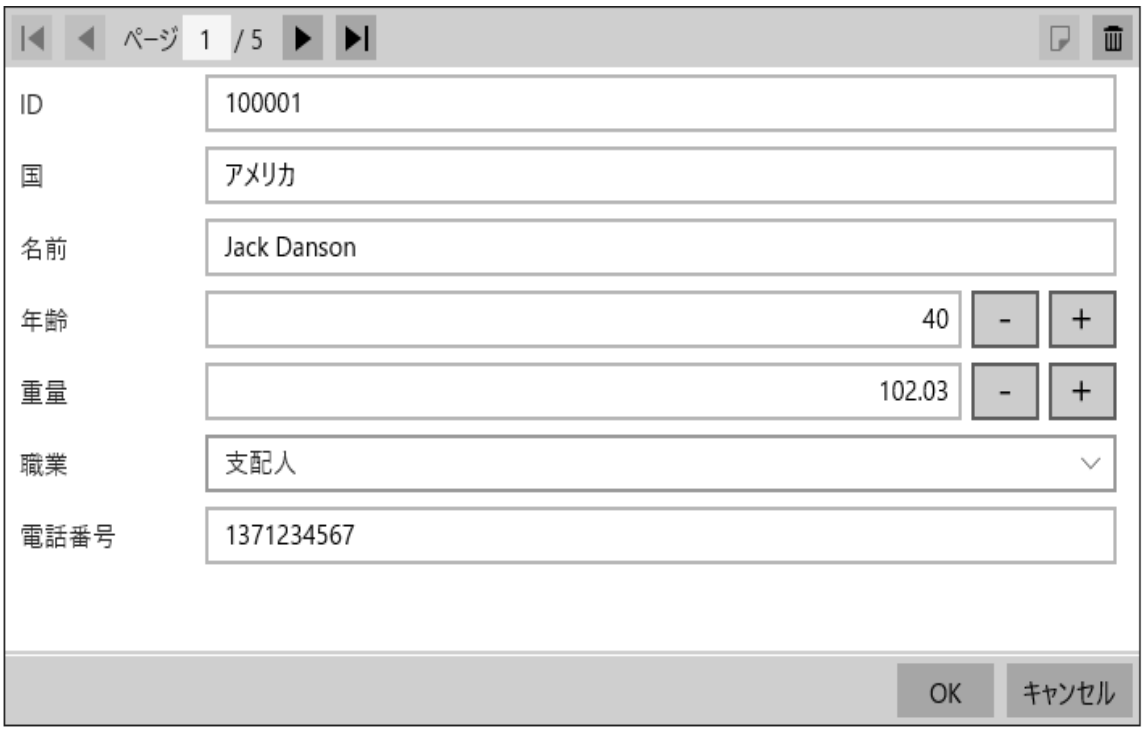
クイックスタート

このクイックスタートガイドでは、リストを使用した InputPanel コントロールへのデータの追加と表示について詳しく説明します。最初に Visual Studio で UWPアプリケーションを作成し、そこに InputPanel コントロールを追加します。項目のリストを作成し、そのリストを InputPanel に連結します。

InputPanel コントロールにデータを追加して表示する単純な UWP 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. **アプリケーションの設定**
2. **InputPanel へのデータの追加と表示**

次の図に、C1InputPanel コントロールに表示されたレコードを示します。



アプリケーションの設定

1. Visual Studio で **Blank App (ユニバーサル Windows)**プロジェクトを作成します。
2. **InputPanel** コントロールを **XAML** デザイナに追加し、コントロールの名前を **'InPanel'**に設定します。
   C1.UWP.InputPanel と共に、次の参照がアプリケーションに自動的に追加されます。
   - C1.UWP
   - C1.UWP.Calendar
   - C1.UWP.DateTimeEditors

先頭に戻る

InputPanel へのデータの追加と表示

1. コードビューに切り替え、データを定義するためにクラス **Customer** を追加します。
2. 次のコードを追加して、クラスに 1 つの列挙といくつかのプロパティを作成します。
   - **Visual Basic**

```
Public Property ID() As String
    Get
        Return m_ID
    End Get
    Set
        m_ID = Value
    End Set
End Property
```

```
Private m_ID As String
Public Property 国() As String
    Get
        Return m_Country
    End Get
    Set
        m_Country = Value
    End Set
End Property
Private m_Country As String
Public Property 名前() As String
    Get
        Return m_Name
    End Get
    Set
        m_Name = Value
    End Set
End Property
Private m_Name As String
Public Property 年齢() As Integer
    Get
        Return m_Age
    End Get
    Set
        m_Age = Value
    End Set
End Property
Private m_Age As Integer
Public Property 重量() As Double
    Get
        Return m_Weight
    End Get
    Set
        m_Weight = Value
    End Set
End Property
Private m_Weight As Double
Public Property 職業() As Occupation
    Get
        Return m_Occupation
    End Get
    Set
        m_Occupation = Value
    End Set
End Property
Private m_Occupation As Occupation
Public Property 電話番号() As String
    Get
        Return m_Phone
    End Get
    Set
        m_Phone = Value
    End Set
End Property
Private m_Phone As String

Public Sub New(id As String, country As String, name As String, age As Integer,
               weight As Double, occupation As Occupation, phone As String)
    Me.ID = id
    Me.国 = country
    Me.名前 = name
    Me.年齢 = age
    Me.重量 = weight
    Me.職業 = occupation
    Me.電話番号 = phone
End Sub
```

```
End Class

Public Enum Occupation
    医師
    芸術家
    教育者
    エンジニア
    支配人
    その他
End Enum
```

- **C#**

```
    public string ID { get; set; }
    public string 国 { get; set; }
    public string 名前 { get; set; }
    public int 年齢 { get; set; }
    public double 重量 { get; set; }
    public Occupation 職業 { get; set; }
    public string 電話番号 { get; set; }

    public Customer(string id, string country, string name, int age, double weight,
                    Occupation occupation, string phone)
    {
        this.ID = id; this.国 = country; this.名前 = name; this.年齢 = age;
        this.重量 = weight; this.職業 = occupation; this.電話番号 = phone;
    }
}

public enum Occupation
{
    医者,
    芸術家,
    教育者,
    エンジニア,
    支配人,
    その他
}
```

3. 次のコードを使用して、Customer のリストを作成し、リストにデータを追加します。
   - **Visual Basic**

```
Dim data As New List(Of Customer)()
data.Add(New Customer("100001", "アメリカ", "Jack Danson", 40, 102.03, Occupation.支配人,
    "1371234567"))
data.Add(New Customer("100002", "中国", "Tony Tian", 32, 82.2, Occupation.エンジニア,
    "1768423846"))
data.Add(New Customer("100003", "イラン", "Larry Frommer", 15, 40.432, Occupation.芸術家,
    "8473637486"))
data.Add(New Customer("100004", "ドイツ", "Charlie Krause", 26, 69.32, Occupation.医師,
    "675245438"))
data.Add(New Customer("100005", "インド", "Mark Ambers", 51, 75.45, Occupation.その他,
    "1673643842"))
```

   - **C#**

```
List<Customer> data = new List<Customer>();
data.Add(new Customer("100001", "アメリカ", "Jack Danson", 40, 102.03,
         Occupation.支配人, "1371234567"));
data.Add(new Customer("100002", "中国", "Tony Tian", 32, 82.2,
         Occupation.エンジニア, "1768423846"));
data.Add(new Customer("100003", "イラン", "Larry Frommer", 15, 40.432,
         Occupation.芸術家, "8473637486"));
data.Add(new Customer("100004", "ドイツ", "Charlie Krause", 26, 69.32,
         Occupation.医者, "675245438"));
data.Add(new Customer("100005", "インド", "Mark Ambers", 51, 75.45,
         Occupation.その他, "1673643842"));
```

4. 次のコードに示すように、**ItemsSource** プロパティを使用してリストを InputPanel に連結します。
   - **Visual Basic**

```
InPanel.ItemsSource = data
```

   - **C#**

```
InPanel.ItemsSource = data;
```

先頭に戻る

データ連結

InputPanel ではデータ連結がサポートされており、わずか 1 行のコードを使用するだけでコントロールにデータを挿入できます。InputPanel コントロールは、モデルオブジェクトデータ、データテーブル、CollectionView、ObservableCollection などの任意の種類のコレクションデータを含むさまざまなデータソースのデータに連結できます。InputPanel をデータソースオブジェクトに連結すると、基底のデータに応じてフィールドが生成されます。

以下のトピックでは、これらの実装について詳細に説明します。

InputPanel と ObservableCollection の連結
: コードで ObservableCollection を使用してデータを連結する方法を説明します。

InputPanel と CollectionView の連結
: コードで CollectionView を使用してデータを連結する方法を説明します。

InputPanel と ObservableCollection の連結

InputPanel では、通常のコレクションと同様に機能する ObservableCollection を使用してコレクション連結を実装できます。InputPanel を ObservableCollection に連結するには、ObservableCollection<T> クラスを使用して、連結ソースとなるコレクションを取得します。次に、**ItemsSource** プロパティがこのコレクションを受け取って InputPanel コントロールに連結します。

ObservableCollection<T> クラスを使用してデータ連結を行うには、次の手順を実行します。

1. **アプリケーションの設定**
2. **InputPanel 用データソースの作成**
3. **ObservableCollection への InputPanel の連結**

アプリケーションの設定

1. UWP アプリケーションを作成します。
2. アプリケーションに InputPanel コントロールを追加し、「InPanel」という名前を付けます。

先頭に戻る

InputPanel 用データソースの作成

1. 新しいクラス **Employee** をアプリケーションに追加します。
2. このクラスにデータジェネレータとフィールドを追加します。
    - **Visual Basic**

```
'' ** フィールド
Private cid As Integer
Private eid As Integer
Private empoccupation As EOccupation
Private firstname As String, lastname As String
Private empfather As String
Private activeemp As Boolean
Private hiredate As DateTime
Private empweight As Double

' ** データジェネレータ
Shared rnd As New Random()
Shared firstNames As String() = "Mao|Ben|Charlie|Fred|Jack|Karl|Larry|Mark".Split("|"c)
Shared lastNames As String() = "Zedong|Cole|Danson|Myers|Paulson|Richards".Split("|"c)
Shared countries As String() = "中国|インド|アメリカ|ブラジル|ロシア|日本".Split("|"c)
```

    - **C#**

```
// ** フィールド
int id, cid;
string first, last;
string father;
EOccupation occupation;
bool active;
DateTime hired;
```

```
double weight;

// ** データジェネレータ
static Random rnd = new Random();
static string[] firstNames = "Mao|Ben|Charlie|Fred|Jack|Karl|Larry|Mark".Split('|');
static string[] lastNames = "Zedong|Cole|Danson|Myers|Paulson|Richards".Split('|');
static string[] countries = "中国|インド|アメリカ|ブラジル|ロシア|日本".Split('|');
```

3. 次のコードを使用してクラスにプロパティを追加します。
    - **Visual Basic**

```
Public Property ID() As Integer
    Get
        Return eid
    End Get
    Set
        If Value <> eid Then
            eid = Value
        End If
    End Set
End Property
Public ReadOnly Property Country() As String
    Get
        Return countries(cid)
    End Get
End Property

Public Property 国ID() As Integer
    Get
        Return cid
    End Get
    Set
        If Value <> cid AndAlso Value > -1 AndAlso Value < countries.Length Then
            cid = Value
        End If
    End Set
End Property

Public Property 職業() As EOccupation
    Get
        Return empoccupation
    End Get
    Set
        If Value <> empoccupation Then
            empoccupation = Value
        End If
    End Set
End Property

Public Property アクティブ() As Boolean
    Get
        Return activeemp
    End Get
    Set
        If Value <> activeemp Then
            activeemp = Value
        End If
    End Set
End Property

Public Property 名() As String
    Get
        Return firstname
    End Get
    Set
        If Value <> firstname Then
            firstname = Value
        End If
    End Set
```

```
End Property

Public Property 姓() As String
    Get
        Return lastname
    End Get
    Set
        If Value <> lastname Then
            lastname = Value
        End If
    End Set
End Property

Public Property 採用() As DateTime
    Get
        Return hiredate
    End Get
    Set
        If Value <> hiredate Then
            hiredate = Value
        End If
    End Set
End Property

Public Property 重量() As Double
    Get
        Return empweight
    End Get
    Set
        If Value <> empweight Then
            empweight = Value
        End If
    End Set
End Property

' 読み取り専用のもの
Public ReadOnly Property 父親名() As String
    Get
        Return empfather
    End Get
End Property

' ** ユーティリティ
Private Shared Function GetString(arr As String()) As String
    Return arr(rnd.[Next](arr.Length))
End Function

' ** 静的値プロバイダ
Public Shared Function GetCountries() As String()
    Return countries
End Function
Public Shared Function GetFirstNames() As String()
    Return firstNames
End Function
Public Shared Function GetLastNames() As String()
    Return lastNames
End Function
```

- **C#**

```
public int ID
{
    get { return id; }
    set
    {
        if (value != id)
        {
            id = value;
        }
```

```
        }
    }
    public string 名前
    {
        get { return string.Format("{0} {1}", 名, 姓); }
    }

    public string 国
    {
        get { return countries[cid]; }
    }


    public int 国ID
    {
        get { return cid; }
        set
        {
            if (value != cid && value > -1 && value < countries.Length)
            {
                cid = value;
            }
        }
    }


    public EOccupation 職業
    {
        get
        {
            return occupation;
        }
        set
        {
            if (value != occupation)
            {
                occupation = value;
            }
        }
    }



    public bool アクティブ
    {
        get { return active; }
        set
        {
            if (value != active)
            {
                active = value;
            }
        }
    }


    public string 名
    {
        get { return first; }
        set
        {
            if (value != first)
            {
                first = value;
            }
        }
    }
```

```
public string 姓
{
    get { return last; }
    set
    {
        if (value != last)
        {
            last = value;
        }
    }
}


public DateTime 採用
{
    get { return hired; }
    set
    {
        if (value != hired)
        {
            hired = value;
        }
    }
}


public double 重量
{
    get { return weight; }
    set
    {
        if (value != weight)
        {
            weight = value;
        }
    }
}

// 読み取り専用のもの
public string 父親名
{
    get { return father; }
}

// ** ユーティリティ
static string GetString(string[] arr)
{
    return arr[rnd.Next(arr.Length)];
}

// ** 静的値プロバイダ
public static string[] GetCountries() { return countries; }
public static string[] GetFirstNames() { return firstNames; }
public static string[] GetLastNames() { return lastNames; }
```

4. Employee クラスのコンストラクタを作成し、次のコードを追加します。
    - **Visual Basic**

```
Private values As Array = [Enum].GetValues(GetType(EOccupation))
Public Sub New(eid As Integer)
    ID = eid
    名 = GetString(firstNames)
    姓 = GetString(lastNames)
    国ID = rnd.[Next]() Mod countries.Length
    職業 = DirectCast(values.GetValue(rnd.[Next] _
                (values.Length - 1)), EOccupation)
    アクティブ = rnd.NextDouble() >= 0.5
```

```
        採用 = DateTime.Today.AddDays(-rnd.[Next](1, 365))
        重量 = 50 + rnd.NextDouble() * 50
        empfather = String.Format("{0} {1}", GetString(firstNames), Last)
    End Sub
```

- **C#**

```
    Array values = Enum.GetValues(typeof(EOccupation));
    public Employee(int id)
    {
        ID = id;
        名 = GetString(firstNames);
        姓 = GetString(lastNames);
        国ID = rnd.Next() % countries.Length;
        職業 = (EOccupation)(values.GetValue(rnd.Next
                    (values.Length - 1)));
        アクティブ = rnd.NextDouble() >= .5;
        採用 = DateTime.Today.AddDays(-rnd.Next(1, 365));
        重量 = 50 + rnd.NextDouble() * 50;
        father = string.Format("{0} {1}", GetString(firstNames), 姓);
    }
```

5. 次のコードを使用して、ObservableCollection<T> クラスのメソッド GetEmployeeList を作成します。

- **Visual Basic**

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

- **C#**

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

先頭に戻る

ObservableCollection への InputPanel の連結

1. 次のコードを追加して、ItemsSource プロパティを使用して InputPanel コントロールをデータと連結します。

- **Visual Basic**

```
InPanel.ItemsSource = Employee.GetEmployeeList(50)
```

- **C#**

```
InPanel.ItemsSource = Employee.GetEmployeeList(50);
```

2. [F5]キーを押してアプリケーションを実行します。

先頭に戻る

InputPanel と CollectionView の連結

InputPanel では、レコード管理、フィルタ処理、グループ化、ソートの各機能を備えたインタフェース ICollectionView を使用して、コレクション連結を実装できます。InputPanel を ObservableCollection に連結するために、ICollectionView インタフェースを実装するオブジェクトに InputPanel を連結できます。以下の例では、ObservableCollection<T> クラスを連結ソースとして使用してコレクションを取得し、ICollectionView インタフェースを実装する **CollectionView** クラスを使用してソースコレク

ションを表示しています。その後、**C1InputPanel** クラスの **ItemsSource** プロパティを使用して、InputPanel コントロールを ICollectionView に連結しています。

ICollectionView を使用してデータ連結を行うには、次の手順を実行します。

1. **アプリケーションの設定**
2. **InputPanel 用データソースの作成**
3. **ICollectionView への InputPanel の連結**

アプリケーションの設定

1. UWP アプリケーションを作成します。
2. アプリケーションに InputPanel コントロールを追加し、「InPanel」という名前を付けます。

先頭に戻る

InputPanel 用データソースの作成

1. 新しいクラス **Product** をアプリケーションに追加します。
2. このクラスに次のフィールドを追加します。
   - **Visual Basic**

```
Shared lines As String() = "コンピュータ|ワッシャ|ストーブ".Split("|"c)
Shared colors As String() = "赤|緑|青|白".Split("|"c)
```

   - **C#**

```
static string[] product = "コンピュータ|ワッシャ|ストーブ".Split('|');
static string[] colors = "赤|緑|青|白".Split('|');
```

3. 次のプロパティとメソッドをクラスに追加します。
   - **Visual Basic**

```
<Display(Name:="製品")>
Public Property Line() As String
    Get
        Return DirectCast(GetValue("Line"), String)
    End Get
    Set
        SetValue("Line", Value)
    End Set
End Property

<Display(Name:="色")>
Public Property Color() As String
    Get
        Return DirectCast(GetValue("Color"), String)
    End Get
    Set
        SetValue("Color", Value)
    End Set
End Property

<Display(Name:="名前")>
Public Property Name() As String
    Get
        Return DirectCast(GetValue("Name"), String)
    End Get
    Set
        SetValue("Name", Value)
    End Set
End Property

<Display(Name:="価格")>
Public Property Price() As Double
    Get
        Return CDbl(GetValue("Price"))
```

```
        End Get
        Set
            SetValue("Price", Value)
        End Set
    End Property

    <Display(Name:="重量")>
    Public Property Weight() As Double
        Get
            Return CDbl(GetValue("Weight"))
        End Get
        Set
            SetValue("Weight", Value)
        End Set
    End Property

    <Display(Name:="原価")>
    Public Property Cost() As Double
        Get
            Return CDbl(GetValue("Cost"))
        End Get
        Set
            SetValue("Cost", Value)
        End Set
    End Property

    <Display(Name:="量")>
    Public Property Volume() As Double
        Get
            Return CDbl(GetValue("Volume"))
        End Get
        Set
            SetValue("Volume", Value)
        End Set
    End Property

    <Display(Name:="廃止")>
    Public Property Discontinued() As Boolean
        Get
            Return CBool(GetValue("Discontinued"))
        End Get
        Set
            SetValue("Discontinued", Value)
        End Set
    End Property

    <Display(Name:="評価")>
    Public Property Rating() As Integer
        Get
            Return CInt(GetValue("Rating"))
        End Get
        Set
            SetValue("Rating", Value)
        End Set
    End Property

    ' 取得/設定の値
    Private values As New Dictionary(Of String, Object)()
    Private Function GetValue(p As String) As Object
        Dim value As Object
        values.TryGetValue(p, value)
        Return value
    End Function
```

```
    Private Sub SetValue(p As String, value As Object)
        If Not Object.Equals(value, GetValue(p)) Then
            values(p) = value
            OnPropertyChanged(p)
        End If
    End Sub
    Protected Overridable Sub OnPropertyChanged(p As String)
        RaiseEvent PropertyChanged(Me, New PropertyChangedEventArgs(p))
    End Sub

    Public Shared Function GetLines() As String()
        Return lines
    End Function


#Region "INotifyPropertyChanged Members"

    Public Event PropertyChanged As PropertyChangedEventHandler _
        Implements INotifyPropertyChanged.PropertyChanged
```

- **C#**

```
[Display(Name = "製品")]
public string Line
{
    get { return (string)GetValue("product"); }
    set { SetValue("product", value); }
}

[Display(Name = "色")]
public string Color
{
    get { return (string)GetValue("Color"); }
    set { SetValue("Color", value); }
}

[Display(Name = "名前")]
public string Name
{
    get { return (string)GetValue("Name"); }
    set { SetValue("Name", value); }
}

[Display(Name = "価格")]
public double Price
{
    get { return (double)GetValue("Price"); }
    set { SetValue("Price", value); }
}

[Display(Name = "重量")]
public double Weight
{
    get { return (double)GetValue("Weight"); }
    set { SetValue("Weight", value); }
}

[Display(Name = "原価")]
public double Cost
{
    get { return (double)GetValue("Cost"); }
    set { SetValue("Cost", value); }
}

[Display(Name = "量")]
```

```
public double Volume
{
    get { return (double)GetValue("Volume"); }
    set { SetValue("Volume", value); }
}

[Display(Name = "廃止")]
public bool Discontinued
{
    get { return (bool)GetValue("Discontinued"); }
    set { SetValue("Discontinued", value); }
}

[Display(Name = "評価")]
public int Rating
{
    get { return (int)GetValue("Rating"); }
    set { SetValue("Rating", value); }
}

// 取得/設定の値
Dictionary<string, object> values = new Dictionary<string, object>();
object GetValue(string p)
{
    object value;
    values.TryGetValue(p, out value);
    return value;
}
void SetValue(string p, object value)
{
    if (!object.Equals(value, GetValue(p)))
    {
        values[p] = value;
        OnPropertyChanged(p);
    }
}
protected virtual void OnPropertyChanged(string p)
{
    if (PropertyChanged != null)
        PropertyChanged(this, new PropertyChangedEventArgs(p));
}

public static string[] GetLines()
{
    return product;
}


#region INotifyPropertyChanged Members

public event PropertyChangedEventHandler PropertyChanged;
```

4. 次のコードを使用して、IEnumerable インタフェースのメソッド GetProducts を作成します。
    - **Visual Basic**

```
Public Shared Function GetProducts(count As Integer) As IEnumerable
    Dim list = New ObservableCollection(Of Products)()

    Dim rnd = New Random(0)
    For i As Integer = 0 To count - 1
        Dim p = New Products()
        p.Line = lines(rnd.[Next]() Mod lines.Length)
        p.Color = colors(rnd.[Next]() Mod colors.Length)
        p.Name = String.Format("{0} {1}{2}",
                 p.Line.Substring(0, p.Line.Length - 1),
```

```
                p.Line(0), i)
        p.Price = (rnd.[Next](1, 1000) + rnd.[Next](1, 1000) _
                  + rnd.[Next](1, 1000)) / 3
        p.Weight = (rnd.[Next](1, 100) + rnd.[Next](1, 100) _
                   + rnd.[Next](1, 300)) / 5
        p.Cost = rnd.[Next](1, 600)
        p.Volume = rnd.[Next](500, 5000)
        p.Discontinued = rnd.NextDouble() < 0.1
        p.Rating = rnd.[Next](0, 5)
        list.Add(p)
    Next
    Return list
End Function
```

- **C#**

```
public static IEnumerable GetProducts(int count)
{
    var list = new ObservableCollection<Products>();

    var rnd = new Random(0);
    for (int i = 0; i < count; i++)
    {
        var p = new Products();
        p.Line = product[rnd.Next() % product.Length];
        p.Color = colors[rnd.Next() % colors.Length];
        p.Name = string.Format("{0} {1}{2}",
                 p.Line.Substring(0, p.Line.Length - 1),
                 p.Line[0], i);
        p.Price = (rnd.Next(1, 1000) + rnd.Next(1, 1000) +
                  rnd.Next(1, 1000)) / 3;
        p.Weight = (rnd.Next(1, 100) + rnd.Next(1, 100) +
                   rnd.Next(1, 300)) / 5;
        p.Cost = rnd.Next(1, 600);
        p.Volume = rnd.Next(500, 5000);
        p.Discontinued = rnd.NextDouble() < .1;
        p.Rating = rnd.Next(0, 5);
        list.Add(p);
    }
    return list;
}
```

5. 次のコードを追加して、ICollectionView インタフェースのプロパティ CustomerCollectionView を作成します。これは、C1CollectionView クラスを使用してソースコレクションを表示します。

- **Visual Basic**

```
Private Shared view As ICollectionView
Public Shared ReadOnly Property CustomerCollectionView() As ICollectionView
    Get
        If view Is Nothing Then
            Dim products__1 = Products.GetProducts(50)
            view = New C1CollectionView(products__1)
        End If
        Return view
    End Get
End Property
```

- **C#**

```
private static ICollectionView view;
public static ICollectionView CustomerCollectionView
{
    get
    {
        if (view == null)
        {
            var products = Products.GetProducts(50);
            view = new C1CollectionView(products);
```

```
            }
            return view;
        }
    }
```

先頭に戻る

ICollectionView への InputPanel の連結

1. 次のコードを追加して、ItemsSource プロパティを使用して InputPanel コントロールをデータと連結します。
    - **Visual Basic**

```
InPanel.ItemsSource = CustomerCollectionView
```

    - **C#**

```
InPanel.ItemsSource = CustomerCollectionView;
```

2. [F5]キーを押してアプリケーションを実行します。

先頭に戻る

機能

機能セクションでは、InputPanel コントロールが備えるすべての機能について説明します。

レコードの追加、編集、削除
　レコードを追加、編集、および削除する方法を説明します。
レコードナビゲーション
　レコードナビゲーション機能で提供される表示モードについて説明します。
フィールドの自動生成
　コードでフィールドを自動的に生成する方法を説明します。
データの自動コミット
　コードを使用してデータを自動的に保存する方法を説明します。
データ検証
　>コードでデータを検証する方法を説明します。
カスタムテンプレート
　カスタムテンプレートの作成方法を説明します。
キーボードナビゲーション
　移動や編集に使用されるキーボードキーについて説明します。

レコードの追加、編集、削除

InputPanel コントロールでは、コントロールの右上に[追加]ボタンと[削除]ボタンが用意されています。[追加]ボタンを使用すると、レコードのコレクションに新しいレコードを挿入でき、[削除]ボタンを使用すると、現在表示されているレコードを 1 回のクリックで削除できます。次の図に、InputPanel コントロールの[追加]ボタンと[削除]ボタンを示します。

InputPanel でレコードを編集することもできますが、編集専用のボタンはありません。表示されたレコード内のフィールドを選択するだけで、その内容を編集できます。

InputPanel では、**C1InputPanel** クラスの **AddButtonVisibility** プロパティと **DeleteButtonVisibility** プロパティを使用して、それぞれ[追加]ボタンと[削除]ボタンの表示/非表示を設定できます。デフォルトでは、これらのボタンの表示/非表示は **true** に設定されていますが、次のコードを使用してボタンを非表示にすることができます。

Visual Basic

```
InPanel.AddButtonVisibility = Visibility.Collapsed
InPanel.DeleteButtonVisibility = Visibility.Collapsed
```

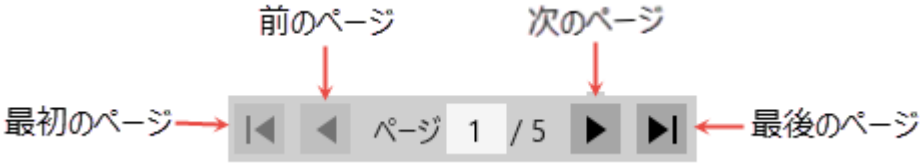
C#

```
InPanel.AddButtonVisibility = Visibility.Collapsed;
InPanel.DeleteButtonVisibility = Visibility.Collapsed;
```

レコードナビゲーション

InputPanel では、レコード間を簡単に移動できるレコードナビゲーションが提供されています。そのためのコードを実装する必要はありません。InputPanel のナビゲーションを使用すると、コレクションの最初、前、次、および最後のレコードに移動するこ

とができます。特定のレコードの番号を入力することで、そのレコードにジャンプすることもできます。次の図に、InputPanel コントロールのナビゲーションボタンを示します。



次の表で、InputPanel コントロールで提供されているナビゲーションボタンとオプションについて説明します。

名前	説明
最初	最初のレコードに移動します。
前へ	前のレコードに移動します。
現在のレコード	現在のレコードの番号を示します。
合計レコード数	表示できるレコードの総数を示します。
次へ	次のレコードに移動します。
最後	最後のレコードに移動します。

InputPanel では、**C1InputPanel** クラスの **DataPagerVisibility** プロパティを使用して、ナビゲーションパネルの表示/非表示を設定できます。デフォルトでは、パネルの表示/非表示は **Visible** に設定されていますが、次のコードを使用してパネルを非表示にすることができます。

Visual Basic

```
InPanel.DataPagerVisibility = Visibility.Collapsed
```

C#

```
InPanel.DataPagerVisibility = Visibility.Collapsed;
```

フィールドの自動生成

InputPanel のフィールドは、個々のフィールドのデータ型に基づいて自動的に生成されます。InputPanel でのフィールドの自動生成は、**C1InputPanel** クラスの **AutoGenerate** プロパティに基づいてサポートされます。このプロパティは、デフォルトで **true** に設定されています。ただし、AutoGenerate プロパティの値を **false** に設定し、フィールドを自分で作成することもできます。

AutoGenerate プロパティの値は、XAML およびコードビューで設定できます。

XAML の場合

AutoGenerate プロパティの値を XAML ビューで設定するには、次のコードを使用します。

XAML

```
<InputPanel:C1InputPanel x:Name="InPanel" AutoGenerate="False"/>
```

コードの場合

AutoGenerate プロパティの値をコードビューで設定するには、次のコードを使用します。

Visual Basic

```
InPanel.AutoGenerate = False
```

C#

```
InPanel.AutoGenerate = false;
```

データの自動コミット

InputPanel では、[OK]ボタンをクリックしなくても、データを自動的に保存できます。InputPanel の自動コミット機能は、**C1InputPanel** クラスの **AutoCommit** プロパティに基づいてサポートされます。このプロパティは、デフォルトで **true** に設定され、入力が行われるたびにデータを自動的に保存します。ただし、AutoCommit プロパティを **false** に設定して、確認後または[OK]ボタンがクリックされた時点で編集が保存されるようにすることもできます。

AutoCommit プロパティの値は、XAML およびコードビューで設定できます。

XAML の場合

AutoCommit プロパティの値を XAML ビューで設定するには、次のコードを使用します。

XAML

```
<InputPanel:C1InputPanel x:Name="InPanel" AutoCommit="False"/>
```

コードの場合

AutoCommit プロパティの値をコードビューで設定するには、次のコードを使用します。

Visual Basic

```
InPanel.AutoCommit = False
```

C#

```
InPanel.AutoCommit = false;
```

データ検証

InputPanel は、無効なユーザー入力をユーザーがチェックできるデータ検証機能をサポートします。複雑なシナリオの検証を

適用するために、標準属性マークアップおよびカスタム属性マークアップを使用したプロパティレベルの検証もサポートしています。さまざまなシナリオでデータ検証を行うためのマークアップをコードで適用できます。たとえば、テキスト内の不要なスペース、null 値、特殊文字をチェックしたり、数値で入力される年齢に制限を設けることができます。さらに、InputPanel では、検証規則を適用するイベントも提供されています。

次に、InputPanel でユーザー入力を検証する方法を示します。

プロパティによる検証
: プロパティレベルの検証をコードで実装する方法を説明します。

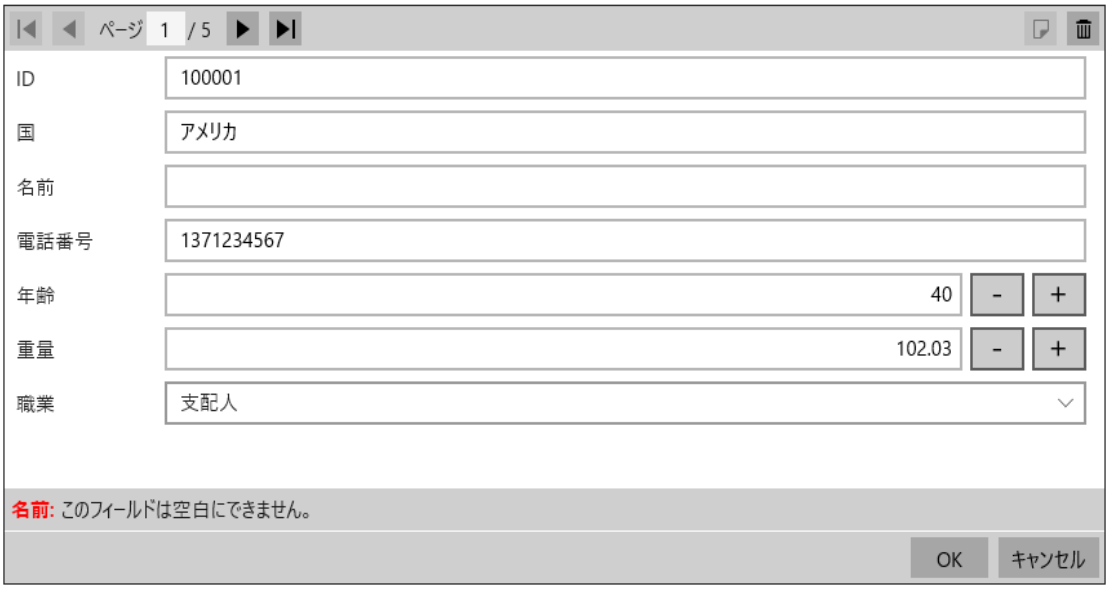
イベントによるデータ検証
: さまざまな変換をコードで適用する方法を説明します。

プロパティによる検証

InputPanel では、組み込みの検証機能が失敗したり不十分である場合にユーザー入力を検証するためのプロパティレベルの検証機能が提供されています。このタイプの検証では、データ検証規則をプロパティセッターコード内で指定します。InputPanel は、プロパティレベルの検証を実装するために 2 種類のマークアップをサポートしています。

- **標準属性マークアップ**
- **カスタムマークアップ**

次の図に、無効な入力に対するプロパティレベルの検証の例を示します。



標準属性マークアップ

アプリケーションのプロパティセッターコード内に標準属性マークアップを追加することで、ユーザー入力に対してプロパティレベルの検証を適用できます。コントロールは、**System.ComponentModel.DataAnnotations** アセンブリ内のいくつかのクラスを直接使用して、このマークアップにアクセスします。

次のコードは、標準属性マークアップをプロパティセッターコード内に追加して、検証を適用する例を示します。この例では、「クイックスタート」で作成したサンプルを使用します。

- **Visual Basic**

```
<Display(Name:="名前")>
<Required(ErrorMessage:="このフィールドは空白にできません。")>
Public Property 名前() As String
    Get
        Return m_Name
    End Get
    Set
        m_Name = Value
    End Set
```

```
End Property
Private m_Name As String

<Display(Name:="電話番号")>
<Required(ErrorMessage:="このフィールドは空白にできません。")>
Public Property 電話番号() As String
    Get
        Return m_Phone
    End Get
    Set
        m_Phone = Value
    End Set
End Property
Private m_Phone As String
```

- **C#**

```
[Display(Name = "名前")]
[Required(ErrorMessage = "このフィールドは空白にできません。")]
public string 名前 { get; set; }

[Display(Name = "電話番号")]
[Required(ErrorMessage = "このフィールドは空白にできません。")]
public string 電話番号 { get; set; }
```

カスタム属性マークアップ

InputPanel では、プロパティレベルの検証を行うためにカスタムマークアップもサポートしています。カスタムマークアップは、ビジネスのニーズに応じて検証規則をカスタマイズする場合に便利です。また、カスタムマークアップを使用すると、入力フィールドに対して複数の検証規則を組み合わせることができます。たとえば、カスタムマークアップを使用して、**電話番号** フィールド内の null 値またはホワイトスペースの有無およびその最小長さと最大長さを 1 つの検証規則で検証することができます。

以下の手順は、プロパティレベルの検証のためのカスタムマークアップをコードで作成および適用する例を示します。この例では、「クイックスタート」で作成したサンプルを使用します。

1. クラス **CustomValidator**を作成し、null 値またはホワイトスペースの有無のチェックおよび **電話番号** フィールドの最小長さと最大長さのチェックを行う検証規則を定義します。
    - **Visual Basic**

```
Public Class CustomValidator
    Public Shared Function ValidatePhoneNumber(電話番号 As String) _
        As ValidationResult
        If String.IsNullOrWhiteSpace(電話番号) Then
            Return New ValidationResult("このフィールドは空白にできません。",
                                        New List(Of String)() From {"電話番号"})
        ElseIf 電話番号.Length > 12 OrElse 電話番号.Length < 9 Then
            Return New ValidationResult("電話番号は8桁でなければなりません" +
                 " および 12桁未満でなければなりません。", New List(Of String)() From {"電話番号"})
        Else

            Return ValidationResult.Success
        End If
    End Function
End Class
```

    - **C#**

```
public class CustomValidator
{
    public static ValidationResult ValidatePhoneNumber(string 電話番号)
    {
        if (string.IsNullOrWhiteSpace(電話番号))
        {
            return new ValidationResult("このフィールドは空白にできません。",
                                        new List<string>() { "電話番号" });
        }
        else if (電話番号.Length > 12 || 電話番号.Length < 9)
        {
            return new ValidationResult("電話番号は8桁でなければなりません"
            + " および 12桁未満でなければなりません。", new List<string>() { "電話番号" });
        }
```

```
            else
            {
                return ValidationResult.Success;
            }
        }
    }
```

2. プロパティセッターコードにこのカスタムマークアップを追加して、電話番号 フィールドで行われた入力を検証します。
    - **Visual Basic**

```
<Display(Name:="電話番号")>
<CustomValidation(GetType(CustomValidator), "ValidatePhoneNumber")>
Public Property 電話番号() As String
    Get
        Return m_Phone
    End Get
    Set
        m_Phone = Value
    End Set
End Property
Private m_Phone As String
```

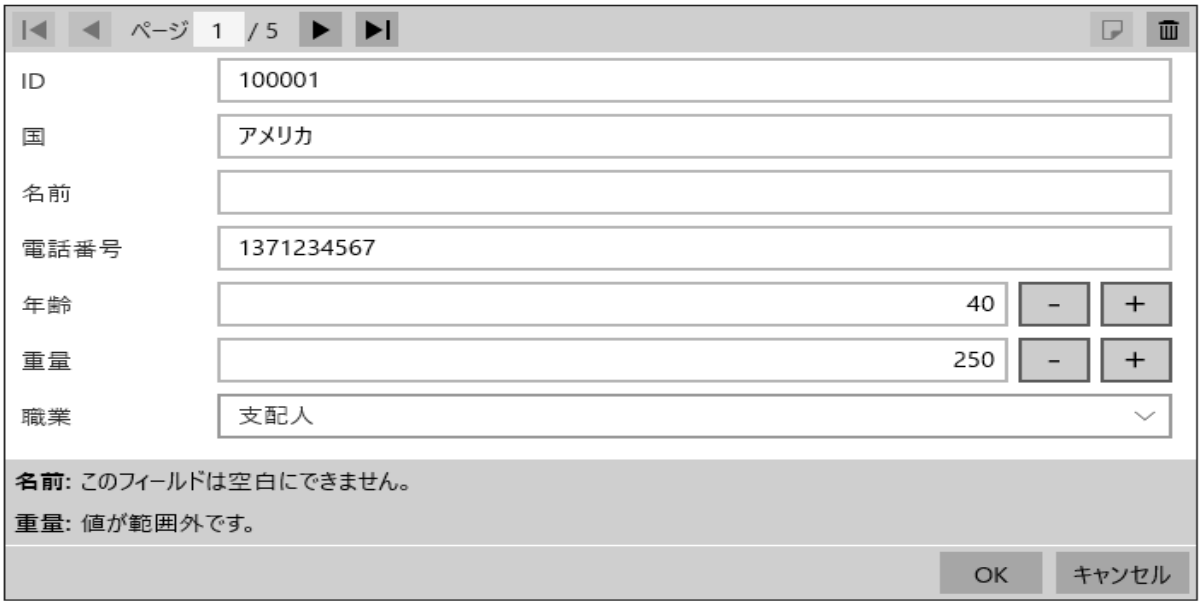
    - **C#**

```
[Display(Name = "電話番号")]
[CustomValidation(typeof(CustomValidator), "ValidatePhoneNumber")]
public string 電話番号 { get; set; }
```

イベントによるデータ検証

InputPanel は、入力検証を行うもう 1 つの方法として、イベントによる方法も提供しています。コントロールには、ユーザー入力の検証に使用できる ValidateCurrentItem イベントが用意されています。

次の図に、イベントを使用して適用された検証を示します。



イベントを使用したデータ検証を実装するには、コードでValidateCurrentItem イベントをサブスクライブし、イベントハンドラに検証規則を追加します。次のコードは、イベントを使用して検証を適用する方法を示します。この例では、「 クイックスタート」で作成したサンプルを使用します。

- **Visual Basic**

```
Dim customer As Customer = TryCast(InPanel.CurrentItem, Customer)

If customer IsNot Nothing Then
    Dim errorList = New ObservableCollection(Of ErrorInfo)()
```

```
    If customer.名前 IsNot Nothing AndAlso String.IsNullOrWhiteSpace _
        (customer.名前.ToString()) Then
        errorList.Add(New ErrorInfo() With {
   .ErrorInputName = "名前",
    .ErrorContent = "このフィールドは空白にできません。"
})
    End If
    If customer.重量 > 150 Then
        errorList.Add(New ErrorInfo() With {
    .ErrorInputName = "重量",
    .ErrorContent = "値が範囲外です。"
})
    End If
    InPanel.ValidationErrors = errorList
    If errorList.Count > 0 Then
        e.Cancel = True
    End If
End If
```

- **C#**

```
Customer customer = InPanel.CurrentItem as Customer;

if (customer != null)
{
    var errorList = new ObservableCollection<ErrorInfo>();

    if (customer.名前 != null && string.IsNullOrWhiteSpace(
        customer.名前.ToString()))
    {
        errorList.Add(new ErrorInfo
        {
            ErrorInputName = "名前",
            ErrorContent = "このフィールドは空白にできません。"
        });
    }
    if (customer.重量 > 150)
    {
        errorList.Add(new ErrorInfo
        {
            ErrorInputName = "重量",
            ErrorContent = "値が範囲外です。"
        });
    }
    InPanel.ValidationErrors = errorList;
    if (errorList.Count > 0)
    {
        e.Cancel = true;
    }
}
```

カスタムテンプレート

InputPanel では、柔軟にカスタムテンプレートを作成して、ユーザーの要件に応じてコントロールのレイアウトを変更できます。カスタムテンプレートを作成することで、アプリケーションに合うコンパクトで見栄えのよいフォームを開発することができます。カスタムレイアウトを定義し、コントロールを表示する方法を変更するには、InputPanel の **DataTemplate** を使用します。

次の図に、InputPanel コントロールにカスタムテンプレートを適用したところを示します。

データテンプレートを使用してカスタムテンプレートを作成し、コントロールのレイアウトを変更するには、以下の手順に従います。この例では、「クイックスタート」で作成したサンプルを使用します。この例で作成するデータテンプレートでは、1 つのスタックパネルに 2 つの横方向のスタックパネルを含めます。内側の 2 つのスタックパネルには横方向に配置するエディタを入れ、外側のスタックパネルには縦方向に配置するエディタを入れます。InputPanel は、**ItemsSource** プロパティを通してこのデータテンプレートにアクセスします。

この例では、**C1InputPanel** クラスの **HeaderTemplate** プロパティを使用して InputPanel のヘッダーテンプレートをカスタマイズする方法も紹介しています。

デフォルトでは、InputPanel のすべてのエディタが縦に並べられます。エディタの配置を変更するには、**AutoGenerate** プロパティを **false** に設定します。

1. 次のコードを <Page> タグ内に追加します。

XAML

```
xmlns:c1="using:C1.Xaml"
xmlns:c1input="using:C1.Xaml.InputPanel"
```

2. 次のコードを <Page> タグの後に追加して、縦と横に積み重ねられたいくつかのエディタでDataTemplateを適用します。

XAML copyCode

```
<Page.Resources>
    <DataTemplate x:Key="InputPanelCustomDataTemplate">
        <StackPanel Background="AliceBlue">
            <StackPanel Orientation="Horizontal">
                <c1input:C1InputTextBox Header="ID" DataBinding="{Binding ID,
Mode=OneWay}" IsReadOnly="True" LabelForeground="{Binding LabelForeground,
ElementName=InPanel}"></c1input:C1InputTextBox>
                <c1input:C1InputTextBox Header="国" DataBinding="{Binding 国,
Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" IsReadOnly="{Binding
IsReadOnly, ElementName=InPanel}" LabelForeground="{Binding LabelForeground,
ElementName=InPanel}"></c1input:C1InputTextBox>
            </StackPanel>
            <c1input:C1InputTextBox Header="名前" DataBinding="{Binding 名前,
Mode=OneWay}" IsReadOnly="True" LabelForeground="{Binding LabelForeground,
ElementName=InPanel}"></c1input:C1InputTextBox>
```

```
            <c1input:C1InputNumericBox Header="電話番号" DataBinding="{Binding 電話番号, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" IsReadOnly="{Binding IsReadOnly, ElementName=InPanel}" LabelForeground="{Binding LabelForeground, ElementName=InPanel}"></c1input:C1InputNumericBox>
            <c1input:C1InputTextBox Header="職業" DataBinding="{Binding 職業, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" IsReadOnly="{Binding IsReadOnly, ElementName=InPanel}" LabelForeground="{Binding LabelForeground, ElementName=InPanel}"></c1input:C1InputTextBox>
            <StackPanel Orientation="Horizontal">
                <c1input:C1InputNumericBox Header="重量" DataBinding="{Binding 重量, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" IsReadOnly="{Binding IsReadOnly, ElementName=InPanel}" LabelForeground="{Binding LabelForeground, ElementName=InPanel}"></c1input:C1InputNumericBox>
                <c1input:C1InputTextBox Header="年齢" DataBinding="{Binding 年齢, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" IsReadOnly="{Binding IsReadOnly, ElementName=InPanel}" LabelForeground="{Binding LabelForeground, ElementName=InPanel}"></c1input:C1InputTextBox>
            </StackPanel>
        </StackPanel>
    </DataTemplate>

    <ItemsPanelTemplate x:Key="ItemsPanel">
        <StackPanel Orientation="Vertical"  Margin="20"/>
    </ItemsPanelTemplate>
</Page.Resources>
```

3. 次のコードを <Grid></Grid> タグ内に追加して、InputPanel とそのヘッダーテンプレートをカスタマイズします。

XAML copyCode

```
<c1input:C1InputPanel x:Name="InPanel" AutoGenerate="False" ItemsPanelTemplate="{StaticResource ItemsPanel}" ItemsTemplate="{StaticResource InputPanelCustomDataTemplate}" HeaderBackground="LightCyan" HeaderFontWeight="Bold" Margin="20,40,58,80">
    <c1input:C1InputPanel.HeaderTemplate>
        <DataTemplate>
            <StackPanel>
                <TextBox Text="カスタムデータテンプレート" Margin="6" Background="LightCyan" Foreground="Blue" />
            </StackPanel>
        </DataTemplate>
    </c1input:C1InputPanel.HeaderTemplate>
</c1input:C1InputPanel>
```

キーボードナビゲーション

InputPanel では、レコード間の移動にキーボードを使用できます。マウスの代わりにキーボードナビゲーションキーを使用して、レコードを編集したりレコード間を移動することができます。使用できるキーと対応するアクションを次の表に一覧します。

キー	アクション
左矢印キー (←)	前の項目に移動します。
右矢印キー (→)	次の項目に移動します。

Ctrl + Home / Ctrl + Left キー	現在の項目を最初の位置に移動します。
Ctrl + End / Ctrl + Right キー	現在の項目を最後の位置に移動します。
Insert	新しいレコードを追加します。
Enter	編集操作をコミットします。
Esc	編集操作をキャンセルします。
Del	現在のレコードを削除します。

InputPanel の操作

「InputPanel の操作」セクションは、ユーザーの皆様が InputPanel コントロールの基礎と機能および一般的な使用方法を理解していることを前提としています。次のセクションでは、InputPanel で提供されている補助機能について説明します。

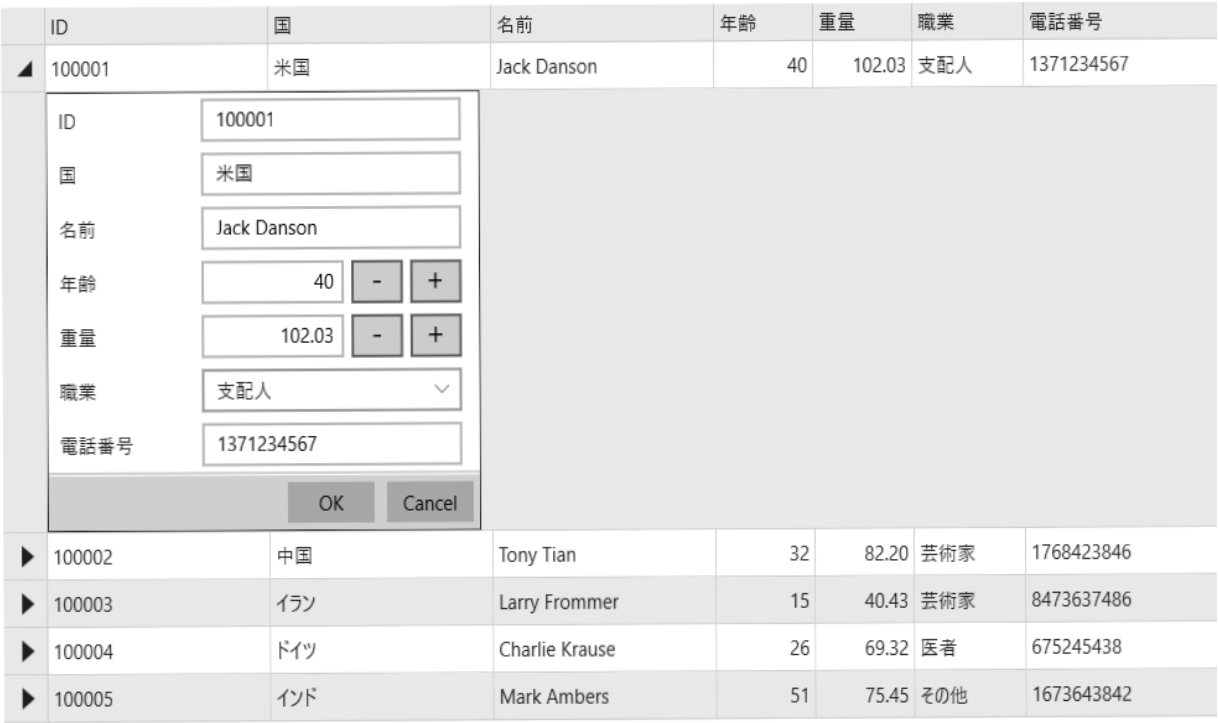
グリッドとの統合
: コードで InputPanel をさまざまなタイプのデータグリッドと統合する方法について説明します。

グリッドとの統合

InputPanel は、行の詳細部分にコンパクトなレイアウトでデータを表示できる FlexGrid との統合をサポートしています。このコントロールを ComponentOne の FlexGrid コントロールに埋め込んで、RowDetailsTemplate を使用してデータを表示できます。このテンプレートを使用して、FlexGrid コントロールの各行の折りたたみ可能なセクション内に InputPanel を埋め込むことができます。

InputPanel を FlexGrid と統合するには、テンプレートを定義する必要があります。このテンプレートで、FlexGrid コントロールの RowDetailsTemplate プロパティに InputPanel コントロールを追加します。

次の図に、RowDetailsTemplate を使用して FlexGrid と統合した InputPanel を示します。



InputPanel を FlexGrid と統合するには

- **手順 1：アプリケーションの設定**
- **手順 2：データソースの作成**
- **手順 3：データソースへの FlexGrid の連結**

先頭に戻る

手順 1：アプリケーションの設定

1. UWP アプリケーションを作成し、FlexGrid コントロールを追加します。
2. FlexGrid の RowDetailsTemplate プロパティで、行の詳細部分に InputPanel コントロールを表示するためのデータテンプレートを作成します。

```
XAML                                                                    copyCode
<FlexGrid:C1FlexGrid x:Name="FlexGrid" Margin="20,50,0,10"
HorizontalAlignment="Left"
  AlternatingRowBackground="{Binding GroupRowBackground, ElementName=FlexGrid}" >
    <FlexGrid:C1FlexGrid.RowDetailsTemplate>
        <DataTemplate>
            <InputPanel:C1InputPanel x:Name="InPanel" CurrentItem="{Binding}"
              HorizontalAlignment="Left"/>
        </DataTemplate>
    </FlexGrid:C1FlexGrid.RowDetailsTemplate>
</FlexGrid:C1FlexGrid>
```

先頭に戻る

手順 2:データソースの作成

1. コードビューに切り替え、InputPanel に顧客の ID、名前、国、年齢、重量、電話番号などのレコードを追加するための Customer クラス、および 職業フィールドの値を受け取るための列挙を作成します。
    - **Visual Basic**

```
Public Class Customer
    Public Property ID() As String
        Get
            Return m_ID
        End Get
        Set
            m_ID = Value
        End Set
    End Property
    Private m_ID As String
    Public Property 国() As String
        Get
            Return m_Country
        End Get
        Set
            m_Country = Value
        End Set
    End Property
    Private m_Country As String
    Public Property 名前() As String
        Get
            Return m_Name
        End Get
        Set
            m_Name = Value
        End Set
    End Property
    Private m_Name As String
    Public Property 年齢() As Integer
        Get
            Return m_Age
        End Get
        Set
            m_Age = Value
        End Set
    End Property
    Private m_Age As Integer
    Public Property 重量() As Double
        Get
            Return m_Weight
        End Get
```

```
        Set
            m_Weight = Value
        End Set
    End Property
    Private m_Weight As Double
    Public Property 職業() As Occupation
        Get
            Return m_Occupation
        End Get
        Set
            m_Occupation = Value
        End Set
    End Property
    Private m_Occupation As Occupation
    Public Property 電話番号() As String
        Get
            Return m_Phone
        End Get
        Set
            m_Phone = Value
        End Set
    End Property
    Private m_Phone As String

    Public Sub New(id As String, country As String, name As String, age As Integer,
    weight As Double, occupation As Occupation,
    phone As String)
        Me.ID = id
        Me.国 = country
        Me.名前 = name
        Me.年齢 = age
        Me.重量 = weight
        Me.職業 = occupation
        Me.電話番号 = phone
    End Sub
End Class

Public Enum Occupation
     医師
     芸術家
     教育者
     エンジニア
     支配人
     その他
End Enum
```

- **C#**

```
public class Customer
{
    public string ID { get; set; }
    public string 国 { get; set; }
    public string 名前 { get; set; }
    public int 年齢 { get; set; }
    public double 重量 { get; set; }
    public Occupation 職業 { get; set; }
    public string 電話番号 { get; set; }

    public Customer(string id, string country, string name, int age,
                    double weight, Occupation occupation, string phone)
    {
        this.ID = id;
        this.国 = country;
        this.名前 = name;
        this.年齢 = age;
        this.重量 = weight;
```

```
            this.職業 = occupation;
            this.電話番号 = phone;
        }
    }

    public enum Occupation
    {
        医者,
        芸術家,
        教育者,
        エンジニア,
        支配人,
        その他
    }
```

2. クラスコンストラクタでプライベートメソッド **InitializeFlexGrid** を作成し、次のコードを追加してレコードのコレクションを作成します。
    - **Visual Basic**

```
Dim data As New List(Of Customer)()
data.Add(New Customer("100001", "米国", "Jack Danson",
         40, 102.03, 職業.支配人, "1371234567"))
data.Add(New Customer("100002", "中国", "Tony Tian",
32, 82.2, 職業.エンジニア, "1768423846"))
data.Add(New Customer("100003", "イラン", "Larry Frommer",
         15, 40.432, 職業.芸術家, "8473637486"))
data.Add(New Customer("100004", "ドイツ", "Charlie Krause",
         26, 69.32, 職業.医師, "675245438"))
data.Add(New Customer("100005", "インド", "Mark Ambers",
         51, 75.45, 職業.その他, "1673643842"))
```

    - **C#**

```
List<Customer> data = new List<Customer>();
data.Add(new Customer("100001", "米国", "Jack Danson",
          40, 102.03, 職業.支配人, "1371234567"));
data.Add(new Customer("100002", "中国", "Tony Tian",
          32, 82.2, 職業.芸術家, "1768423846"));
data.Add(new Customer("100003", "イラン", "Larry Frommer",
          15, 40.432, 職業.芸術家, "8473637486"));
data.Add(new Customer("100004", "ドイツ", "Charlie Krause",
          26, 69.32, 職業.医者, "675245438"));
data.Add(new Customer("100005", "インド", "Mark Ambers",
          51, 75.45, 職業.その他, "1673643842"));
```

3. MainPage クラスのコンストラクタで **InitializeFlexGrid** メソッドを呼び出します。

先頭に戻る

手順 3：データソースへの FlexGrid の連結

1. コードビューで、**FlexGrid** コントロールを **Customer** クラスに連結します（「クイックスタート」参照）。
    - **Visual Basic**

```
'FlexGridをCustomerにバインドし、セルのマージを許可します
FlexGrid.ItemsSource = data
```

    - **C#**

```
//FlexGridをCustomerにバインドし、セルのマージを許可します
FlexGrid.ItemsSource = data.ToList<Customer>();
```

2. アプリケーションをデバッグして、エラーがないかどうかを確認します。
3. **[F5]**キーを押してアプリケーションを実行し、出力を確認します。

先頭に戻る