ComponentOne

InputPanel for WPF

GrapeCity US

GrapeCity
201 South Highland Avenue, Suite 301
Pittsburgh, PA 15206
Tel: 1.800.858.2739 | 412.681.4343
Fax: 412.681.4384
Website: https://www.grapecity.com/en/
E-mail: us.sales@grapecity.com

Trademarks

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for $25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

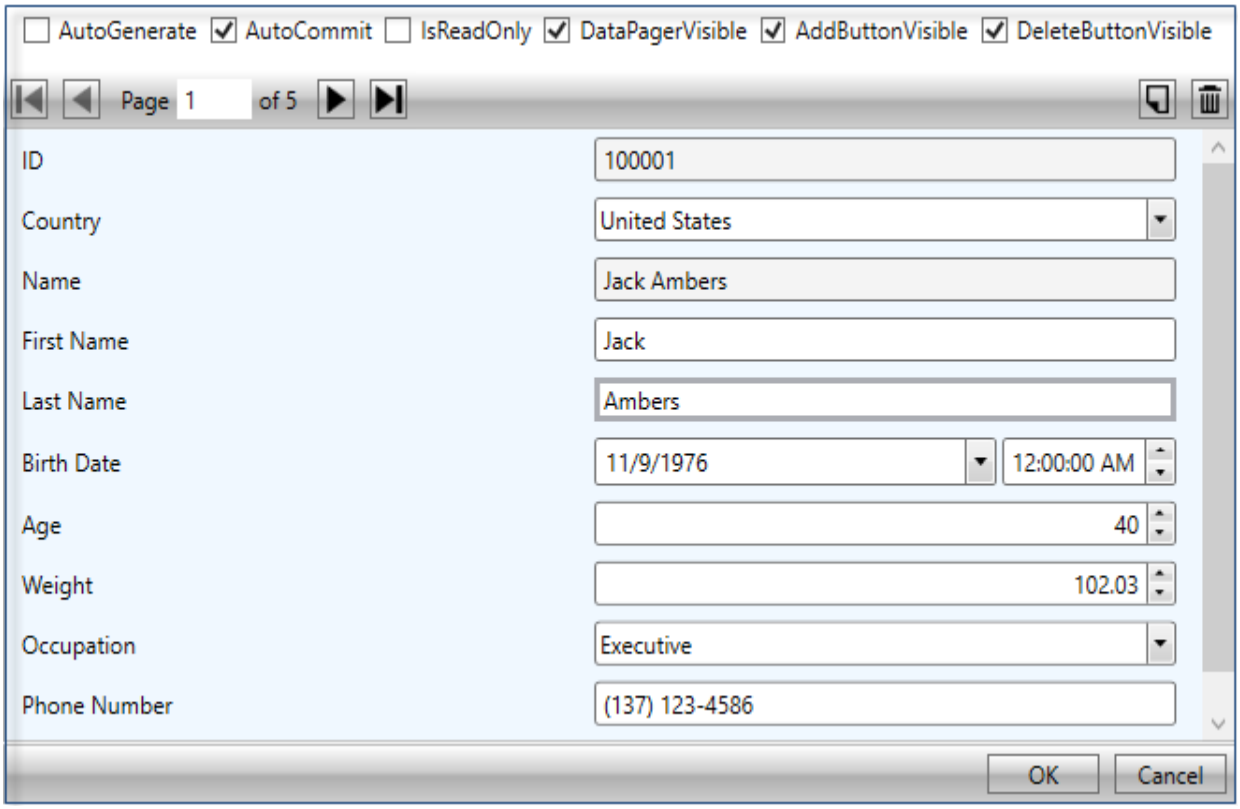
Table of Contents

InputPanel for WPF 2
Help with WPF Edition 2
Key Features 3
Object Model Summary 4
InputPanel Elements 5-6
InputPanel Editors 7
Quick Start 8-11
Data Binding 12
Binding InputPanel with ObservableCollection 12-18
Binding InputPanel with CollectionView 18-23
Binding InputPanel with DataTable 23-25
Features 26
Add, Edit, and Delete Records 26
Record Navigation 26-27
Auto-generate Fields 27-28
Auto-commit Data 28
Data Validation 28-29
Property Level Validation 29-32
Data Validation through Event 32-33
Custom Template 33-35
Keyboard Navigation 35-36
Working with InputPanel 37
Integration with Grids 37-40
Input Panel Samples 41

InputPanel for WPF

ComponentOne Studio introduces **InputPanel for WPF**, a container control designed to create powerful data input applications with minimal effort. The control is a new paradigm to create and maintain data forms that lets you easily view, add, edit, delete, and navigate through data records.

InputPanel can be easily bound to a model object, data table or a collection which automatically populates the UI with built-in editors and hence, there is no need to add any input components explicitly. These editors handle basic level validation for you, so you can handle the inconsistencies in user input with minimal development effort. Besides, the InputPanel control provides you the ability to customize its layout, design and appearance for easy and consistent styling.



Help with WPF Edition

For information on installing **ComponentOne Studio WPF Edition**, licensing, technical support, namespaces, and creating a project with the controls, please visit Getting Started with ComponentOne Studio WPF Edition.

Key Features

- **Auto-generate fields**
  InputPanel automatically generates fields on the basis of the input provided to the control. As the fields get generated, UI of the InputPanel control gets automatically populated with various components depending on the data type of the fields.

- **Add, edit and delete operations**
  InputPanel allows you to add new records, and also lets you edit and delete the current record in data source. It also allows you to delete the currently displayed record.

- **Auto-commit data**
  InputPanel allows you to automatically commit/save the changes every time the user edits the input, or customize this functionality to commit the edits after confirmation.

- **Data binding**
  InputPanel can be bound to model object data, CollectionView, ObservableCollection or any kind of collection data source with minimal code, allowing you to create a fully-navigational database browser in seconds.

- **Validation and error handling**
  InputPanel provides data validation on the user input. Validation errors appear as per the rules defined either through markup or through events.

- **Record navigation**
  InputPanel provides you the buttons to navigate through the records in the UI itself. No extra code is required to implement a pager in the control.

- **Data template support**
  InputPanel allows you to change the layout of the control by creating a user-defined data template. By default, the fields are stacked vertically; however, you can define a layout by creating your own data template as per the requirement.

- **Custom field support**
  InputPanel provides six default editors for rendering the fields on the control. However, you can also replace the standard fields with different input fields according to your requirements.

- **Accelerator keys**
  InputPanel allows you to use the keyboard to edit and navigate through the items without using mouse. The keyboard accelerator keys are automatically generated and assigned when the InputPanel control is populated with the fields from a data source.

Object Model Summary

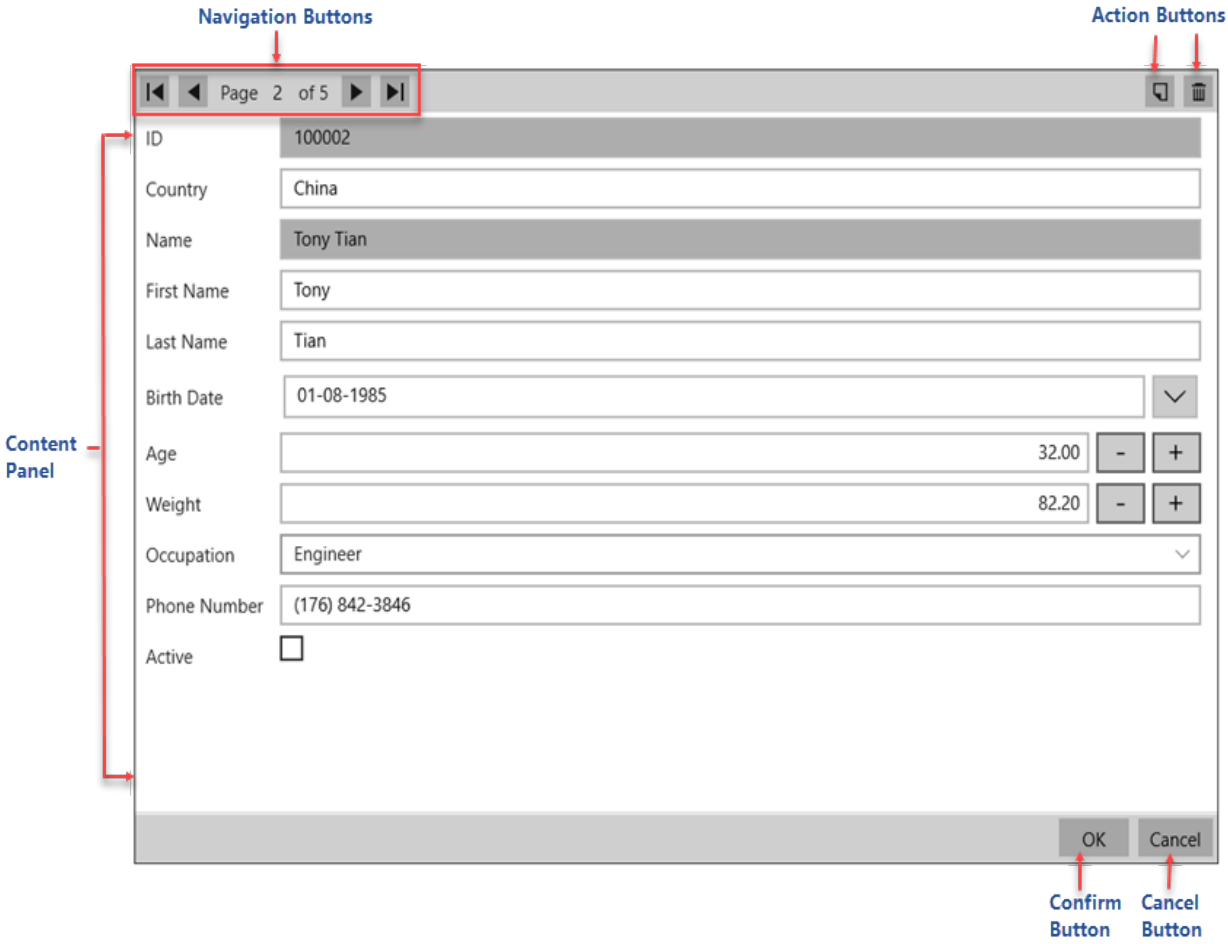
InputPanel comes with a rich object model, providing various classes, objects, collections, associated methods and properties for processing images. The following table lists some of these objects and their properties.

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

InputPanel Elements

InputPanel control provides four types of elements: navigation buttons, action buttons, content panel, and commit/cancel button.

The following image shows the elements of the InputPanel control.



The following table describes the elements of InputPanel control:

Element	Name	Description
	Navigation buttons	Allows navigation to a specific record through next/previous/first/last buttons. These buttons are provided at the top left of the InputPanel control.
	Action buttons	Includes the Add button to add records and Delete button to delete records. These buttons are available at the top right of the InputPanel control.

	Content Panel	Indicates the area that contains the fields in form layout where user can view/edit the records at runtime.
	Commit/Cancel button	Consists of OK/Cancel button where edits are saved/committed with the OK button and cancelled using the Cancel button. These buttons are available at the bottom right of the InputPanel control.

InputPanel Editors

InputPanel provides the following six types of editors to support different types of data. The following editors get automatically rendered for their supported data types in the content panel of the InputPanel control when AutoGenerate property is set to true.

- **InputTextBox**
  The InputTextBox editor is used for fields with their data type set to string or char.

- **InputDateTimePicker**
  The InputDateTimePicker editor is used for fields with their data type set to date, time or datetime. User can edit/enter the date by selecting it from the calendar that appears on clicking the editor, and the time by using the available "+" or "-" buttons. You can modify the behavior of the field using EditMode property of C1InputDateTimePicker class.

- **InputCheckBox**
  The InputCheckBox editor is used for fields with their data type set to Boolean. This editor allows you to set the IsThreeState property of C1InputCheckBox class, to support the three states, checked, unchecked, and indeterminate state of the check box .

- **InputComboBox**
  The InputComboBox editor is used for fields with their data type set to enumeration. The editor displays a list of items in a drop-down menu. This editor allows you to bind to a data source using the ItemsSource property.

- **InputNumbericBox**
  The InputNumbericBox editor is used for fields with their data type set to numeric. You can increase or decrease the value by clicking the "+" and "-" buttons provided by the control. This editor allows you to set the display format using Format property of C1InputNumbericBox class.

- **InputMaskedTextBox**
  The InputMaskedTextBox editor is used for fields in which each character position maps to either a special placeholder or a literal character. Literal characters, or literals, can give visual cues about the type of data being used. This editor allows you to set the format of masked text using the Mask property of C1InputMaskedTextBox class.

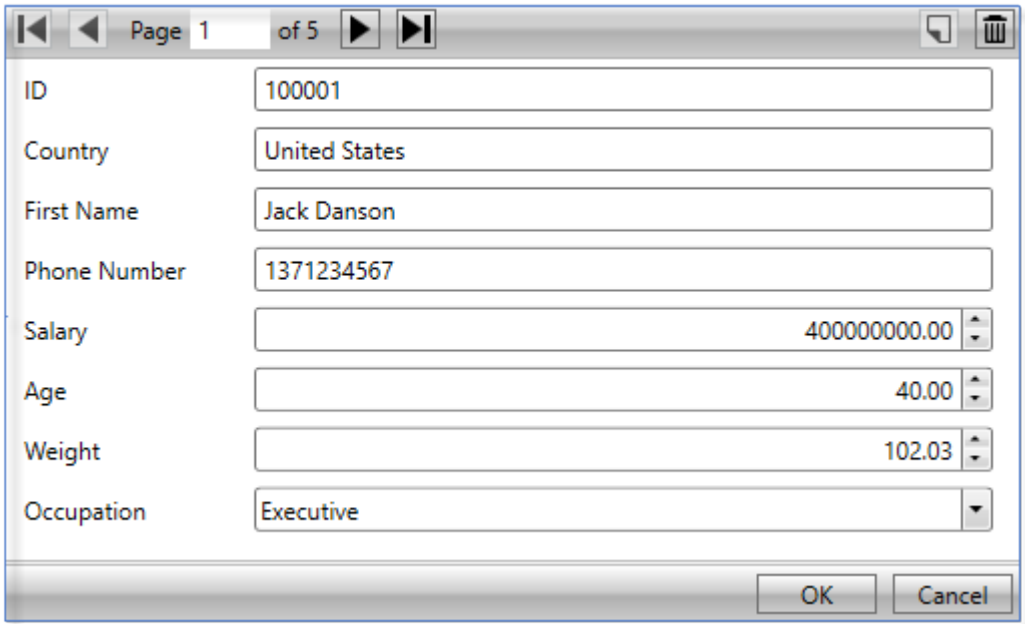
Quick Start

This quick start familiarizes you with adding and displaying data to the InputPanel control using list. You begin with creating a WPF application in Visual Studio, adding the InputPanel control to it, creating a list of items, and binding it to InputPanel.

To create a simple WPF application for adding and displaying data in the InputPanel control, complete the following steps:

1. **Setting up the application**
2. **Adding and displaying data in InputPanel**

The following image shows a record displayed in the C1InputPanel control.



Setting up the application

1. Create a **WPF** project in Visual Studio.
2. Add the **InputPanel** control to the **XAML** designer and set the name of the control to **'InPanel'**.
   Notice that along with C1.WPF.InputPanel, the following references automatically get added to the application.
   - C1.WPF
   - C1.WPF.DateTimeEditors

Back to Top

Adding and displaying data in InputPanel

1. Switch to the code view and add a class, **Customer**, to define data.
2. Add the following code to create an enumeration and add properties to the class.
   - **Visual Basic**

```
Public Class Customer

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

```
    End Set
End Property
Private m_ID As String
Public Property Country() As String
    Get
        Return m_Country
    End Get
    Set(value As String)
        m_Country = value
    End Set
End Property
Private m_Country As String

Public Property Name() As String
    Get
        Return m_Name
    End Get
    Set(value As String)
        m_Name = value
    End Set
End Property
Public Property Phone() As String
    Get
        Return m_Phone
    End Get
    Set(value As String)
        m_Phone = value
    End Set
End Property
Private m_Phone As String
Private m_Name As String
Public Property Age() As Integer
    Get
        Return m_Age
    End Get
    Set(value As Integer)
        m_Age = value
    End Set
End Property
Private m_Age As Integer
Public Property Weight() As Double
    Get
        Return m_Weight
    End Get
    Set(value As Double)
        m_Weight = value
    End Set
End Property
Private m_Weight As Double
Public Property Occupation() As Occupation
    Get
        Return m_Occupation
    End Get
    Set(value As Occupation)
        m_Occupation = value
    End Set
End Property
Private m_Occupation As Occupation


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.Country = country
        Me.Name = name
        Me.Age = age
        Me.Weight = weight
        Me.Occupation = occupation
        Me.Phone = phone
    End Sub
End Class

Public Enum Occupation
    Doctor
    Artist
    Educator
    Engineer
    Executive
    Other
End Enum
```

- **C#**

```
public class Customer
{
    public string ID { get; set; }
    public string Country { get; set; }
    public string Name { get; set; }
    public string Phone { get; set; }
    public int Salary { get; set; }
    public int Age { get; set; }
    public double Weight { get; set; }
    public Occupation Occupation { get; set; }


    public Customer(string id, string country,
        string name, int age,
        double weight, Occupation
        occupation, string phone, int salary)
    {
        this.ID = id;
        this.Country = country;
        this.Name = name;
        this.Age = age;
        this.Weight = weight;
        this.Occupation = occupation;
        this.Phone = phone;
        this.Salary = salary;
    }
}

public enum Occupation
{
    Doctor,
    Artist,
    Educator,
    Engineer,
    Executive,
    Other
}
```

3. Creates a list of Customers and add data to the list using the following code.

- **Visual Basic**

```
Dim data As New List(Of Customer)()
data.Add(New Customer("100001", "United States",
                      "Jack Danson", 40, 102.03,
                      Occupation.Executive, _
    "1371234567"))
data.Add(New Customer("100002", "China",
                      "Tony Tian", 32, 82.2,
                      Occupation.Engineer, _
    "1768423846"))
data.Add(New Customer("100003", "Iran",
                      "Larry Frommer", 15, 40.432,
                      Occupation.Artist, _
    "8473637486"))
data.Add(New Customer("100004", "Germany",
                      "Charlie Krause", 26, 69.32,
                      Occupation.Doctor, _
    "675245438"))
data.Add(New Customer("100005", "India",
                      "Mark Ambers", 51, 75.45,
                      Occupation.Other, _
    "1673643842"))
```

- **C#**

```
List<Customer> data = new List<Customer>();
data.Add(new Customer("100001", "United States", "Jack Danson",
    40, 102.03, Occupation.Executive, "1371234567", 400000000));
data.Add(new Customer("100002", "China", "Tony Tian",
    32, 82.2, Occupation.Engineer, "1768423846", 500));
data.Add(new Customer("100003", "Iran", "Larry Frommer",
    15, 40.432, Occupation.Artist, "8473637486", 600));
data.Add(new Customer("100004", "Germany", "Charlie Krause",
    26, 69.32, Occupation.Doctor, "675245438", 700));
data.Add(new Customer("100005", "India", "Mark Ambers",
    51, 75.45, Occupation.Other, "1673643842", 800));
```

4. Bind the list to InputPanel through the ItemsSource property as given in the following code.

- **Visual Basic**

```
InPanel.ItemsSource = data
```

- **C#**

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

Back to Top

Data Binding

InputPanel provides data binding support that lets you populate data in the control using a single line of code. The InputPanel control can be bound to data from various data sources, such as model object data, data tables, CollectionView, ObservableCollection, or any other kind of collection data. On binding InputPanel to a data source object, fields are generated for underlying data.

Learn about these implementations in detail in the topic enlisted below.

Binding InputPanel with ObservableCollection
: Learn how to bind data using ObservableCollection in code.

Binding InputPanel with CollectionView
: Learn how to bind data using CollectionView in code.

Binding InputPanel with DataTable
: Learn how to bind data using data table in code.

Binding InputPanel with ObservableCollection

Collection binding can be implemented in InputPanel using ObservableCollection which works similar to a regular collection. To bind InputPanel to an ObservableCollection, the ObservableCollection<T> class is used to obtain a collection, which acts as a binding source and then the ItemsSource property gets this collection to bind it to the InputPanel control.

Perform the following steps for data binding using ObservableCollection<T> class:

1. **Set up the application**
2. **Create a data source for InputPanel**
3. **Bind InputPanel to ObservableCollection**

Set up the application

1. Create a WPF application.
2. Add the InputPanel control to the application and name it 'InPanel'.

Back to Top

Create a data source for InputPanel

1. Add a new class, **Employee**, to the application.
2. Add the data generators and fields to the class.
    - **Visual Basic**

```
' ** fields
Private m_id As Integer, cid As Integer
Private m_first As String, m_last As String
Private m_father As String
Private m_occupation As EOccupation
Private m_active As Boolean
Private m_hired As DateTime
Private m_weight As Double

' ** data generators
Shared rnd As New Random()
Shared firstNames As String() = "Andy|Ben|Charlie|Dan|Ed|Fred|Gil".Split("|"c)
Shared lastNames As String() = "Ambers|Bishop|Cole|Danson|Evers|Trask|Ulam".Split("|"c)
Shared countries As String() = "China|India|United States|Japan|Myanmar".Split("|"c)
```

    - **C#**

```
// ** fields
int id, cid;
string first, last;
string father;
```

```
EOccupation occupation;
bool active;
DateTime hired;
double weight;

// ** data generators
static Random rnd = new Random();
static string[] firstNames = "Andy|Ben|Charlie|Dan|Ed|Fred|Gil".Split('|');
static string[] lastNames = "Ambers|Bishop|Cole|Danson|EversTrask|Ulam".Split('|');
static string[] countries = "China|India|United States|Japan|Myanmar".Split('|');
```

3. Add properties to the class using the following code.
    - **Visual Basic**

```
Public Property ID() As Integer
    Get
        Return m_id
    End Get
    Set(value As Integer)
        If value <> m_id Then
            m_id = value
        End If
    End Set
End Property
Public ReadOnly Property Name() As String
    Get
        Return String.Format("{0} {1}", First, Last)
    End Get
End Property

Public ReadOnly Property Country() As String
    Get
        Return countries(cid)
    End Get
End Property


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


Public Property Occupation() As EOccupation
    Get
        Return m_occupation
    End Get
    Set(value As EOccupation)
        If value <> m_occupation Then
            m_occupation = value
        End If
    End Set
End Property



Public Property Active() As Boolean
    Get
        Return m_active
    End Get
```

```
    Set(value As Boolean)
        If value <> m_active Then
            m_active = value
        End If
    End Set
End Property


Public Property First() As String
    Get
        Return m_first
    End Get
    Set(value As String)
        If value <> m_first Then
            m_first = value
        End If
    End Set
End Property


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


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


Public Property Weight() As Double
    Get
        Return m_weight
    End Get
    Set(value As Double)
        If value <> m_weight Then
            m_weight = value
        End If
    End Set
End Property

' some read-only stuff
Public ReadOnly Property Father() As String
    Get
        Return m_father
    End Get
End Property

' ** utilities
Private Shared Function GetString(arr As String()) _
    As String
    Return arr(rnd.[Next](arr.Length))
```

```
End Function

' ** static value providers
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 Name
{
    get { return string.Format("{0} {1}",
        First, Last); }
}

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

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

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

public bool Active
{
```

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

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

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

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

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

    // some read-only stuff
    public string Father
    {
        get { return father; }
    }

    // ** utilities
    static string GetString(string[] arr)
```

```
{
    return arr[rnd.Next(arr.Length)];
}

// ** static value providers
public static string[] GetCountries() { return countries; }
public static string[] GetFirstNames() { return firstNames; }
public static string[] GetLastNames() { return lastNames; }
```

4. Create a constructor of Employee class and add the following code to it.
   - **Visual Basic**

```
Private values As Array = [Enum].GetValues(GetType(EOccupation))
Public Sub New(id__1 As Integer)
    ID = id__1
    First = GetString(firstNames)
    Last = GetString(lastNames)
    CountryID = rnd.[Next]() Mod countries.Length
    Occupation = CType(values.GetValue(rnd.[Next](values.Length - 1)), EOccupation)
    Active = rnd.NextDouble() >= 0.5
    Hired = DateTime.Today.AddDays(-rnd.[Next](1, 365))
    Weight = 50 + rnd.NextDouble() * 50
    m_father = String.Format("{0} {1}", GetString(firstNames), Last)
End Sub
```

   - **C#**

```
Array values = Enum.GetValues(typeof(EOccupation));
public Employee(int id)
{
    ID = id;
    First = GetString(firstNames);
    Last = GetString(lastNames);
    CountryID = rnd.Next() % countries.Length;
    Occupation = (EOccupation)
        (values.GetValue(rnd.Next(values.Length - 1)));
    Active = rnd.NextDouble() >= .5;
    Hired = DateTime.Today.AddDays(-rnd.Next(1, 365));
    Weight = 50 + rnd.NextDouble() * 50;
    father = string.Format("{0} {1}",
        GetString(firstNames), Last);
}
```

5. Create a method, GetEmployeeList, of ObservableCollection<T> class using the following code.
   - **Visual Basic**

```
' ** static list provider
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#**

```
// ** static list provider
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;
}
```

Back to Top

Bind InputPanel to ObservableCollection

1. Add the following code to bind the InputPanel control with data using the ItemsSource property.
    - **Visual Basic**

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

    - **C#**

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

2. Press F5 to run the application.

Back to Top

Binding InputPanel with CollectionView

Collection binding can be implemented in InputPanel using ICollectionView, an interface with record management, filtering, grouping, and sorting functionalities. To bind InputPanel to an ObservableCollection, InputPanel can be bound to an object that implements the ICollectionView interface. In the following example, we have used the ObservableCollection<T> class as a binding source to obtain the collection and the **CollectionView** class that implements the ICollectionView interface to display the source collection. Later, bind the InputPanel control to the ICollectionView using the ItemsSource property of the C1InputPanel class.

Perform the following steps for data binding using ICollectionView:

1. **Set up the application**
2. **Create a data source for InputPanel**
3. **Bind InputPanel to ICollectionView**

Set up the application

1. Create a WPF application.
2. Add InputPanel control to the application and name it 'InPanel'.

Back to Top

Create a data source for InputPanel

1. Add a new class, **Product**, to the application.
2. Add the following fields to the class.
    - **Visual Basic**

```
Shared lines As String() = "Computers|Washers|Stoves".Split("|"c)
Shared colors As String() = "Red|Green|Blue|White".Split("|"c)
```

    - **C#**

```
static string[] lines = "Computers|Washers|Stoves".Split('|');
static string[] colors = "Red|Green|Blue|White".Split('|');
```

3. Add the following properties and methods to the class.
    - **Visual Basic**

```
Public Property Line() As String
    Get
        Return DirectCast(GetValue("Line"), String)
    End Get
    Set(value As String)
        SetValue("Line", value)
    End Set
End Property


Public Property Color() As String
    Get
```

```
        Return DirectCast(GetValue("Color"), String)
    End Get
    Set(value As String)
        SetValue("Color", value)
    End Set
End Property


Public Property Name() As String
    Get
        Return DirectCast(GetValue("Name"), String)
    End Get
    Set(value As String)
        SetValue("Name", value)
    End Set
End Property


Public Property Price() As Double
    Get
        Return CDbl(GetValue("Price"))
    End Get
    Set(value As Double)
        SetValue("Price", value)
    End Set
End Property


Public Property Weight() As Double
    Get
        Return CDbl(GetValue("Weight"))
    End Get
    Set(value As Double)
        SetValue("Weight", value)
    End Set
End Property


Public Property Cost() As Double
    Get
        Return CDbl(GetValue("Cost"))
    End Get
    Set(value As Double)
        SetValue("Cost", value)
    End Set
End Property


Public Property Volume() As Double
    Get
        Return CDbl(GetValue("Volume"))
    End Get
    Set(value As Double)
        SetValue("Volume", value)
    End Set
End Property


Public Property Discontinued() As Boolean
    Get
        Return CBool(GetValue("Discontinued"))
    End Get
```

```
    Set(value As Boolean)
        SetValue("Discontinued", value)
    End Set
End Property


Public Property Rating() As Integer
    Get
        Return CInt(GetValue("Rating"))
    End Get
    Set(value As Integer)
        SetValue("Rating", value)
    End Set
End Property

' get/set values
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
Public Shared Function GetLines() As String()
    Return lines
End Function
```

- **C#**

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

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

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

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

[Display(Name = "Weight")]
public double Weight
```

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

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

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

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

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

// get/set values
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 lines;
}
```

4. Create a method, GetProducts, of IEnumerable interface using the following code.
    - **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 = lines[rnd.Next() % lines.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. Add the following code to create a property, CustomerCollectionView, of ICollectionView interface which uses the C1CollectionView class to display the source collection.

- **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 CollectionView(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 CollectionView(products);
        }
        return view;
    }
}
```

Back to Top

Bind InputPanel to ICollectionView

1. Add the following code to bind the InputPanel control with data using the ItemsSource property.
   - **Visual Basic**

```
InPanel.ItemsSource = CustomerCollectionView
```

   - **C#**

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

2. Press F5 to run the application.

Back to Top

Binding InputPanel with DataTable

InputPanel supports data binding through data tables. You can create a data table using the standard **DataTable** class and bind it to the InputPanel control by setting the ItemsSource property.

Complete the following steps to bind InputPanel to data through data table.

1. **Set up the application**
2. **Create a data table to bind with InputPanel**
3. **Bind the data table to InputPanel**

Set up the application

1. Create a WPF application.
2. Add the InputPanel control and name it InPanel.

Back to Top

Create a data table to bind with InputPanel

1. Add a new class, **Employee**, to the application.
2. Add the following import statement.
   - **Visual Basic**

```
Imports System.Data
```

   - **C#**

```
using System.Data;
```

3. Create an object, employee, of **DataTable** class.
   - **Visual Basic**

```
'Create a data table
Private _employees As DataTable = Nothing
```

- **C#**

```
//Create a data table
private DataTable employees = null;
```

4. Add data fields to be added in the table.
   - **Visual Basic**

```
'Add data fields
Shared _rnd As New Random()
Shared _firstNames As String() = "Andy|Ben|Charlie|Dan|Ed|Fred|Gil".Split("|"c)
Shared _lastNames As String() = "Ambers|Bishop|Cole|Danson|Evers|Trask|Ulam".Split("|"c)
Shared _countries As String() = "China|India|United States|Japan|Myanmar".Split("|"c)
```

   - **C#**

```
//Add data fields
static Random _rnd = new Random();
static string[] _firstNames = "Andy|Ben|Charlie|Dan|Ed|Fred|Gil".Split('|');
static string[] _lastNames = "Ambers|Bishop|Cole|Danson|Evers|Trask|Ulam".Split('|');
static string[] _countries = "China|India|United States|Japan|Myanmar".Split('|');
```

5. Add class definition to assign fields to the data table.
   - **Visual Basic**

```
'Initialize data table
Public ReadOnly Property Employees() As DataTable
    Get
        If _employees Is Nothing Then
            _employees = New DataTable("Employees")
            _employees.Columns.Add("ID", System.Type.[GetType]("System.String"))
            _employees.Columns.Add("FirstName", System.Type.[GetType]("System.String"))
            _employees.Columns.Add("LastName", System.Type.[GetType]("System.String"))
            _employees.Columns.Add("Countries", System.Type.[GetType]("System.String"))
            _employees.Columns.Add("BirthDate", System.Type.[GetType]("System.DateTime"))

            For row As Integer = 0 To 9
                Dim dRow As DataRow = _employees.NewRow()
                dRow("ID") = _rnd.[Next](100000, 999999).ToString()
                dRow("FirstName") = _firstNames(_rnd.[Next](_firstNames.Length))
                dRow("LastName") = _lastNames(_rnd.[Next](_lastNames.Length))
                dRow("Countries") = _countries(_rnd.[Next](_countries.Length))


                dRow("BirthDate") =
                    DateTime.Today.AddDays(-_rnd.[Next](1, 365))
                _employees.Rows.Add(dRow)
            Next
        End If
        Return _employees
    End Get
End Property
```

   - **C#**

```
//Initialize data table
public DataTable Employees
{
    get
    {
        if (employees == null)
        {
            employees = new DataTable("Employees");
            employees.Columns.Add("ID",
                System.Type.GetType("System.String"));
            employees.Columns.Add("FirstName",
                System.Type.GetType("System.String"));
            employees.Columns.Add("LastName",
                System.Type.GetType("System.String"));
            employees.Columns.Add("Countries",
                System.Type.GetType("System.String"));
            employees.Columns.Add("BirthDate",
                System.Type.GetType("System.DateTime"));

            for (int row = 0; row < 10; row++)
```

```
                {
                    DataRow dRow = employees.NewRow();
                    dRow["ID"] = _rnd.Next(100000, 999999).ToString();
                    dRow["FirstName"] = _firstNames[_rnd.Next(_firstNames.Length)];
                    dRow["LastName"] = _lastNames[_rnd.Next(_lastNames.Length)];
                    dRow["Countries"] = _countries[_rnd.Next(_countries.Length)]; ;
                    dRow["BirthDate"] = DateTime.Today.AddDays(-_rnd.Next(1, 365));
                    employees.Rows.Add(dRow);
                }
            }
            return employees;
        }
    }
```

Back to Top

Bind the data table to InputPanel

1. Set the **ItemsSource** property in XAML view to bind InputPanel with the data table.

XAML	copyCode

```
<c1:C1InputPanel Name="InPanel" ItemsSource="{Binding Employees}"/>
```

2. Switch to the **MainWindow.xaml.cs** file and set the **DataContext** property.
   - **Visual Basic**

```
'Set data context
Me.DataContext = New EmployeeDataContext()
```

   - **C#**

```
//Set data context
this.DataContext = new Employee();
```

Back to Top

Features

Features section comprises all the features available in the InputPanel control.

Add, edit, and delete records
: Learn how to add, edit, and delete records.

Record navigation
: Learn about the display modes provided by record navigation.

Auto-generate fields
: Learn how to generate fields automatically in code.

Auto-commit data
: Learn how to automatically save data using code.

Custom template
: Learn how to create a custom template.

Keyboard navigation
: Learn about the Keyboard keys used for navigation and editing.

Add, Edit, and Delete Records

InputPanel control comes with the Add and Delete buttons at the top right of the control. The Add button allows you to insert a new record into the collection of records and the Delete button allows you to delete the currently displayed record in a single click. The following image shows the Add and Delete buttons available in the InputPanel control.

InputPanel also allows you to edit the records, however, the control does not provide any dedicated button for editing. You can simply select a field in the displayed record and edit the content in it.

InputPanel allows you to set the visibility of the Add button through the AddButtonVisibility property and Delete button through the DeleteButtonVisibility property of the C1InputPanel class. By default, the visibility of these buttons is set to **true**, however, you can hide these buttons using the following code:

Visual Basic

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

C#

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

Record Navigation

InputPanel provides record navigation for making it easy to scroll through the records without any code implementation. InputPanel navigation enables movement to the first, previous, next, and last record of a collection. It

also allows you to jump to a particular record by entering the specific record number. The following image shows the navigation buttons available in the InputPanel control.

The following table provides information about the navigation buttons and options provided by the InputPanel control:

Name	Description
First	Moves to the first record.
Previous	Moves to the previous record.
Current Record	Shows the current record number.
Total Records	Shows the total number of records that can be displayed.
Next	Moves to the next record.
Last	Moves to the last record.

InputPanel allows you to set the visibility of the navigation panel through the DataPagerVisibility property of the C1InputPanel class. By default, the visibility of the panel is set to **Visible**, however, you can hide the panel using the following code:

Visual Basic

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

C#

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

Auto-generate Fields

InputPanel fields are automatically generated on the basis of data type of a particular field. Auto-generation of fields in InputPanel is supported through the AutoGenerate property of the C1InputPanel class, which is set to **true** by default. However, you can set the value of the AutoGenerate property to **false** if you want to create the fields on your own.

You can set the value of AutoGenerate property in XAML as well as code view.

In XAML

To set the value of the AutoGenerate property in XAML view, use the following code.

XAML

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

In Code

To set the value of the AutoGenerate property in code view, use the following code.

Visual Basic

```
InPanel.AutoGenerate = False
```

C#

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

Auto-commit Data

InputPanel allows you to automatically save the data without having to click the OK button. The auto-commit feature is supported in InputPanel through the AutoCommit property of the C1InputPanel class, which automatically saves the data after every input and is set to **true** by default. However, you can set the AutoCommit property to **false** for saving the edits after some confirmation or on clicking the OK button.

You can set the value of AutoCommit property in XAML as well as code view.

In XAML

To set the value of the AutoCommit property in XAML view, use the following code.

XAML

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

In Code

To set the value of the AutoCommit property in code view, use the following code.

Visual Basic

```
InPanel.AutoCommit = False
```

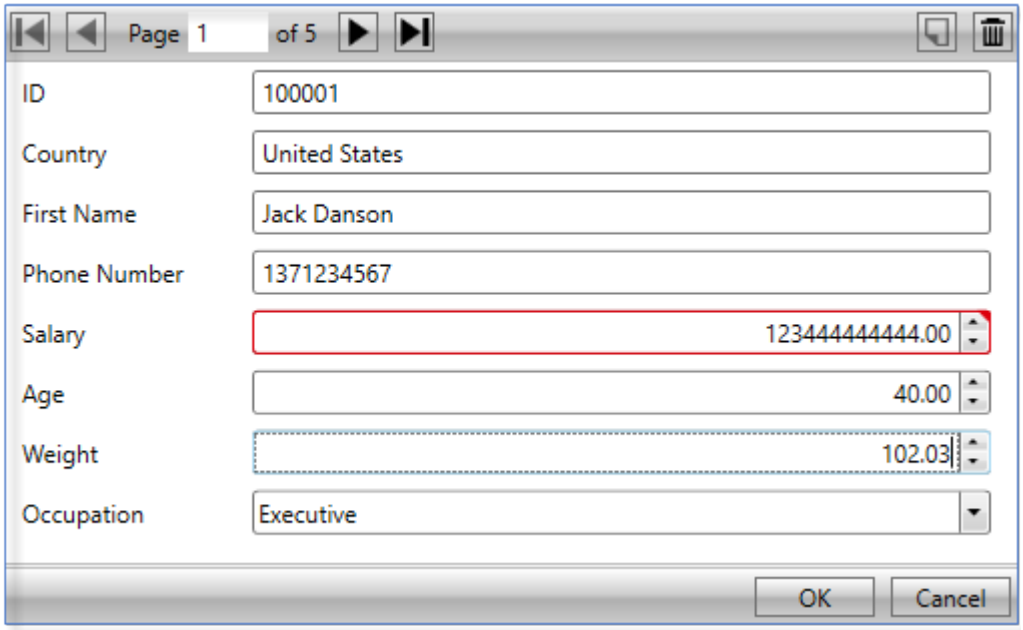
C#

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

Data Validation

InputPanel comes with data validation support to provide users the ability to check invalid user input. The control supports built-in validation, which is performed when invalid or contradictory data is entered in an input field and is committed. For every invalid/inaccurate input, the InputPanel control displays a red frame around the input field as a visual alert.

The image given below illustrates an example where entering an invalid input in one of the input fields shows a visual alert.



The built-in validation provides very basic input checking functionality to users. However, to apply validation in complex scenarios, the control supports property level validation through standard and custom attribute markup. You can apply the markup in code to validate data in diverse scenarios, such as checking for unnecessary spaces, null values, or special characters in text, or putting limits on the age being entered in a numeric. In addition, InputPanel provides an event to apply validation rules.

Here is how you can use validate user input in InputPanel.

Property level validation
: Learn how to implement property level validation in code.

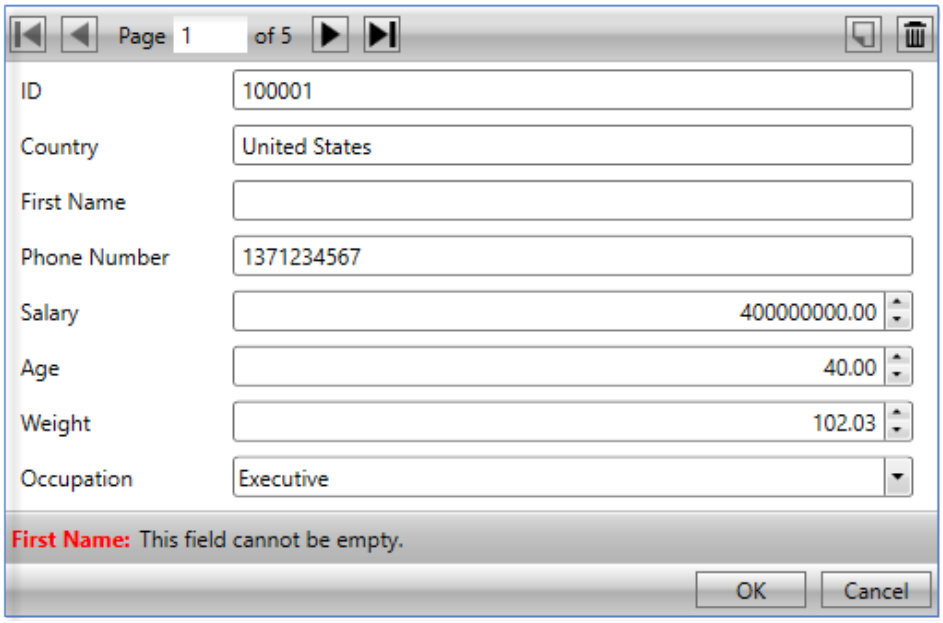
Data validation through event
: Learn how to apply different transformations in code.

Property Level Validation

InputPanel provides property level validation for validating user input in scenarios where built-in validation fails or remains insufficient. In this type of validation, data validation rules are specified inside the property setter code. InputPanel supports two types of markup to implement property level validation.

- **Standard attribute markup**
- **Custom markup**

The following image shows property level validation on entering invalid inputs.



Standard attribute markup

You can apply property level validation on user input by adding standard attribute markup inside the property setter code in the application. The control directly uses classes available in **System.ComponentModel.DataAnnotations** assembly to access this markup.

The following code illustrates adding standard attribute markup inside property setter code to apply validation. This example uses the sample created in the Quick start.

- **Visual Basic**

```
  <DisplayName("FirstName")>
 <Required(ErrorMessage:="This field cannot be empty.")>
  Public Property Name() As String
      Get
          Return m_Name
      End Get
      Set(value As String)
          m_Name = value
      End Set
  End Property

  <DisplayName("PhoneNumber")>
<Required(ErrorMessage:="This field cannot be empty.")>
  Public Property Phone() As String
      Get
          Return m_Phone
      End Get
      Set(value As String)
          m_Phone = value
      End Set
  End Property
```

- **C#**

```
[DisplayName("First Name")]
[Required(ErrorMessage = "This field cannot be empty.")]
public string Name { get; set; }

[DisplayName("Phone Number")]
[Required(ErrorMessage = "This field cannot be empty.")]
public string Phone { get; set; }
```

Custom markup

InputPanel also supports custom markup for achieving property level validation. Custom markup is useful in scenarios where users want to customize the validation rules as per their business needs. In addition, custom markup lets you combine multiple validation rules on an input field. For instance, custom markup allows validating **Phone Number** for null or white spaces as well as for minimum and maximum length in a single validation rule.

The steps given below illustrate creating and applying custom markup in code for property level validation. This example uses the sample created in the Quick start.

1. Create a class, **CustomValidator**, and define validation rules to check for null values or white spaces as well as minimum and maximum length of the **Phone Number** field.
   - **Visual Basic**

```
Public Class CustomValidator
    Public Shared Function ValidatePhoneNumber(PhoneNumber As String) As ValidationResult
        If String.IsNullOrWhiteSpace(PhoneNumber) Then
            Return New ValidationResult("Phone number cannot be none.",
                                        New List(Of String)() From { _
                "Phone" _
            })
        ElseIf PhoneNumber.Length > 12 OrElse PhoneNumber.Length < 9 Then
            Return New ValidationResult("Phone number should be more than" +
                                       " 8 digits and less than 12 digits.",
                                        New List(Of String)() From { _
                "Phone" _
            })
        Else

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

   - **C#**

```
public class CustomValidator
{
    public static ValidationResult
        ValidatePhoneNumber(string PhoneNumber)
    {
        if (string.IsNullOrWhiteSpace(PhoneNumber))
        {
            return new ValidationResult("Phone number cannot be none.",
                new List<string>() { "Phone" });
        }
        else if (PhoneNumber.Length > 12 || PhoneNumber.Length < 9)
        {
            return new ValidationResult
  ("Phone number should be more than 8 and less than 12 digits.",
                new List<string>() { "Phone" });
        }

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

2. Add the custom markup in the property setter code to validate the input entered in Phone Number field.
   - **Visual Basic**

```
  <DisplayName("PhoneNumber")>
<CustomValidation(GetType(CustomValidator), "ValidatePhoneNumber")>
  Public Property Phone() As String
      Get
          Return m_Phone
      End Get
      Set(value As String)
          m_Phone = value
```

```
        End Set
    End Property
```

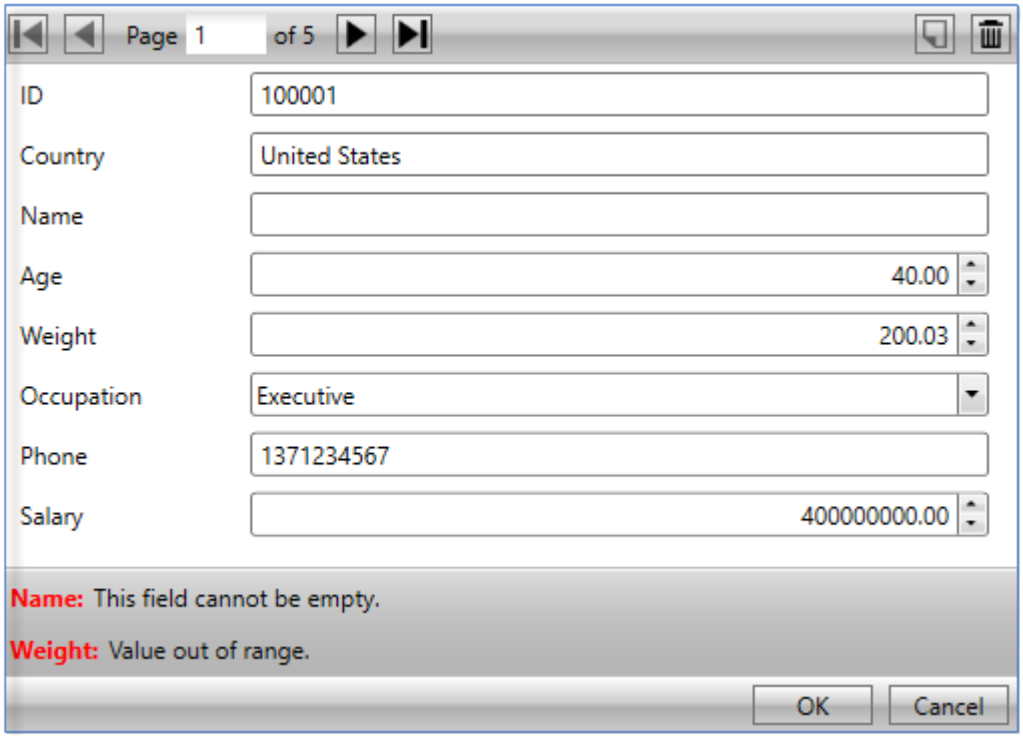
- **C#**

```
[DisplayName("Phone Number")]
[CustomValidation(typeof(CustomValidator), "ValidatePhoneNumber")]
public string Phone { get; set; }
public int Salary { get; set; }
```

Data Validation through Event

InputPanel provides another way of handling input validation through event. The control comes with the ValidateCurrentItem event that can be used for validating user input.

The following image shows validation applied through event.



To implement data validation through event, subscribe the **ValidateCurrentItem** event in code and add validation rules in the event handler. The following code shows how to apply validation using event. This example uses the sample created in the Quick start.

- **Visual Basic**

```
Private Sub InPanel_ValidateCurrentItem(sender As Object, e As CancelEventArgs) _
    Handles InPanel.ValidateCurrentItem
    Dim inputPanel As C1InputPanel = TryCast(sender, C1InputPanel)
    Dim customer As Customer = TryCast(inputPanel.CurrentItem, Customer)

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

        If customer.Name IsNot Nothing AndAlso _
            String.IsNullOrWhiteSpace(customer.Name.ToString()) Then
            errorList.Add(New ErrorInfo() With { _
                .ErrorInputName = "Name", _
                .ErrorContent = "This field cannot be empty" _
            })
        End If
```

```
        If customer.Weight > 110 Then
            errorList.Add(New ErrorInfo() With { _
                .ErrorInputName = "Weight", _
                .ErrorContent = "Value out of range." _
            })
        End If
        inputPanel.ValidationErrors = errorList
        If errorList.Count > 0 Then
            e.Cancel = True
        End If
    End If
End Sub
```

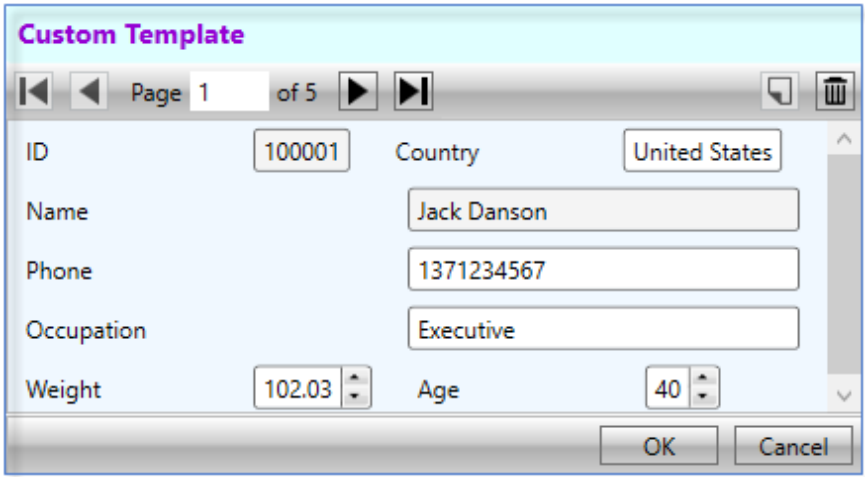
- **C#**

```
private void InPanel_ValidateCurrentItem
    (object sender, System.ComponentModel.CancelEventArgs e)
{
    C1InputPanel inputPanel = sender as C1InputPanel;
    Customer customer = inputPanel.CurrentItem as Customer;

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

        if (customer.Name != null &&
            string.IsNullOrWhiteSpace(customer.Name.ToString()))
        {
            errorList.Add(new ErrorInfo { ErrorInputName = "Name",
                ErrorContent = "This field cannot be empty." });
        }
        if (customer.Weight > 110)
        {
            errorList.Add(new ErrorInfo { ErrorInputName = "Weight",
                ErrorContent = "Value out of range." });
        }
        inputPanel.ValidationErrors = errorList;
        if (errorList.Count > 0)
        {
            e.Cancel = true;
        }
    }
}
```

Custom Template

InputPanel provides the flexibility to create custom template to change the layout of the control according to user requirements. By creating a custom template, you can develop compact and visually-appealing forms that match your application. InputPanel provides **DataTemplate** to define a custom layout and change the way the control appears.

The following image shows a custom template applied to the InputPanel control.



You can create a custom template using DataTemplate to change the layout of the control using the following steps. This example uses the sample created in Quick start. In this example, a data template is created that contains a StackPanel comprising two StackPanels with horizontal orientation. These two inner StackPanels consists horizontally-aligned editors while the outer StackPanel consists vertically-aligned editors. InputPanel accesses this data template through the ItemsSource property.

The example also showcases the customization of InputPanel header template using the HeaderTemplate property of the C1InputPanel class.

> By default, all the editors in InputPanel are stacked vertically. To change the alignment of editors, set the AutoGenerate property to **false**.

1. Add the following code inside the <UserControl.Resources> </UserControl.Resources> tags to create a custom data template with various editors stacked vertically and horizontally.

XAML copyCode

```
<UserControl.Resources>
    <DataTemplate x:Key="Template">
        <StackPanel Background="AliceBlue">
            <StackPanel Orientation="Horizontal">
                <c1:C1InputTextBox Header="ID" DataBinding="{Binding ID,
Mode=OneWay}"
                     IsReadOnly="True" LabelForeground="{Binding LabelForeground,
                    ElementName=InPanel}">
                </c1:C1InputTextBox>
                <c1:C1InputTextBox Header="Country" DataBinding="{Binding
Country,
                    Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
                    IsReadOnly="{Binding IsReadOnly, ElementName=InPanel}"
                    LabelForeground="{Binding LabelForeground,
ElementName=InPanel}">
                </c1:C1InputTextBox>
            </StackPanel>
            <c1:C1InputTextBox Header="Name" DataBinding="{Binding Name,
Mode=OneWay}"
                    IsReadOnly="True" LabelForeground="{Binding LabelForeground,
                ElementName=InPanel}"></c1:C1InputTextBox>
            <c1:C1InputMaskedTextBox Header="Phone" DataBinding="{Binding Phone,
Mode=TwoWay,
                UpdateSourceTrigger=PropertyChanged}"
```

```
                IsReadOnly="{Binding IsReadOnly, ElementName=InPanel}"
                LabelForeground="{Binding LabelForeground, ElementName=InPanel}">
            </c1:C1InputMaskedTextBox>
            <c1:C1InputTextBox Header="Occupation" DataBinding="{Binding
Occupation,
                Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
                IsReadOnly="{Binding IsReadOnly, ElementName=InPanel}"
                LabelForeground="{Binding LabelForeground, ElementName=InPanel}">
            </c1:C1InputTextBox>
            <StackPanel Orientation="Horizontal">
                <c1:C1InputNumericBox Header="Weight" DataBinding="{Binding
Weight,
                    Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
                    IsReadOnly="{Binding IsReadOnly, ElementName=InPanel}"
                    LabelForeground="{Binding LabelForeground,
ElementName=InPanel}">
                </c1:C1InputNumericBox>
                <c1:C1InputNumericBox Header="Age" DataBinding="{Binding Age,
Mode=TwoWay,
                    UpdateSourceTrigger=PropertyChanged}"
                    IsReadOnly="{Binding IsReadOnly, ElementName=InPanel}"
                    LabelForeground="{Binding LabelForeground,
ElementName=InPanel}">
                </c1:C1InputNumericBox>
            </StackPanel>
        </StackPanel>
    </DataTemplate>
    <ItemsPanelTemplate x:Key="ItemsPanel">
        <StackPanel Orientation="Vertical" Margin="20"/>
    </ItemsPanelTemplate>
</UserControl.Resources>
```

2. Add the following code inside the <Grid></Grid> tags to customize the InputPanel and header template.

XAML copyCode

```
<Grid>
    <c1:C1InputPanel x:Name="InPanel" AutoGenerate="False"
                     ItemsTemplate="{StaticResource Template}"
                     ItemsPanelTemplate="{StaticResource ItemsPanel}"
                    HeaderBackground="LightCyan" HeaderFontWeight="Bold"
     Margin="20,40,150,286">
        <c1:C1InputPanel.HeaderTemplate>
            <DataTemplate>
                <StackPanel>
                    <TextBlock Text="Custom Template" Margin="6"
                               Foreground="DarkViolet" />
                </StackPanel>
            </DataTemplate>
        </c1:C1InputPanel.HeaderTemplate>
                </c1:C1InputPanel>
</Grid>
```

Keyboard Navigation

InputPanel provides keyboard support that can be used to navigate through the records. The keyboard navigation keys can replace the use of mouse by allowing you to edit and navigate through the records. The keys and their corresponding actions are listed below:

Keys	Actions
Left key 	Navigates to the previous item.
Right key 	Navigates to the next items.
Ctrl + Home, Ctrl+Left arrow	Moves the current item to the first position.
Ctrl + End, Ctrl+Right arrow	Moves the current item to the last position.
Insert	Adds a new record.
Enter	Commits the edit operation.
Escape	Cancels the edit operation.
Delete	Deletes the current record.

Working with InputPanel

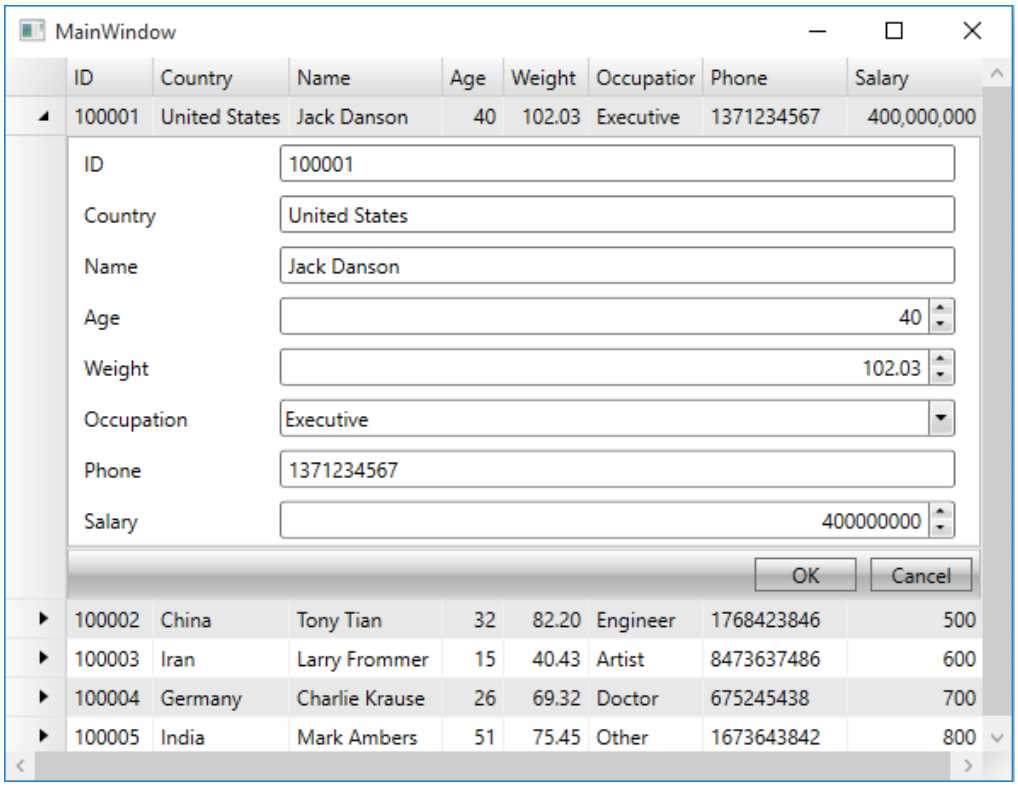
Working with InputPanel section assumes that you are familiar with the basics and features of the InputPanel control and know how to use it in general. The following section provides information on auxiliary functionality offered by InputPanel.

Integrating InputPanel with Grids
: Learn how to integrate InputPanel with different types of data grids in code.

Integration with Grids

InputPanel supports seamless integration with grid controls including **MS DataGrid** and **ComponentOne's FlexGrid** and **DataGrid**. These grid controls come with a baked-in data template, RowDetailsTemplate, which can be used to embed UI elements within a collapsible section for each row. Using this template, the InputPanel can be embedded to display the details of each row in a compact layout. You can interact with the template in XAML view, and set its binding in code to implement integration. In this section, we discuss how InputPanel can be integrated in FlexGrid control.

The following image shows an InputPanel integrated with a FlexGrid (C1FlexGrid).



To integrate InputPanel with ComponentOne FlexGrid

- **Step 1: Set up the application**
- **Step 2:Create a data source**
- **Step 3:Integrate InputPanel with FlexGrid**

Back to Top

Step 1: Set up the application

1. Create a **WPF** application and add the InputPanel control onto the designer.
2. Add **C1.WPF.DataGrid** dll in the **References** folder of your application.
3. Initialize the RowDetailsTemplate of the grid in XAML view and set binding property as illustrated.

XAML copyCode

```
<c1:C1FlexGrid Name="flexgrid">
```

```
    <c1:C1FlexGrid.RowDetailsTemplate>
        <DataTemplate>
            <c1:C1InputPanel CurrentItem="{Binding .}"/>
        </DataTemplate>
    </c1:C1FlexGrid.RowDetailsTemplate>
</c1:C1FlexGrid>
```

Step 2: Create a data source

1. Create a **Customer** class to add records into the InputPanel, and an enumeration to accept values for Occupation field.
    - **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 Country() As String
        Get
            Return m_Country
        End Get
        Set
            m_Country = Value
        End Set
    End Property
    Private m_Country As String

    Public Property Name() As String
        Get
            Return m_Name
        End Get
        Set
            m_Name = Value
        End Set
    End Property
    Private m_Name As String


    Public Property Age() As Integer
        Get
            Return m_Age
        End Get
        Set
            m_Age = Value
        End Set
    End Property
    Private m_Age As Integer
    Public Property Weight() As Double
        Get
            Return m_Weight
        End Get
        Set
            m_Weight = Value
        End Set
    End Property
    Private m_Weight As Double
    Public Property Occupation() As Occupation
        Get
            Return m_Occupation
        End Get
        Set
            m_Occupation = Value
        End Set
    End Property
    Private m_Occupation As Occupation
    Public Property Phone() As String
        Get
```

```
            Return m_Phone
        End Get
        Set
            m_Phone = Value
        End Set
    End Property
    Private m_Phone As String
    Public Property Salary() As Integer
        Get
            Return m_Salary
        End Get
        Set
            m_Salary = Value
        End Set
    End Property
    Private m_Salary As Integer

    Public Sub New(id As String, country As String, name As String,
                   age As Integer, weight As Double, occupation As Occupation,
        phone As String, salary As Integer)
        Me.ID = id
        Me.Country = country
        Me.Name = name
        Me.Age = age
        Me.Weight = weight
        Me.Occupation = occupation
        Me.Phone = phone
        Me.Salary = salary
    End Sub
End Class

Public Enum Occupation
    Doctor
    Artist
    Educator
    Engineer
    Executive
    Other
End Enum
```

- **C#**

```
public class Customer
{
    public string ID { get; set; }
    public string Country { get; set; }

    public string Name { get; set; }


    public int Age { get; set; }
    public double Weight { get; set; }
    public Occupation Occupation { get; set; }
    public string Phone { get; set; }
    public int Salary { get; set; }

    public Customer(string id, string country, string name,
        int age, double weight, Occupation occupation, string phone, int salary)
    {
        this.ID = id;
        this.Country = country;
        this.Name = name;
        this.Age = age;
        this.Weight = weight;
        this.Occupation = occupation;
        this.Phone = phone;
        this.Salary = salary;
    }
}
public enum Occupation
{
    Doctor,
    Artist,
    Educator,
    Engineer,
```

```
    Executive,
    Other
}
```

2. Switch to the MainWindow.xaml.cs file and add the following code to create a collection of records in the class constructor.
   - **Visual Basic**

```
Dim data As New List(Of Customer)()
data.Add(New Customer("100001", "United States", "Jack Danson", 40, 102.03, Occupation.Executive,
"1371234567", 400000000))
data.Add(New Customer("100002", "China", "Tony Tian", 32, 82.2, Occupation.Engineer,
"1768423846", 500))
data.Add(New Customer("100003", "Iran", "Larry Frommer", 15, 40.432, Occupation.Artist,
"8473637486", 600))
data.Add(New Customer("100004", "Germany", "Charlie Krause", 26, 69.32, Occupation.Doctor,
"675245438", 700))
data.Add(New Customer("100005", "India", "Mark Ambers", 51, 75.45, Occupation.Other,
"1673643842", 800))
```

   - **C#**

```
List<Customer> data = new List<Customer>();
data.Add(new Customer("100001", "United States", "Jack Danson",
    40, 102.03, Occupation.Executive, "1371234567", 400000000));
data.Add(new Customer("100002", "China", "Tony Tian",
    32, 82.2, Occupation.Engineer, "1768423846", 500));
data.Add(new Customer("100003", "Iran", "Larry Frommer",
    15, 40.432, Occupation.Artist, "8473637486", 600));
data.Add(new Customer("100004", "Germany", "Charlie Krause",
    26, 69.32, Occupation.Doctor, "675245438", 700));
data.Add(new Customer("100005", "India", "Mark Ambers",
    51, 75.45, Occupation.Other, "1673643842", 800));
```

Back to Top

Step 3: Integrate InputPanel with FlexGrid

1. To integrate the InputPanel with FlexGrid, set the ItemsSource property of the grid to the collection in the class constructor.
   - **Visual Basic**

```
flexgrid.ItemsSource = data
```

   - **C#**

```
flexgrid.ItemsSource = data.ToList<Customer>();
```

Back to Top

Similarly, you can integrate InputPanel with MS DataGrid and ComponentOne DataGrid using RowDetailTemplate property.

Input Panel Samples

With the C1Studio installer, you get InputPanel samples that help you understand the implementation of the product. A C# sample is available at the default installation folder:
Documents\ComponentOne Samples\WPF\C1.WPF.InputPanel\CS

Sample	Description
InputPanelSamples	This sample demonstrates an example of binding data to InputPanel in code.