

ListView, Trees and Grids

Agenda

- ListView
- TreeView
- DataGrid

ListView

```
<ListView Margin="5" Name="lstProducts">
  <ListView.View>
    <GridView>
      <GridView.Columns>
        <GridViewColumn Header="Name"
          DisplayMemberBinding="{Binding Path=ModelName}" />
        <GridViewColumn Header="Model"
          DisplayMemberBinding="{Binding Path=ModelName}" />
        <GridViewColumn Header="Price"
          DisplayMemberBinding="{Binding Path=Price}"
          StringFormat="{0:C2}" />
      </GridView.Columns>
    </GridView>
  </ListView.View>
</ListView>
```

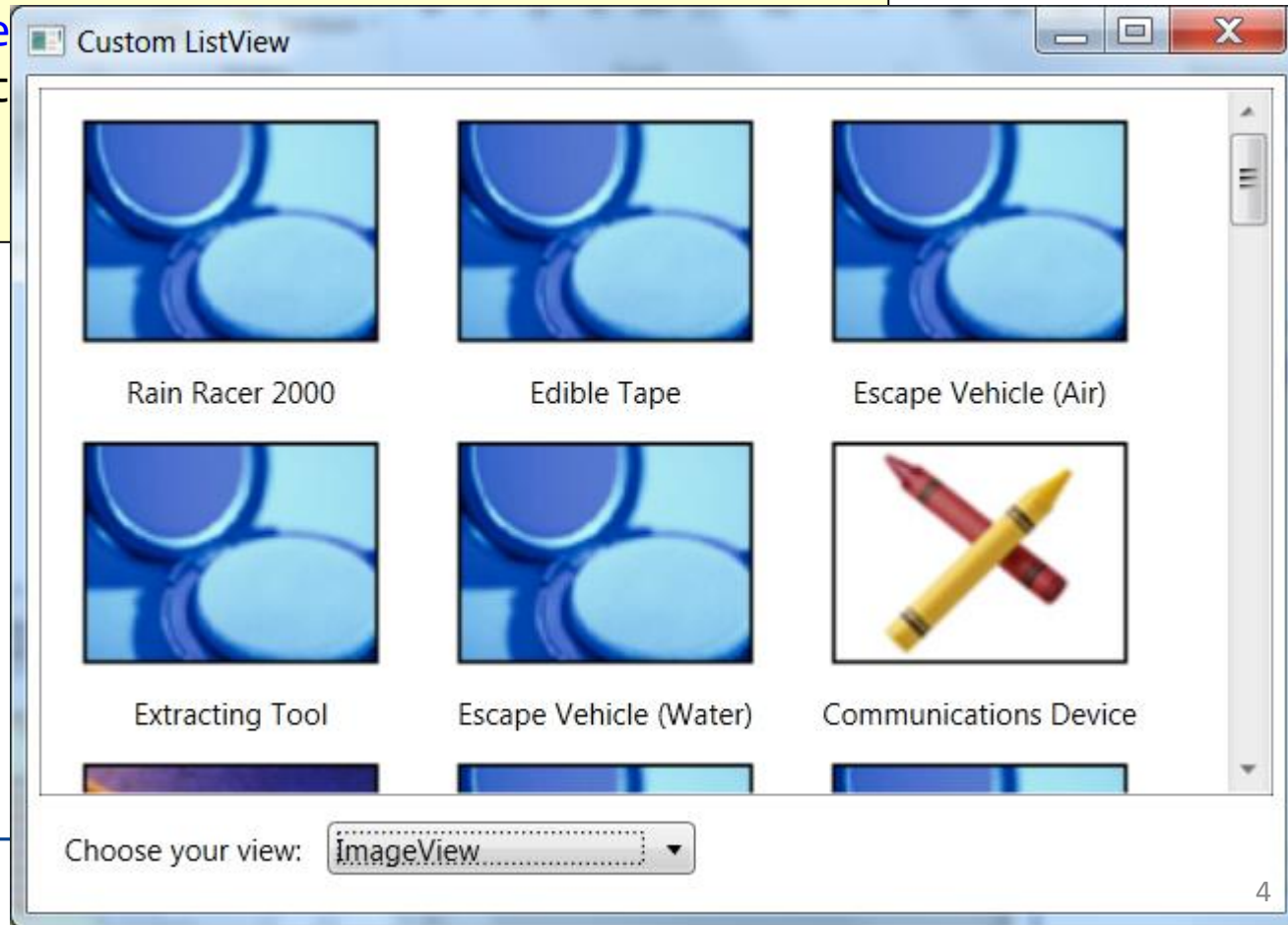


The screenshot shows a Windows application window titled "Basic ListView". Inside the window is a data grid with three columns: "Name", "Model", and "Price". The grid contains ten rows of data. The "Price" column is formatted with two decimal places and a dollar sign.

Name	Model	Price
Rain Racer 2000	RU007	\$1,499.99
Edible Tape	STKY1	\$3.99
Escape Vehicle (Air)	P38	\$2.99
Extracting Tool	NOZ119	\$199.00
Escape Vehicle (Water)	PT109	\$1,299.99
Communications Device	RED1	\$49.99
Persuasive Pencil	LK4TLNT	\$1.99
Multi-Purpose Rubber Band	NTMBS1	\$1.99
Universal Repair System	NE1RPR	\$4.99
Effective Flashlight	BRTLGT1	\$9.99
The Incredible Versatile Paperclip	INCPPRCLP	\$1.49
Toaster Boat	DNTRPR	\$19,999.98
Multi-Purpose Towelette	TGFDA	\$12.99

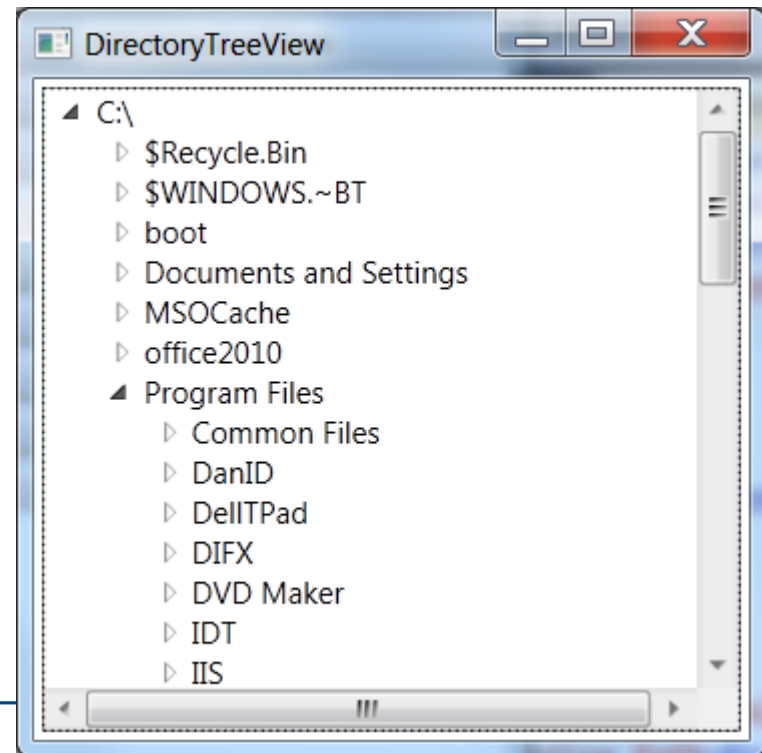
Custom ListView

```
public class TileView : ViewBase
{
    private DataTemplate itemTemplate;
    public DataTemplate ItemTemplate
    {
        get { return itemTemplate; }
        set { itemTemplate = value; }
    }
    . . .
}
```



TreeView

- The TreeView is a specialized ItemsControl that hosts TreeViewItem
- And each TreeViewItem is a separate HeaderedItemsControl
 - with the ability to hold more TreeViewItems
- This flexibility allows you to create a deeply layered data display



Hierarchical Binding

- A classic way to store and classify information is through the use of hierarchies
- The HierarchicalDataTemplate class is designed to be used with HeaderedItemsControl types to display such data
- **HierarchicalDataTemplates** differ from flat templates primarily through a single property: ItemsSource
- The idea here is that for any given bound object, it may have a collection of child objects
- **ItemsSource** allows you to specify what property you need to get to the children
 - By adding this property, HierarchicalDataTemplates can go into a bound data object recursively
 - If the children are of a different type than the parent you may need to specify an additional HierarchicalDataTemplates
(se example “Support for Hierarchical Data” from “Data Templating Overview” in the online help)

Hierarchical Binding

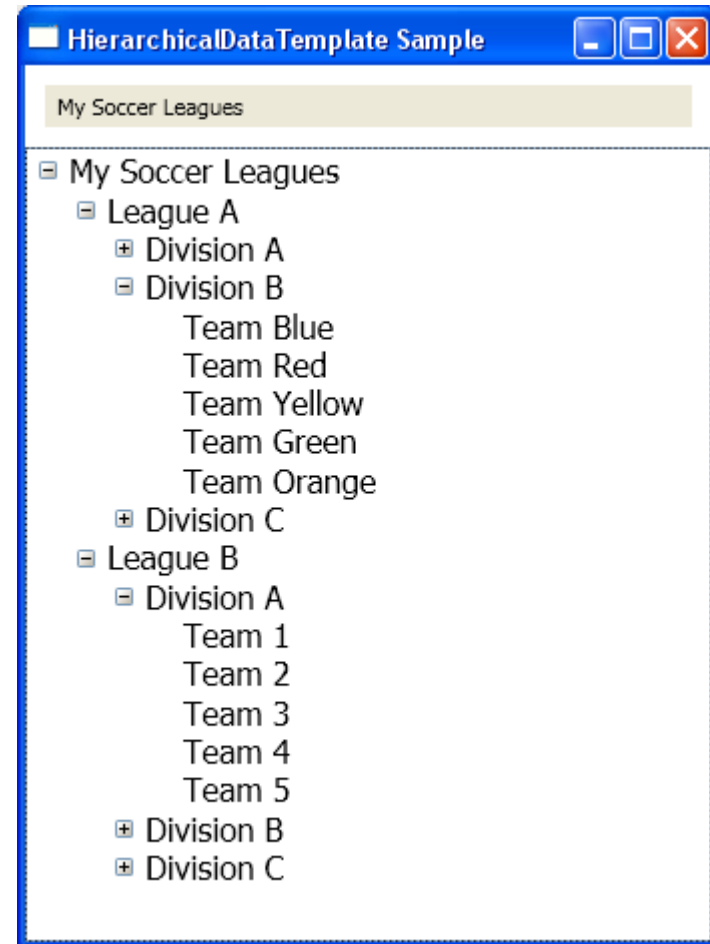
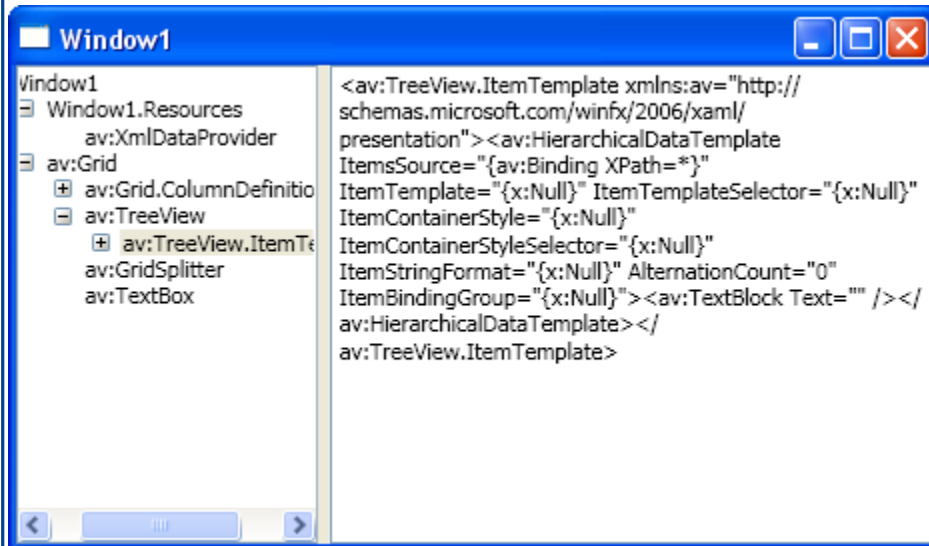
```
<TreeView Grid.Column="0" Name="treeView1"
  ItemsSource="{Binding Source={StaticResource xaml},
    XPath=*}">
  <TreeView.ItemTemplate>
    <HierarchicalDataTemplate ItemsSource="{Binding
      XPath=*}">
      <TextBlock Text="{Binding Path=Name}" />
    </HierarchicalDataTemplate>
  </TreeView.ItemTemplate>
</TreeView>
```

Another example:

```
<HierarchicalDataTemplate DataType = "{x:Type src:League}"
  ItemsSource = "{Binding Path=Divisions}">
  <TextBlock Text="{Binding Path=Name}" />
</HierarchicalDataTemplate>

<HierarchicalDataTemplate DataType = "{x:Type src:Division}"
  ItemsSource = "{Binding Path=Teams}">
  <TextBlock Text="{Binding Path=Name}" />
</HierarchicalDataTemplate>
```

Hierarchical Binding



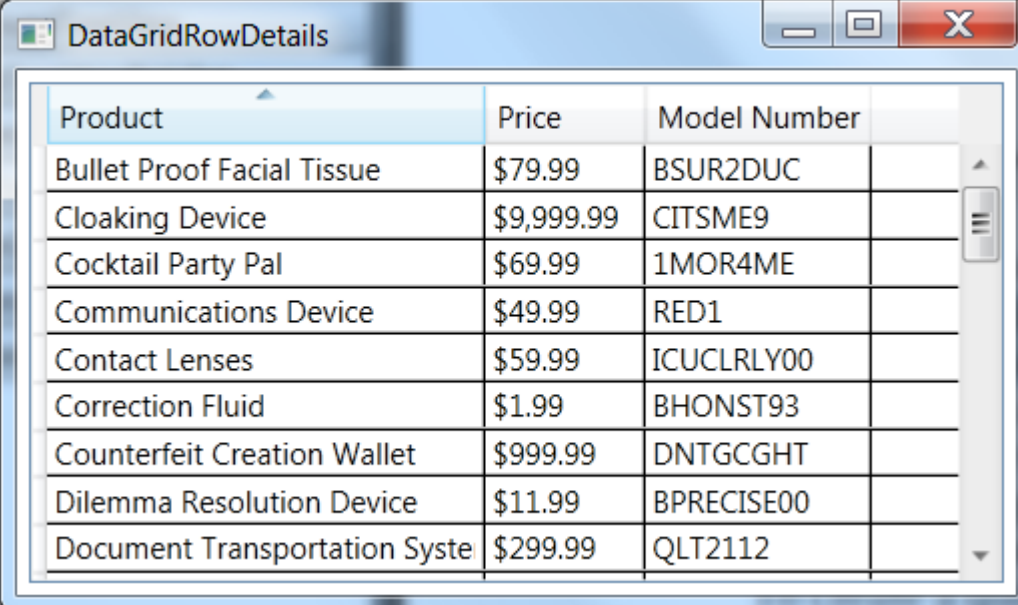
DataGrid

- The DataGrid is a data-display control that takes the information from a collection of objects and renders it in a grid of rows and cells
- Each row corresponds to a separate object, and each column corresponds to a property in that object
- **The user can edit the data!**

<Grid>

```
<DataGrid x:Name="gridProducts" AutoGenerateColumns="True" />
```

</Grid>



Product	Price	Model Number
Bullet Proof Facial Tissue	\$79.99	BSUR2DUC
Cloaking Device	\$9,999.99	CITSME9
Cocktail Party Pal	\$69.99	1MOR4ME
Communications Device	\$49.99	RED1
Contact Lenses	\$59.99	ICUCLRLY00
Correction Fluid	\$1.99	BHONST93
Counterfeit Creation Wallet	\$999.99	DNTGCGHT
Dilemma Resolution Device	\$11.99	BPRECISE00
Document Transportation Syste	\$299.99	QLT2112

Automatic Column Generation

- The DataGrid uses reflection to find every public property in the bound data object
 - It creates a column for each property
- To display nonstring properties, the DataGrid calls ToString()
 - Works well for numbers, dates, and other simple data types
- For more complex data types you may want to explicitly define your columns, which gives you the chance to:
 - Bind to a subproperty
 - Use a value converter
 - Apply a template

Columns

- The DataGrid control generates columns automatically when you set the ItemsSource property
 - You can turn this off with `AutoGenerateColumns="False"`
- The type of column that is generated depends on the type of data in the column:

Data Type	Generated Column Type
String	DataGridTextBoxColumn
Boolean	DataGridCheckBoxColumn
Enum	DataGridComboBoxColumn
Uri	DataGridHyperlinkColumn

- When columns are auto-generated, you can handle the `AutoGeneratingColumn` event to customize or cancel columns before they are added to the DataGrid
- If you add both user-defined columns and auto-generated columns to the DataGrid, the user-defined columns are added first
- To rearrange the display order of the columns, you can set the `DisplayIndex` property for individual columns

Defining Columns

- By setting `AutoGenerateColumns` to `false`, you can define the columns you want explicitly, with the settings you want and in the order you specify
- When you define a column, you almost always set three details: the header text that appears at the top of the column, the width of the column, and the binding that gets the data

```
<DataGrid Margin="5" AutoGenerateColumns="False">
  <DataGrid.Columns>
    <DataGridTextColumn Header="Product" Width="175"
                        Binding="{Binding Path=ModelName}">
    </DataGridTextColumn>
    <DataGridTextColumn Header="Price"
                        Binding="{Binding Path=UnitCost}">
    </DataGridTextColumn>
  </DataGrid.Columns>
</DataGrid>
```

Editing

- You put the current cell into edit mode by clicking it or pressing F2
- A cell-level edit is committed when you move to another cell in the same row or press ENTER while the cell is in edit mode
- All edits in a row are committed when you move to another row or press ENTER while the row is in edit mode
- You cancel a cell edit by pressing ESC one time, and cancel all edits in a row by pressing ESC two times
- Developer can set the CanUserAddRows and CanUserDeleteRows properties to specify whether a user can add or delete rows

Referances & Links

- DataGrid

[https://msdn.microsoft.com/en-us/library/system.windows.controls.datagrid\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.windows.controls.datagrid(v=vs.110).aspx)