PEF – 3528 – Ferramentas Computacionais na Mecânica das Estruturas: Criação e Concepção

Prof. Dr. Rodrigo Provasi

e-mail: provasi@usp.br

Sala 09 – LEM – Prédio de Engenharia Civil

Interfaces em C#

Applications

Iniciando uma aplicação em WPF

- Até o presente momento, conseguiu-se construir uma aplicação (*Window* ou *Page*) porém não fazer ele rodar diretamente.
- Isso é feito através do método show.

Iniciando uma aplicação em WPF

- De uma aplicação comum (Console) com um *main* é possível chamar a janela e exibi-la da seguinte forma:

```
public static void Main()
{
    MainWindow window = new MainWindow();
    window.Show();
}
```

Classe *Application*

- Uma outra maneira é utilizar a classe *Application:*

```
[STAThread]
public static void Main()
{
        Application app = new Application();
        MainWindow window = new MainWindow();
        window.Show();
        app.Run(window);
}
```

Classe *Application*

- Ou ainda:

```
[STAThread]
public static void Main()
{
        Application app = new Application();
        app.StartupUri = new Uri("MainWindow.xaml", UriKind.Relative);
        app.Run();
}
```

Classe *Application*

- Isso pode ser feito em um arquivo XAML (e é o que é feito pelo Visual Studio):

```
<Application x:Class="PhotoGallery.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="MainWindow.xaml"/>
```

Interfaces em C#

Recursos

Recursos

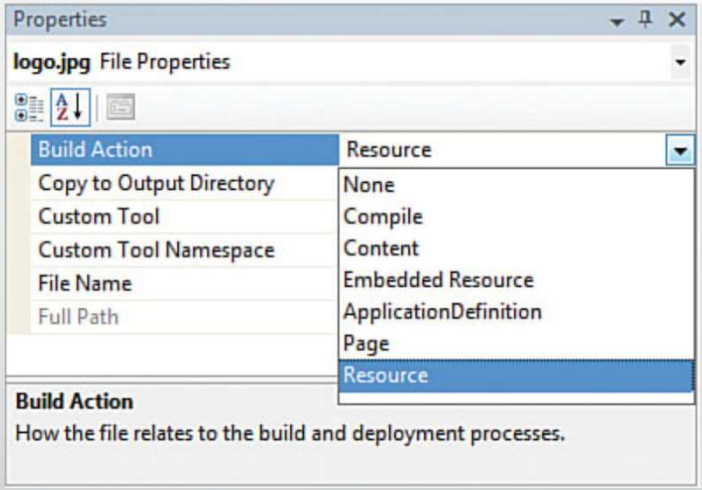
- Existem dois tipos de recursos: binários e lógicos.
- Binários são ligados à arquivos externos.
- Lógicos são ligados aos controles e suas propriedades.

Recursos Binários

- Quando se inclui um arquivo no Visual studio, pode-se escolher como ele será tratado:

- ***Resource:*** coloca o arquivo como um arquivo dentro do *assembly*.
- ***Content:*** deixa o arquivo solto, mas adiciona um atributo no *assembly* que grava a localização relativa do arquivo.

Recurso Binário



Acessando o recurso

```
<StackPanel Grid.Column="1" Orientation="Horizontal" HorizontalAlignment="Center">
        <Button x:Name="previousButton" ToolTip="Previous (Left Arrow)" ...>
                <Image Height="21" Source="previous.gif"/>
        </Button>
        <Button x:Name="slideshowButton" ToolTip="Play Slide Show (F11)" ...>
                <Image Height="21" Source="slideshow.gif"/>
        </Button>
        <Button x:Name="nextButton" ToolTip="Next (Right Arrow)" ...>
                <Image Height="21" Source="next.gif"/>
        </Button>
</StackPanel>
```

Acessando o recurso

Using `logo.jpg` as the Resource Name	
If the URI Is...	**The Resource Is...**
`logo.jpg`	▶ Embedded in the current assembly, or ▶ Loose and alongside the current XAML page or assembly (the latter case only if marked as `Content` in the project)
`A/B/logo.jpg`	▶ Embedded in the current assembly using an internal subfolder (`A\B`) structure defined at compile time, or ▶ Loose and in an `A\B` subfolder relative to the current XAML page or assembly (the latter case only if marked as `Content` in the project)
`c:\temp\logo.jpg`	Loose in the local `c:\temp` folder
`file://c:/temp/logo.jpg`	Loose in the local `c:\temp` folder
`\\pc1\images\logo.jpg`	Loose on the `\\pc1\images` UNC share
`http://pinvoke.net/logo.jpg`	Loose and hosted at the pinvoke.net website
`MyDll;Component/logo.jpg`	Embedded in a different assembly called `MyDll.dll` or `MyDll.exe`
`MyDll;Component/A/B/logo.jpg`	Embedded in a different assembly called `MyDll.dll` or `MyDll.exe` using an internal subfolder structure (`A\B`) defined at compile time
`pack://siteOfOrigin:,,,/logo.jpg`	Loose at the site of origin
`pack://siteOfOrigin:,,,/A/B/logo.jpg`	Loose at the site of origin in an `A\B` subfolder

Recursos Lógicos

- São recursos na linguagem do XAML (são dicionários com características para serem usadas no arquivo).
- Eles são armazenados num elemento *Resource.*

Recursos

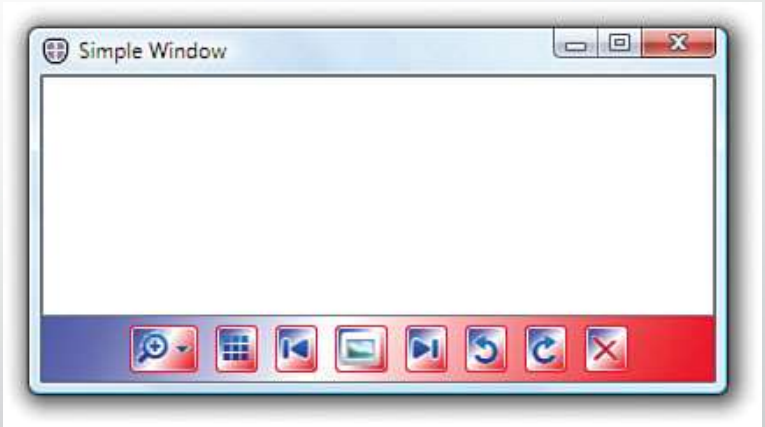
```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Title="Simple Window" Background="Yellow">
        <DockPanel>
                <StackPanel DockPanel.Dock="Bottom" Orientation="Horizontal"
HorizontalAlignment="Center">
                <Button Background="Yellow" BorderBrush="Red" Margin="5">
                        <Image Height="21" Source="zoom.gif"/>
                </Button>
                <Button Background="Yellow" BorderBrush="Red" Margin="5">
                        <Image Height="21" Source="defaultThumbnailSize.gif"/>
                </Button>
                <Button Background="Yellow" BorderBrush="Red" Margin="5">
                        <Image Height="21" Source="previous.gif"/>
                </Button>
                <Button Background="Yellow" BorderBrush="Red" Margin="5">
                        <Image Height="21" Source="slideshow.gif"/>
                </Button>
                <Button Background="Yellow" BorderBrush="Red" Margin="5">
                        <Image Height="21" Source="next.gif"/>
                </Button>
                <Button Background="Yellow" BorderBrush="Red" Margin="5">
                        <Image Height="21" Source="counterclockwise.gif"/>
                </Button>
                <Button Background="Yellow" BorderBrush="Red" Margin="5">
                        <Image Height="21" Source="clockwise.gif"/>
                </Button>
                <Button Background="Yellow" BorderBrush="Red" Margin="5">
                        <Image Height="21" Source="delete.gif"/>
                </Button>
                </StackPanel>
        <ListBox/>
        </DockPanel>
</Window>
```

Recursos

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Simple Window">
    <Window.Resources>
        <SolidColorBrush x:Key="backgroundBrush">Yellow</SolidColorBrush>
        <SolidColorBrush x:Key="borderBrush">Red</SolidColorBrush>
    </Window.Resources>
    <Window.Background>
        <StaticResource ResourceKey="backgroundBrush"/>
    </Window.Background>
    <DockPanel>
        <StackPanel DockPanel.Dock="Bottom" Orientation="Horizontal" HorizontalAlignment="Center">
            <Button Background="{StaticResource backgroundBrush}" BorderBrush="{StaticResource borderBrush}" Margin="5">
                <Image Height="21" Source="zoom.gif"/>
            </Button>
            <Button Background="{StaticResource backgroundBrush}" BorderBrush="{StaticResource borderBrush}" Margin="5">
                <Image Height="21" Source="defaultThumbnailSize.gif"/>
            </Button>
            <Button Background="{StaticResource backgroundBrush}" BorderBrush="{StaticResource borderBrush}" Margin="5">
                <Image Height="21" Source="previous.gif"/>
            </Button>
            <Button Background="{StaticResource backgroundBrush}" BorderBrush="{StaticResource borderBrush}" Margin="5">
                <Image Height="21" Source="slideshow.gif"/>
            </Button>
            <Button Background="{StaticResource backgroundBrush}" BorderBrush="{StaticResource borderBrush}" Margin="5">
                <Image Height="21" Source="next.gif"/>
            </Button>
            <Button Background="{StaticResource backgroundBrush}" BorderBrush="{StaticResource borderBrush}" Margin="5">
                <Image Height="21" Source="counterclockwise.gif"/>
            </Button>
            <Button Background="{StaticResource backgroundBrush}" BorderBrush="{StaticResource borderBrush}" Margin="5">
                <Image Height="21" Source="clockwise.gif"/>
            </Button>
            <Button Background="{StaticResource backgroundBrush}" BorderBrush="{StaticResource borderBrush}" Margin="5">
                <Image Height="21" Source="delete.gif"/>
            </Button>
        </StackPanel>
        <ListBox/>
    </DockPanel>
</Window>
```

Recursos

```
<LinearGradientBrush x:Key="backgroundBrush"
StartPoint="0,0" EndPoint="1,1">
    <GradientStop Color="Blue" Offset="0"/>
    <GradientStop Color="White" Offset="0.5"/>
    <GradientStop Color="Red" Offset="1"/>
</LinearGradientBrush>
```



Recursos Estáticos e Dinâmicos

- Existem duas maneiras de usar um recurso. Com a palavra chave *StaticResource* como anteriormente visto e também com a palavra chave *DynamicResource*.

- A diferença é que a segunda palavra chave permite que o recurso utilizado pode ser modificado e essa modificação seja refletida em todos os elementos que a ele foram associados.

Recursos Estáticos e Dinâmicos

- Recursos estáticos são mais lentos para carregar ao passo que os dinâmicos tem um maior *overhead* de processamento.

- Recursos dinâmicos só podem ser aplicados em *Dependency Properties*, ao passo que os estáticos podem ser aplicados praticamente em todos os lugares.

Dicionários

- É possível criar um arquivo apenas para armazenar os recursos. Isso é conhecido como *Resource Dictionary*.

```
<Window.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary Source="file1.xaml"/>
            <ResourceDictionary Source="file2.xaml"/>
        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</Window.Resources>
```

- Em um outro arquivo (por exemplo, file1.xaml):

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Image x:Key="logo" Source="logo.jpg"/>
</ResourceDictionary>
```

Interfaces em C#

Data Binding

Data Binding

- *Binding* pode ser entendido como uma ligação entre duas propriedades de dois objetos.
- Imagine um *TextBlock* e *TreeView*. Deseja se exibir no texto o nome do item selecionado na árvore.

Binding

<TextBlock x:Name="currentFolder" DockPanel.Dock="Top" Background="AliceBlue" FontSize="16" />

- No código:

```
void treeView_SelectedItemChanged(object sender, RoutedPropertyChangedEventArgs<object> e)
{
    currentFolder.Text = (treeView.SelectedItem as TreeViewItem).Header.ToString();
    Refresh();
}
```

Binding

- Isso pode ser feito via *binding*, linkando o texto do *TextBlock* com o item selecionado da *TreeView*.

```
public MainWindow()
{
        InitializeComponent();
        Binding binding = new Binding();
        binding.Source = treeView; // Set source object
        binding.Path = new PropertyPath("SelectedItem.Header"); // Set source property
        currentFolder.SetBinding(TextBlock.TextProperty, binding); // Attach to target property
}
```

Binding

- Essa ligação pode ser feita diretamente no *markup*:

```
<TextBlock x:Name="currentFolder" DockPanel.Dock="Top" Text="{Binding
    ElementName=treeView, Path=SelectedItem.Header}"
    Background="AliceBlue" FontSize="16" />
```

Binding

- É possível escrever passando um *string* como parâmetro inicial do *binding*:

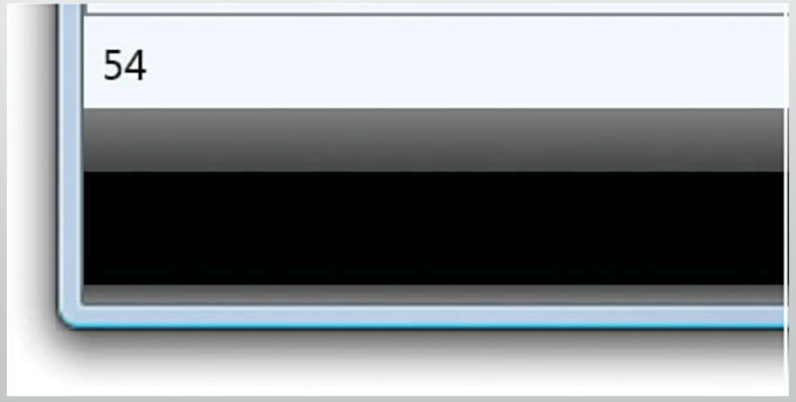
<TextBlock x:Name="currentFolder" DockPanel.Dock="Top"

Text="{Binding SelectedItem.Header, ElementName=treeView}"
Background="AliceBlue" FontSize="16" />

Binding

- Imagine uma coleção de fotos (como um recurso estático). É possível utilizar o seguinte *markup*:

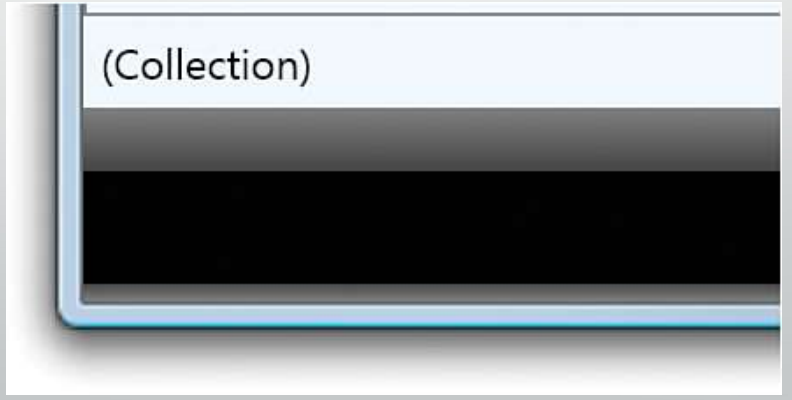
<Label x:Name="numItemsLabel" **Content="{Binding Source={StaticResource photos}, Path=Count}"** DockPanel.Dock="Bottom"/>



Binding

- É possível associar ao conteúdo como completo:

- <Label x:Name="numItemsLabel" **Content="{Binding Source={StaticResource photos}}"** DockPanel.Dock="Bottom"/>



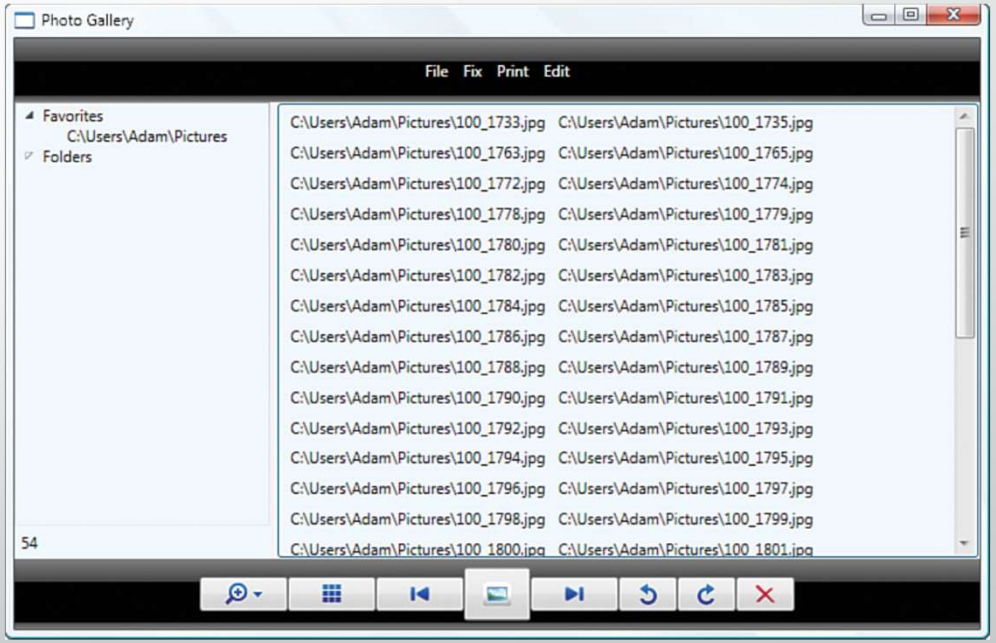
Binding para coleções

- É possível direcionar direto para uma coleção:

<ListBox x:Name="pictureBox"
ItemsSource="{Binding Source={StaticResource photos}}" ...>

...

</ListBox>



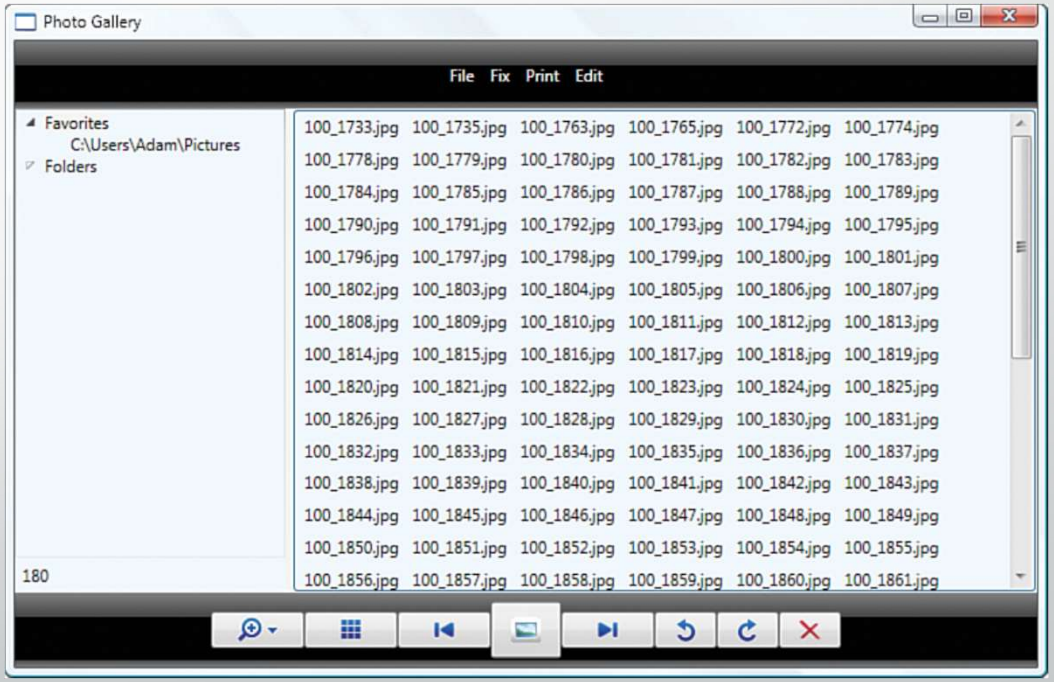
Binding para coleções

- É possível melhorar a exibição:

<ListBox x:Name="pictureBox"
DisplayMemberPath="Name"
ItemsSource="{Binding Source={StaticResource photos}}" ...>

...

</ListBox>



Binding com coleções

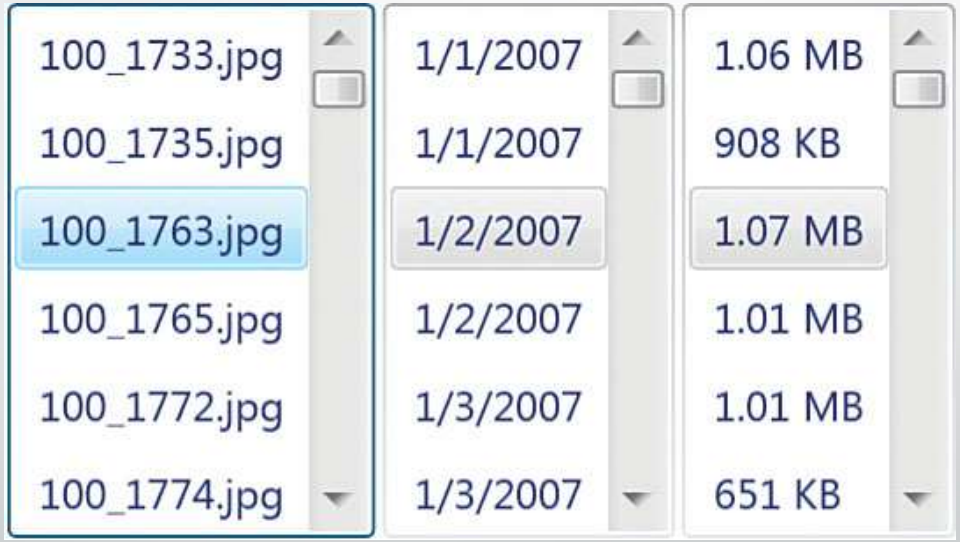
- É possível também sincronizar usando a mesma coleção:

<ListBox **IsSynchronizedWithCurrentItem="True"** DisplayMemberPath="Name" ItemsSource="{Binding Source={StaticResource photos}}"></ListBox>

<ListBox **IsSynchronizedWithCurrentItem="True"** DisplayMemberPath="DateTime" ItemsSource="{Binding Source={StaticResource photos}}"></ListBox>

<ListBox **IsSynchronizedWithCurrentItem="True"** DisplayMemberPath="Size" ItemsSource="{Binding Source={StaticResource photos}}"></ListBox>

Binding com coleções



Interfaces em C#

Data Grid

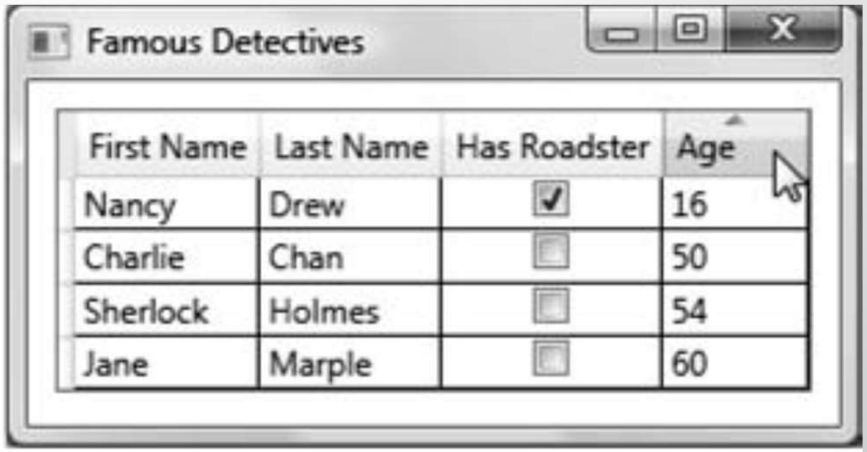
Data Grid

- O data grid é um controle que coloca os dados na forma de uma tabela.

- Eles permitem :
  - ***Sorting***: You can programmatically sort the rows on a particular column. The user can sort on a column by clicking its header.
  - ***Column headers***: You can display just column headers, just row headers, or both.
  - ***Rearrange columns***: You can rearrange the columns programmatically, or the user can rearrange them by dragging the headers left or right.
  - ***Specialized cell types***: The grid supplies specialized column types for text, Buttons, CheckBoxes, ComboBoxes, and Hyperlinks.
  - ***Customized appearance***: You can attach Styles and Templates to the DataGrid, as well as to most of its components. This gives you a large number of options for customizing the grid's appearance.

Data Grid

Initial Display



Sorted on Age

Data Grid

```
<DataGrid Name="dg" AutoGenerateColumns="False">   ← Explicitly turn off autogeneration.
  <DataGrid.Columns>
    <DataGridTextColumn Binding="{Binding FirstName}" Header="First Name"/>
    <DataGridTextColumn Binding="{Binding LastName}"  Header="Last Name"/>
```

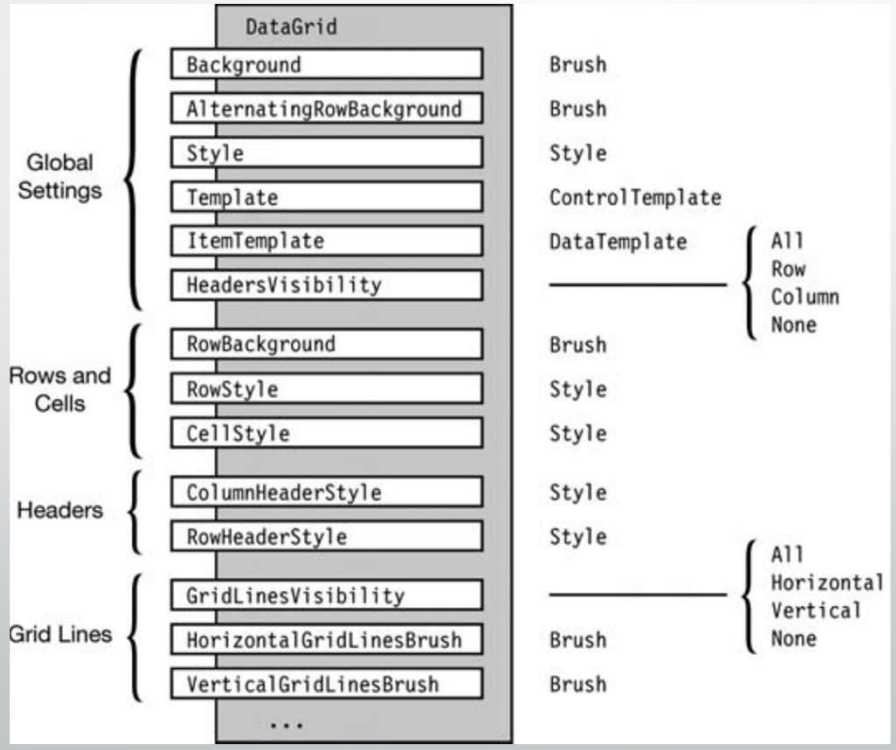
Use the correct type of colum. Bind to the data field. Set the text of the header.

```
<StackPanel>
  <DataGrid Name="dg" AutoGenerateColumns="False" Margin="10">
    <DataGrid.Columns>
      <DataGridTextColumn Binding="{Binding FirstName}"
                          Header="First Name"/>
      <DataGridTextColumn Binding="{Binding LastName}"
                          Header="Last Name"/>
      <DataGridCheckBoxColumn Binding="{Binding HasRoadster}"
                          Header="Has Roadster"
                          Width="SizeToHeader"/>          ← Width
      <DataGridTextColumn Binding="{Binding Age}"
                          Header="Age"
                          Width="*"/>                     ← Width
    </DataGrid.Columns>
  </DataGrid>
</StackPanel>
```

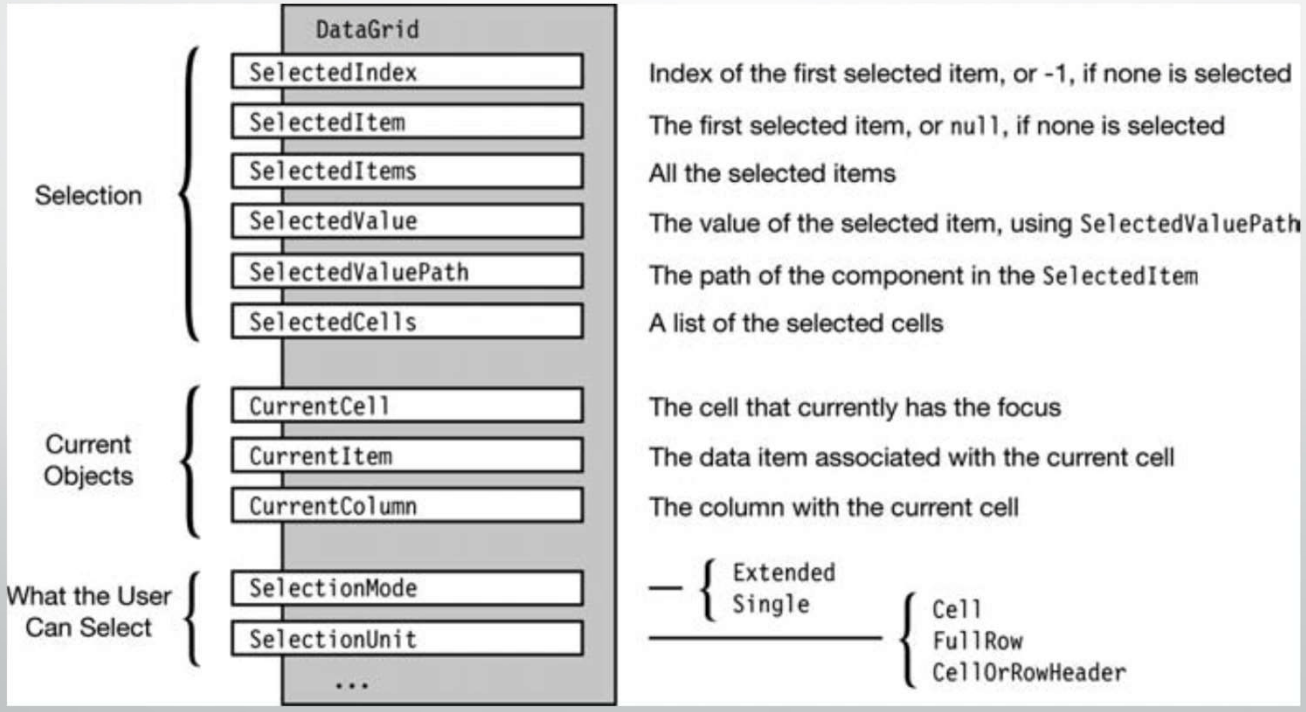
Data Grid

```
class Person
{
   public string FirstName   { get; set; }
   public string LastName    { get; set; }
   public int    Age         { get; set; }
   public bool   HasRoadster { get; set; }

   public Person(string fName, string lName, int age, bool hasRoadster)
   {
       FirstName   = fName;
       LastName    = lName;
       Age         = age;
       HasRoadster = hasRoadster;
   }
}

public partial class Window1 : Window
{
   List<Person>  people = new List<Person>();

   public Window1()
   {  InitializeComponent();

       people.Add(new Person("Sherlock", "Holmes", 54, false));
       people.Add(new Person("Jane",     "Marple", 60, false));
       people.Add(new Person("Nancy",    "Drew",   16, true));
       people.Add(new Person("Charlie",  "Chan",   50, false));

      dg.ItemsSource =  people;
   }
}
```

Data Grid


DataGrid
Global Settings
Background
AlternatingRowBackground
Style
Template
ItemTemplate
HeadersVisibility
Rows and Cells
RowBackground
RowStyle
CellStyle
Headers
ColumnHeaderStyle
RowHeaderStyle
Grid Lines
GridLinesVisibility
HorizontalGridLinesBrush
VerticalGridLinesBrush
...
Brush
Brush
Style
ControlTemplate
DataTemplate
All
Row
Column
None
Brush
Style
Style
Style
Style
All
Horizontal
Vertical
None
Brush
Brush

Data Grid


DataGrid
Selection
SelectedIndex
SelectedItem
SelectedItems
SelectedValue
SelectedValuePath
SelectedCells
Current Objects
CurrentCell
CurrentItem
CurrentColumn
What the User Can Select
SelectionMode
SelectionUnit
...
Index of the first selected item, or -1, if none is selected
The first selected item, or null, if none is selected
All the selected items
The value of the selected item, using SelectedValuePath
The path of the component in the SelectedItem
A list of the selected cells
The cell that currently has the focus
The data item associated with the current cell
The column with the current cell
Extended
Single
Cell
FullRow
CellOrRowHeader

Tópicos em C#

Arquivos

Arquivos

- Existem vários possíveis formatos para arquivos a serem utilizados no C#, como por exemplo:
  - Binários
  - Json
  - Texto puro (TXT)

Arquivos binários

- Arquivos binários armazenam os bytes relativos às variáveis em questão.
- Armazenam *doubles*, *ints*, *chars*, *strings*, entre outros.
- É necessário escrever uma função de leitura e outra de gravação para cada objeto.
- A ordem em que essa gravação é feita é importante.

Arquivos binários

```
if (File.Exists(FILE_NAME))
{
        Console.WriteLine("{0} already exists!", FILE_NAME);
        return;
}
using (FileStream fs = new FileStream(FILE_NAME, FileMode.CreateNew))
 {
        using (BinaryWriter w = new BinaryWriter(fs))
        {
            for (int i = 0; i < 11; i++)
            {
                    w.Write(i);
             }
```

```
        }
}
using (FileStream fs = new FileStream(FILE_NAME, FileMode.Open, FileAccess.Read))
    {
      using (BinaryReader r = new BinaryReader(fs))
      {
        for (int i = 0; i < 11; i++)
        {
          Console.WriteLine(r.ReadInt32());
        }
      }
    }
```

Arquivos de texto

- Arquivos de texto são a forma mais comum de entrada.
- A leitura é feita item a item ou linha a linha e processado de acordo.
- Pode-se fazer uma lógica para permitir que os itens estejam fora de ordem (mas com restrições).

Arquivos de texto

- Escrevendo um arquivo:

```
string[] lines = { "First line", "Second line", "Third line" }; // Create a string array with the lines of text

// Set a variable to the My Documents path.
string mydocpath =    Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

// Write the string array to a new file named "WriteLines.txt".
using (StreamWriter outputFile = new StreamWriter(Path.Combine(mydocpath,"WriteLines.txt")))
{
        foreach (string line in lines)
                outputFile.WriteLine(line);
}
```

Arquivos de texto

- Lendo um arquivo:

```
public static void Main()
  {
    try
    {
      // Open the text file using a stream reader.
      using (StreamReader sr = new StreamReader("TestFile.txt"))
      {
            // Read the stream to a string, and write the string to the console.
            String line = sr.ReadToEnd();
            Console.WriteLine(line);
      }
    }
    catch (Exception e)
    {
      Console.WriteLine("The file could not be read:");
      Console.WriteLine(e.Message);
    }
  }
```

Arquivos JSon

- Um arquivo JSon é um arquivo de texto organizado na forma de dicionário onde há uma chave e um valor que pode ser um valor simples (double, int, string...), uma lista ou outro dicionário.

Arquivos Json (usando biblioteca)

- Leitura:

```
JObject o1 =JObject.Parse(File.ReadAllText(@"c:\videogames.json"));
// read JSON directly from a file
using (StreamReader file = File.OpenText(@"c:\videogames.json"))
    using (JsonTextReader reader = new JsonTextReader(file))
    {
        JObject o2 = (JObject)JToken.ReadFrom(reader);
    }
```

Arquivos Json (usando biblioteca)

- Gravação:

```
JObject videogameRatings = new JObject(    new JProperty("Halo", 9),
                                           new JProperty("Starcraft", 9),
                                           new JProperty("Call of Duty", 7.5));
File.WriteAllText(@"c:\videogames.json", videogameRatings.ToString());

// write JSON directly to a file
using (StreamWriter file = File.CreateText(@"c:\videogames.json"))
        using (JsonTextWriter writer = new JsonTextWriter(file))
        {
                videogameRatings.WriteTo(writer);
        }
```