Advanced Programming 2
Recitation 5 – WPF Part II

Roi Yehoshua
2017

Resources and Styles

WPF Resources

- Arbitrary named .NET objects, stored in the `Resources` collection property of an element
  - Typically for sharing the resource among child objects
- Add a `Resources` property to some element
  - Usually a `Window` or the `Application`
  - Any child element can reference those resources
- Add the objects with a `x:Key` attribute (must be unique in this resource dictionary)
- Use the `StaticResource` markup extension with the resource key name

Resources Example

```
<StackPanel Margin="4" Orientation="Horizontal" TextBlock.FontSize="20">
        <Button Margin="4" Content="1" Padding="4">
            <Button.Background>
                <LinearGradientBrush>
                    <GradientStop Offset="0" Color="Blue" />
                    <GradientStop Offset="1" Color="Yellow" />
                </LinearGradientBrush>
            </Button.Background>
        </Button>
        <Button Margin="4" Content="2" Padding="4">
            <Button.Background>
                <LinearGradientBrush>
                    <GradientStop Offset="0" Color="Blue" />
                    <GradientStop Offset="1" Color="Yellow" />
                </LinearGradientBrush>
            </Button.Background>
        </Button>
        <Button Margin
            <Button.Ba
                <Linea
                    <G
                    <G
                </Linea
            </Button.Ba
        </Button>
</StackPanel>
```


```
<StackPanel Margin="4" Orientation="Horizontal" TextBlock.FontSize="20">
      <StackPanel.Resources>
          <LinearGradientBrush x:Key="back">
              <GradientStop Offset="0" Color="Blue" />
              <GradientStop Offset="1" Color="Yellow" />
          </LinearGradientBrush>
      </StackPanel.Resources>
      <Button Margin="4" Content="1" Padding="4" Background="{StaticResource back}" />
      <Button Margin="4" Content="2" Padding="4" Background="{StaticResource back}" />
      <Button Margin="4" Content="3" Padding="4" Background="{StaticResource back}" />
</StackPanel>
```

Styles

- A style is a collection of dependency property setters (and triggers)
- Usually defined as a resource
- Can be applied to any element with the `Style` property
- Any specific property changed by the element that conflicts with the style takes precedence over the style setting

Style Example

```
<Window x:Class="Demo.StyleWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Style Demo" SizeToContent="WidthAndHeight">
    <Window.Resources>
        <Style x:Key="fancyButton">
            <Setter Property="Button.Background" Value="Purple" />
            <Setter Property="Button.FontSize" Value="20" />
            <Setter Property="Button.FontWeight" Value="Bold" />
            <Setter Property="Button.Foreground" Value="Yellow" />
            <Setter Property="Button.Margin" Value="4" />
            <Setter Property="Button.LayoutTransform">
                <Setter.Value>
                    <RotateTransform Angle="15" />
                </Setter.Value>
            </Setter>
        </Style>
    </Window.Resources>
    <StackPanel Margin="4">
        <Button Content="Click Me" Style="{StaticResource fancyButton}" />
        <Button Content="Hello" Style="{StaticResource fancyButton}" />
        <Button Content="Me Fancy" Style="{StaticResource fancyButton}" />
    </StackPanel>
</Window>
```

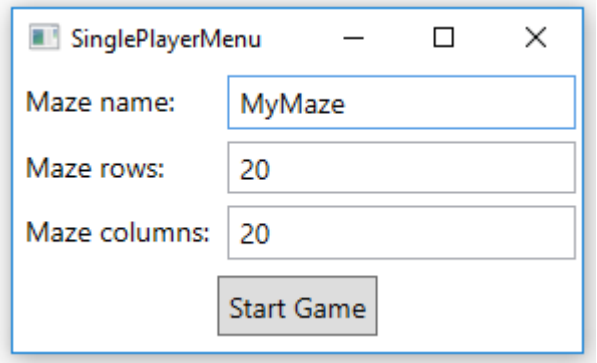
Restricting Style Usage

- A style may be restricted to work on certain types using the `TargetType` property
- In this case, the full property qualification is unnecessary
- With the `TargetType` property the **x:Key** attribute may be omitted
- The style will automatically apply to all elements of the specified type
- Elements can still override properties or use a different style, or clear the style by setting it to `null`

```
<Style x:Key="fancyButton" TargetType="{x:Type Button}">
   <Setter Property="Background" Value="Purple" />
   <Setter Property="FontSize" Value="20" />
   <Setter Property="FontWeight" Value="Bold" />
   <Setter Property="Foreground" Value="Yellow" />
   <Setter Property="Margin" Value="4" />
   <Setter Property="LayoutTransform">
      <Setter.Value>
         <RotateTransform Angle="15" />
      </Setter.Value>
   </Setter>
</Style>
```

Grid With Style

```
<Grid TextBlock.FontSize="14">
    <Grid.RowDefinitions>
        …
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        …
    </Grid.ColumnDefinitions>
    <Grid.Resources>
        <Style TargetType="TextBlock">
            <Setter Property="Padding" Value="3"/>
            <Setter Property="Margin" Value="3"/>
        </Style>
        <Style TargetType="TextBox">
            <Setter Property="Padding" Value="3"/>
            <Setter Property="Margin" Value="3"/>
        </Style>
    </Grid.Resources>
    <TextBlock>Maze name:</TextBlock>
    <TextBox x:Name="txtMazeName" Grid.Column="2" Text="{Binding MazeName}"></TextBox>
    <TextBlock Grid.Row="1">Maze rows:</TextBlock>
    <TextBox x:Name="txtRows" Grid.Row="1" Grid.Column="2" Text="{Binding MazeRows}"></TextBox>
    <TextBlock Grid.Row="2">Maze columns:</TextBlock>
    <TextBox x:Name="txtCols" Grid.Row="2" Grid.Column="2" Text="{Binding MazeCols}"></TextBox>
    <Button x:Name="btnStart" Grid.Row="3" Grid.ColumnSpan="2" HorizontalAlignment="Center" Margin="5"
Padding="5" Click="btnStart_Click">Start Game</Button>
</Grid>
```



Application and Windows

Roi Yehoshua, Bar Ilan University

The Application Object

- Every WPF application is represented by an instance of **`System.Windows.Application`**
  - A singleton (per AppDomain)
- The static property **`Application.Current`** returns that instance
- The Visual Studio wizard creates a XAML file App.xaml for the application
  - Useful for application level resources
  - Also can set the **`StartupUri`** property to indicate which window to show on startup

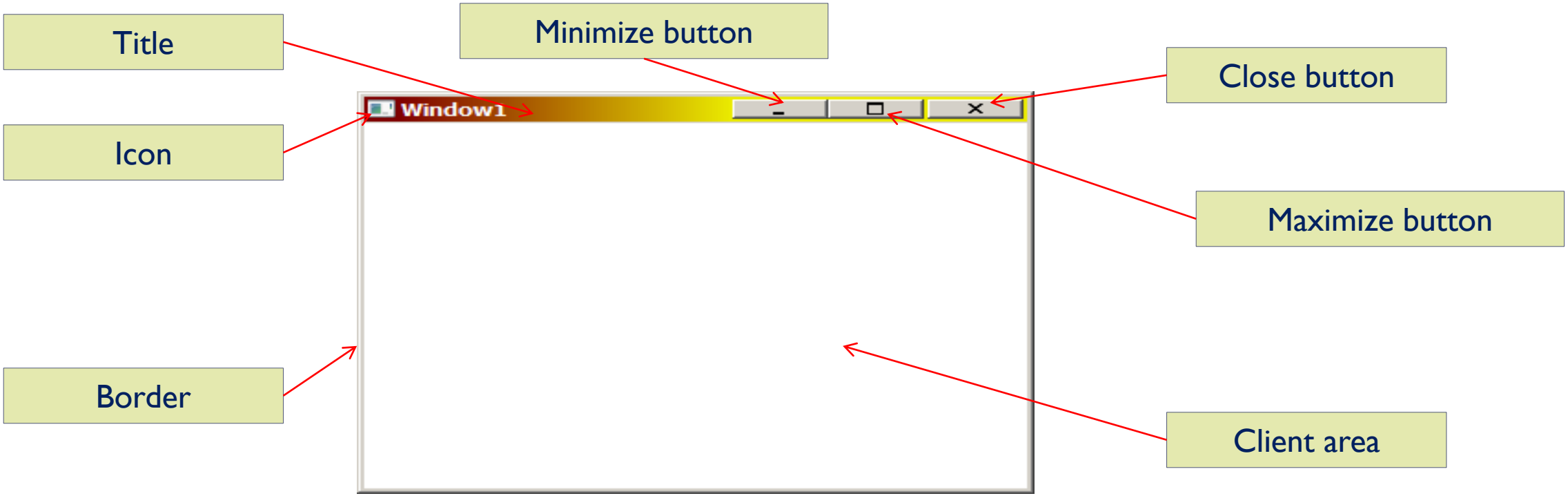
```
<Application x:Class="MazeGUI.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:MazeGUI"
             StartupUri="MainWindow.xaml">
    <Application.Resources>
    </Application.Resources>
</Application>
```

Top Level Windows

- A top level window is not contained and not owned by another window
- By default, the first top level window created is designated the main window (accessed with the `Application.MainWindow` property)
  - Can change the main window programmatically
- A collection of all top level windows is maintained in the `Application.Windows` property

The `Window` Class

- Technically, a content control
  - Content is typically a layout panel

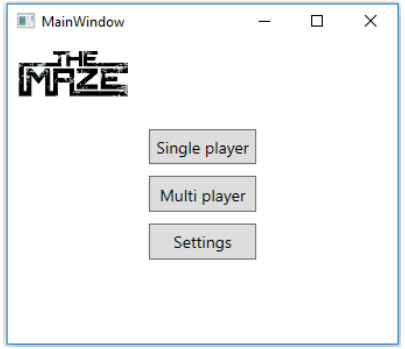


Basic Window Properties

- `Title`
  - The title (caption) of the window
- `ResizeMode` (of type enum `ResizeMode`)
  - Controls whether the user can resize the window
  - Also affects the visibility of the Maximize and Minimize buttons
- `WindowStartupLocation`
  - Sets the initial location of the window
  - `Manual` – set explicitly using the `Top` and `Left` properties
  - `CenterScreen` – centered on screen
  - `CenterOwner` – centered with respect to the owner window (must set the `Owner` property to non-null)
- `SizeToContent` (of type enum `SizeToContent`)
  - Allows the window to change size based on its content
  - `Manual`, `Width`, `Height`, `WidthAndHeight`

Showing a Window

- Modeless window
  - Call the **`Show`** method
- Modal window (usually a dialog box)
  - Call the **`ShowDialog`** window
  - Cannot access other windows until the modal window is dismissed
- Closing a window
  - Call the **`Close`** method
- Hiding a window
  - Call the **`Hide`** method
  - Or change the **`Visibility`** property to **`Hidden`**
  - Only makes sense for modeless windows




```
private void btnSinglePlayer_Click(object sender,
RoutedEventArgs e)
{
    SinglePlayerMenu menu = new SinglePlayerMenu();
    menu.ShowDialog();
}
```

WPF User Control

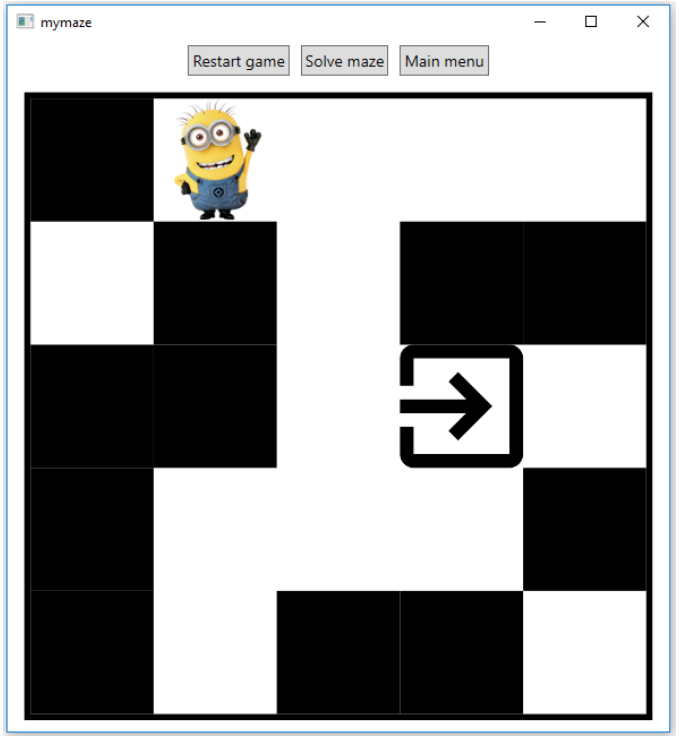
- Allows you to design a reusable component that can be added to a windos just like a system control
- User controls can contain other controls, resources, and animation timelines, just like a WPF window
- The only difference is that the root element is a **UserControl** instead of a **Window** or a **Page**
- Typically user control exposes its properties as **dependency properties** to enable binding them to other objects (and also to styles, animation, etc.)

User Control Example

```
<UserControl x:Class="MazeGUI.Controls.MazeBoard"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300" Loaded="UserControl_Loaded"
    <Viewbox>
        <Border BorderBrush="Black" BorderThickness="3">
            <Canvas x:Name="myCanvas" Width="300" Height="300">
            </Canvas>
        </Border>
    </Viewbox>
</UserControl>
```

```
<Window x:Class="MazeGUI.SinglePlayerWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:controls="clr-namespace:MazeGUI.Controls"
        mc:Ignorable="d" Height="650" Width="600"
WindowStartupLocation="CenterScreen“>
    <Grid>
        …

        <controls:MazeBoard x:Name="mazeBoard" Rows="5" Cols="5"
Maze="1,0,1,0,0,0,1,0,1,1,1,1,1,0,0,1,0,0,0,1,1,0,1,1,0" InitialPos="0,1"
GoalPos="2,3" PlayerImageFile="minion.png" ExitImageFile="exit.png" Grid.Row="1"
Margin="10" ></controls:MazeBoard>
    </Grid>
</Window>
```



Note the XAML namespace mapping

Dependency Property in User Control

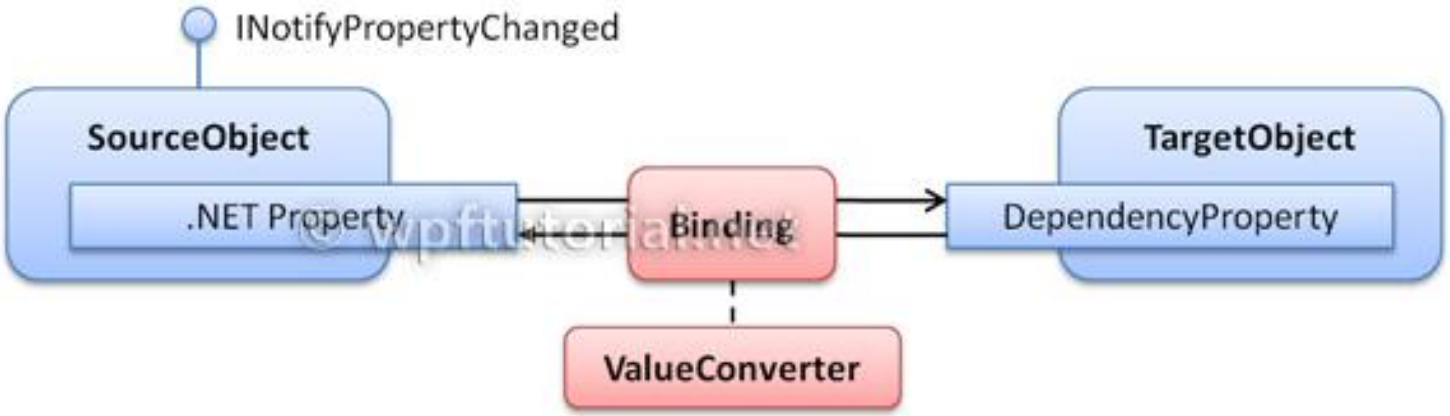
```
public int Rows
{
    get { return (int)GetValue(RowsProperty); }
    set { SetValue(RowsProperty, value); }
}

// Using a DependencyProperty as the backing store for Rows.  This enables animation, styling,
binding, etc...
public static readonly DependencyProperty RowsProperty =
    DependencyProperty.Register("Rows", typeof(int), typeof(MazeBoard), new
PropertyMetadata(0));
```

Data Binding

What is Data Binding?

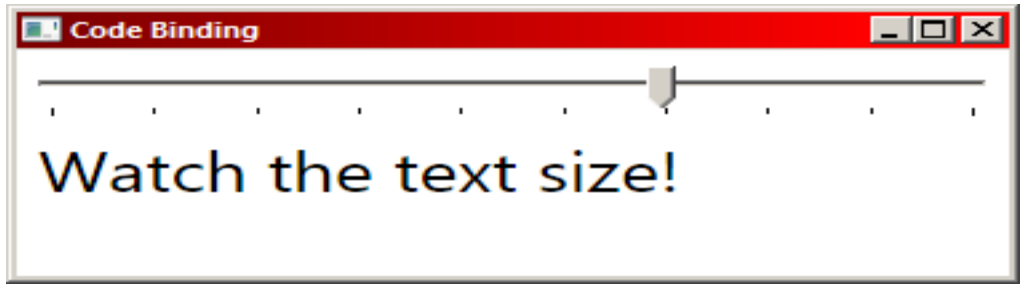
- Data in WPF can reference any .NET object
- Data binding means tying two arbitrary objects
- Typical scenario is a data object (or collection) to a visual element
- Any changes to the data object are reflected in the visual element (and optionally vice versa)



Data Binding With XAML

- Use the Binding markup extension
- Implemented by the **`Binding`** class
- For source elements, use the **`ElementName`** property
- Use the **`Path`** property to specify the source property to bind to

```
<StackPanel Margin="4" Orientation="Vertical" >
    <Slider x:Name="sizeSlider" Minimum="4" TickPlacement="BottomRight“
     TickFrequency="4" Maximum="40" Margin="4" Value="10"/>
    <TextBlock Text="Watch the text size!" Margin="4"
     FontSize="{Binding ElementName=sizeSlider, Path=Value}"/>
</StackPanel>
```



Binding Direction

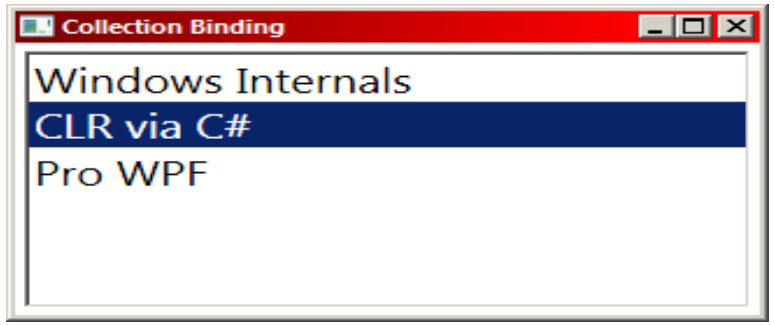
- The **Binding** object allows specifying how the target/source properties are updated
- **Mode** property (of type enum **BindingMode**)

Binding Mode	Meaning
`OneWay`	The target property is updated by source property changes
`TwoWay`	OneWay + the source property is updated by changes of the target property
`OneWayToSource`	Similar to OneWay, but from target to source For source properties that are not dependency properties
`OneTime`	Target is updated by the source the first time they are bound
`Default`	Depends on the target property. TwoWay for user settable properties, OneWay for everything else. This is the default value

Binding to a Collection

- Collections (implementing `IEnumerable`, etc.) can be bound to elements that naturally hold collections of items
  - E.g. `ListBox`
- The **`ItemsSource`** property of the base class **`ItemsControl`** can be used for binding collections
  - The **`DisplayMemeberPath`** property can be used to indicate what property to render for each item in the collection
  - The **`Items`** property cannot be used in conjunction with **`ItemsSource`**

Binding to a Collection Example




```
namespace Demo {
    class Book {
        public string Name { get; set; }
        public decimal Price { get; set; }
        public string Author { get; set; }
    }

    class Books : List<Book> {
    }
}
```

```
<Window x:Class="Demo.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:Demo"
    Title="Collection Binding" Width="350" Height="200" FontSize="20">
    <Window.Resources>
        <local:Books x:Key="books">
            <local:Book Name="Windows Internals" Price="40" Author="Mark Russinovich" />
            <local:Book Name="CLR via C#" Price="30" Author="Jeffrey Richter" />
            <local:Book Name="Pro WPF" Price="50" Author="Matthew MacDonald" />
        </local:Books>
    </Window.Resources>
    <ListBox Margin="4" ItemsSource="{StaticResource books}"
     DisplayMemberPath="Name" />
</Window>
```