Advanced Programming 2
Recitation 6 – WPF Part III

Roi Yehoshua
2017

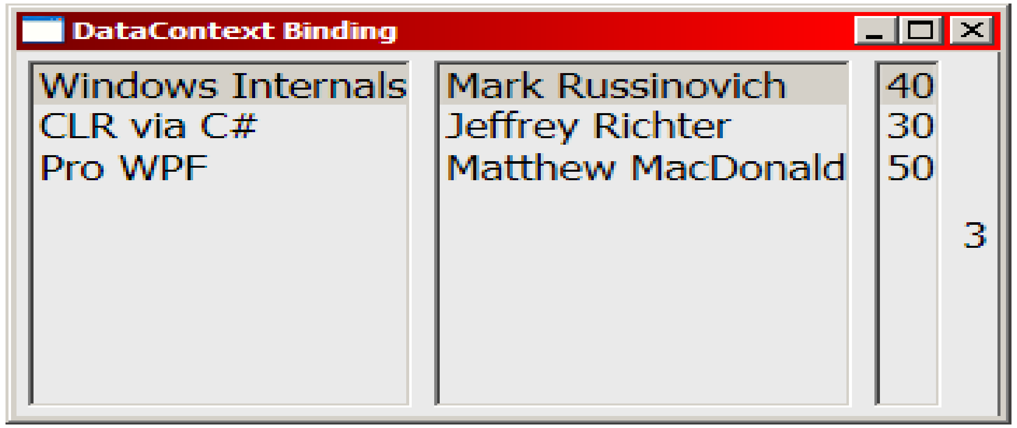
Data Binding (cont.)

The `DataContext`

- Sometimes many elements bind to the same object
  - Perhaps with different properties
- The object may be specified as the **`DataContext`** property on any common parent element
- Whenever the **`Source`** or **`RelativeSource`** properties are not specified in the **`Binding`**, a data context object is searched up the element hierarchy
  - If found, becomes the binding source object
- Can be used programmatically without the need to create the source object in XAML

DataContext Example

```
<StackPanel Orientation="Horizontal" DataContext="{StaticResource books}">
  <ListBox IsSynchronizedWithCurrentItem="True" Margin="4"
     ItemsSource="{Binding}" DisplayMemberPath="Name" />
  <ListBox IsSynchronizedWithCurrentItem="True" Margin="4"
     ItemsSource="{Binding}" DisplayMemberPath="Author" />
  <ListBox IsSynchronizedWithCurrentItem="True" Margin="4"
     ItemsSource="{Binding}" DisplayMemberPath="Price" />
  <TextBlock Margin="4" VerticalAlignment="Center" Text="{Binding Path=Count}" />
</StackPanel>
```



```
DataContext Binding

Windows Internals   Mark Russinovich    40
CLR via C#          Jeffrey Richter     30
Pro WPF             Matthew MacDonald   50
                                            3
```

Change Notifications

- An object must notify when one of its properties changes
  - By implementing the `INotifyPropertyChanged` interface
- A collection needs to notify when an item is added or removed from that collection
  - By implementing the `INotifyCollectionChanged` interface
  - `List<T>` does not implement this interface
  - But WPF's `ObservableCollection<T>` does

INotifyPropertyChanged Example

```
public class Book : INotifyPropertyChanged {
   public event PropertyChangedEventHandler PropertyChanged;

   protected void OnPropertyChanged(string name) {
      if(PropertyChanged != null)
         PropertyChanged(this, new PropertyChangedEventArgs(name));
   }
   private string _bookName;
   public string BookName {
      get { return _bookName; }
      set {
         if(_bookName != value) {
            _bookName = value;
            OnPropertyChanged("BookName");
         }
      }
   }
   private decimal _price;
   public decimal Price {
      get { return _price; }
      set {
         if(_price != value) {
            _price = value;
            OnPropertyChanged("Price");
         }
      }
   }
}
```

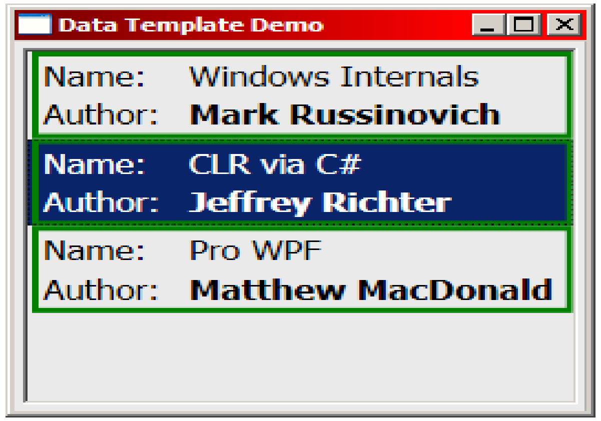
Customizing Rendering

- The default data binding rendering are usually not enough
  - Different types (e.g. images), formatting, colors
- Rendering can be modified via
  - Data templates
  - Value converters
- Technically, unrelated to data binding
  - i.e., can be used even if objects are added to the `ItemsControl` manually

Data Templates

- A data template is a piece of UI that describes how to display a source object

```
<ListBox Margin="4" ItemsSource="{Binding Source={StaticResource books}}"
           HorizontalContentAlignment="Stretch">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <Border BorderBrush="Green" BorderThickness="3">
                <Grid>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="Auto"/>
                        <ColumnDefinition Width="Auto"/>
                     </Grid.ColumnDefinitions>
                    <Grid.RowDefinitions>
                        <RowDefinition />
                        <RowDefinition />
                     </Grid.RowDefinitions>
                    <TextBlock Text="Name: " Margin="2"/>
                    <TextBlock Grid.Column="1" Margin="6,2" Text="{Binding Path=Name}" />
                    <TextBlock Text="Author: " Margin="2" Grid.Row="1" />
                    <TextBlock Margin="6,2" Grid.Column="1" Grid.Row="1" FontWeight="Bold“
                       Text="{Binding Path=Author}" />
                   </Grid>
               </Border>
           </DataTemplate>
       </ListBox.ItemTemplate>
</ListBox>
```



Name: Windows Internals
Author: **Mark Russinovich**
Name: CLR via C#
Author: **Jeffrey Richter**
Name: Pro WPF
Author: **Matthew MacDonald**

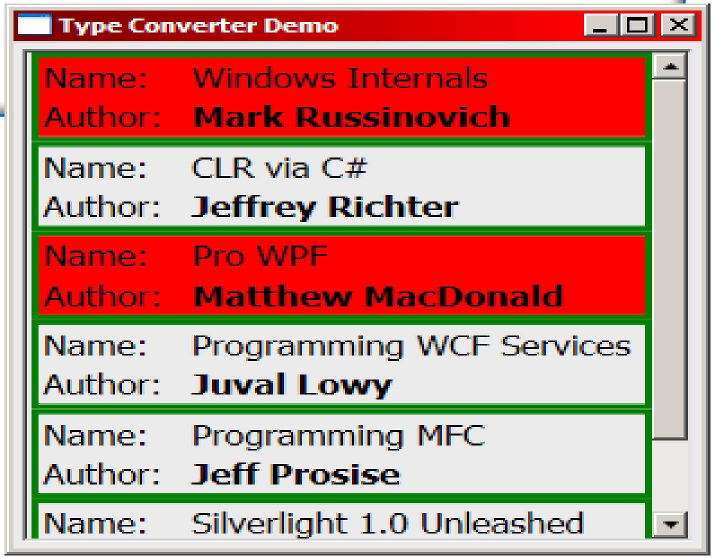
Value Converters

- A value converter can completely alter the way the source is interpreted into the target
- Often used to match source and target that are of incompatible types
  - E.g. show a red background when the price of a book is greater than 45
- Create a converter class (implementing the **`IValueConverter`** interface, in the `System.Windows.Data` namespace)
- Create an instance in XAML in a resource dictionary
- Set the **`Converter`** property of the **`Binding`** object to the converter instance

Value Converter Example

```
<Border BorderBrush="Green" BorderThickness="3"
     Background="{Binding Path=Price, Converter={StaticResource priceToBack}}">
```

```
class PriceToBackgroundConverter : IValueConverter {
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture) {
        if(targetType != typeof(Brush))
            throw new InvalidOperationException("Must convert to a brush!");
        decimal price = (decimal)value;
        return price > 45 ? Brushes.Red : Brushes.Transparent;
    }
}
```



```
Name:    Windows Internals
Author:  Mark Russinovich
Name:    CLR via C#
Author:  Jeffrey Richter
Name:    Pro WPF
Author:  Matthew MacDonald
Name:    Programming WCF Services
Author:  Juval Lowy
Name:    Programming MFC
Author:  Jeff Prosise
Name:    Silverlight 1.0 Unleashed
```

Data Validation

- Validation of data is sometimes required
  - In a `TwoWay` or `OneWayToSource` binding mode
- Raising errors in the data object
  - Throw exceptions in a set property procedure
  - Implement the interface `System.ComponentModel.IDataErrorInfo` and indicate errors without throwing exceptions
- Validation at the binding level
  - Generally more flexible
- Can use a combination of both approaches

Throwing Exceptions in a Setter (1)

```
public class Book : INotifyPropertyChanged {
   private string _name;

   public string Name {
      get { return _name; }
      set {
         if(string.IsNullOrEmpty(value))
            throw new ArgumentException("Book name cannot be empty");
         _name = value;
         if(PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs("Name"));
      }
   }

   public event PropertyChangedEventHandler PropertyChanged;
}
```

- By default, data binding fails silently, with no visual indication

Throwing an Exception in a Setter (2)

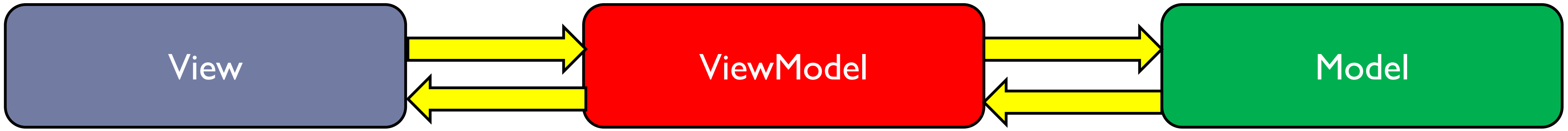
- Must add an **`ExceptionValidationRule`** object to the **`Binding.ValidationRules`** collection
- Or set the **`Binding.ValidatesOnExceptions`** property to true
- Can throw an exception from other related code
  - E.g. exception thrown by a converter

```
<TextBox>
   <TextBox.Text>
      <Binding Path="Name">
         <Binding.ValidationRules>
            <ExceptionValidationRule />
         </Binding.ValidationRules>
      </Binding>
   </TextBox.Text>
</TextBox>
```

MVVM

The MVVM Pattern

- Model – View – ViewModel
- Separation of concerns
- Natural pattern for XAML based applications
  - Data binding is key
- Enables developer-designer workflow
- Increases application testability



MVVM Participants

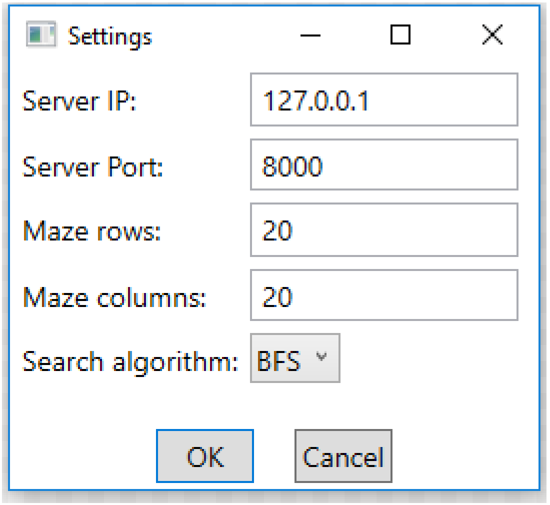
- Model
  - Business logic and data
  - Implements change notification for properties and collections
  - Can implement validation interfaces (e.g. **`IDataErrorInfo`**)
- View
  - Data display and user interactivity
  - Implemented as a `Window`, `UserControl`, `DataTemplate` or custom control
  - Has little or no code behind
- ViewModel
  - UI logic and data for the View
  - Abstracts the Model for View usage
    - Can be an Adapter as well if necessary
  - Exposes commands (**`ICommand`**) to be used by the View
  - Implements change notifications
  - Maintains state for the View (communicates via data binding)

The View

- Provides the user interface and interaction
- The **`DataContext`** property points to the ViewModel
- Updated using property changes from the ViewModel
- Binds to commands (on **`ICommandSource`** elements) provided by the ViewModel

View Example

```
<Grid TextBlock.FontSize="14">
    <Grid.RowDefinitions>
        …
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        …
    </Grid.ColumnDefinitions>
    <TextBlock>Server IP:</TextBlock>
    <TextBox x:Name="txtIP" Grid.Column="1" Text="{Binding ServerIP}"></TextBox>
    <TextBlock Grid.Row="1">Server Port:</TextBlock>
    <TextBox x:Name="txtPort" Grid.Row="1" Grid.Column="2" Text="{Binding ServerPort}"></TextBox>
    <TextBlock Grid.Row="2">Maze rows:</TextBlock>
    <TextBox x:Name="txtRows" Grid.Row="2" Grid.Column="2" Text="{Binding MazeRows}"></TextBox>
    <TextBlock Grid.Row="3">Maze columns:</TextBlock>
    <TextBox x:Name="txtCols" Grid.Row="3" Grid.Column="2" Text="{Binding MazeCols}"></TextBox>
    <TextBlock Grid.Row="4">Search algorithm:</TextBlock>
    <ComboBox x:Name="cboSearchAlgo" Grid.Row="4" Grid.Column="2" HorizontalAlignment="Left"
SelectedIndex="{Binding SearchAlgorithm}" IsEditable="False">
        <ComboBoxItem>BFS</ComboBoxItem>
        <ComboBoxItem>DFS</ComboBoxItem>
    </ComboBox>
    …
</Grid>
```



Server IP:	127.0.0.1
Server Port:	8000
Maze rows:	20
Maze columns:	20
Search algorithm:	BFS

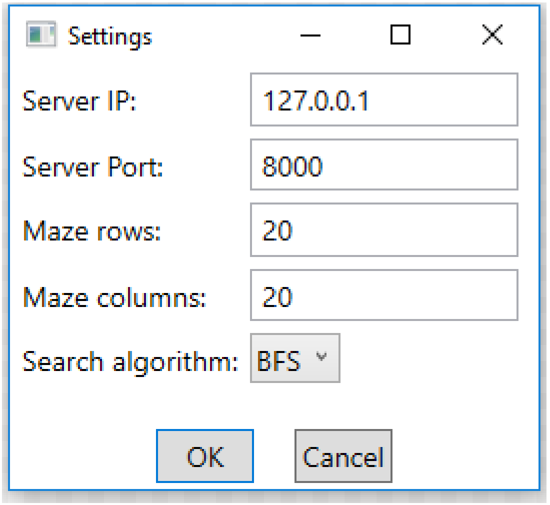
View Example

```
public partial class SettingsWindow : Window
{
    private SettingsViewModel vm;

    public SettingsWindow()
    {
        InitializeComponent();
        vm = new SettingsViewModel();
        this.DataContext = vm;
    }

    private void btnOK_Click(object sender, RoutedEventArgs e)
    {
        vm.SaveSettings();
        MainWindow win = (MainWindow)Application.Current.MainWindow;
        win.Show();
        this.Close();
    }

    private void btnCancel_Click(object sender, RoutedEventArgs e)
    {
        MainWindow win = (MainWindow)Application.Current.MainWindow;
        win.Show();
        this.Close();
    }
}
```


Settings

Server IP:	127.0.0.1
Server Port:	8000
Maze rows:	20
Maze columns:	20
Search algorithm:	BFS

The View Model

- Exposes properties the View binds to
- Can be an adapter if some functionality missing from Model classes
- Exposes commands to be invoked by the view
- Maintains state for the View
- Implements change notifications (**`INotifyPropertyChanged`**, **`INotifyCollectionChanged`**)
  - Uses **`ObservableCollection<T>`** that already implements `INotifyCollectionChanged`

A Base `INotifyPropertyChanged` Class

```
abstract class ViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    public void NotifyPropertyChanged(string propName)
    {
        this.PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propName));
    }
}
```

ViewModel Example

```
class SettingsViewModel : ViewModel
{
    private ISettingsModel model;

    public SettingsViewModel(ISettingsModel model)
    {
        this.model = model;
    }
    public string ServerIP
    {
        get { return model.ServerIP; }
        set {
            model.ServerIP = value;
            NotifyPropertyChanged("ServerIP");
        }
    }
    public int ServerPort
    {
        get { return model.ServerPort; }
        set {
            model.ServerPort = value;
            NotifyPropertyChanged("ServerPort");
        }
    }
    …
    public void SaveSettings()
    {
        model.SaveSettings();
    }
}
```

The Model

- Responsible for business logic and data, e.g.
  - Data Transfer Objects (DTO)
  - POCOs (Plain Old CLR Objects)
  - Generated entity objects
- Provides change notifications
- Provides validation if appropriate
  - In setters, or
  - By implementing **`IDataErrorInfo`**

Model Example

```
interface ISettingsModel
{
    string ServerIP { get; set; }
    int ServerPort { get; set; }
    int MazeRows { get; set; }
    int MazeCols { get; set; }
    int SearchAlgorithm { get; set; }

    void SaveSettings();
}
```

```
class ApplicationSettingsModel : ISettingsModel
{
    public string ServerIP
    {
        get { return Properties.Settings.Default.ServerIP; }
        set { Properties.Settings.Default.ServerIP = value; }
    }

    public int ServerPort
    {
        get { return Properties.Settings.Default.ServerPort; }
        set { Properties.Settings.Default.ServerPort = value; }
    }
    …

    public void SaveSettings()
    {
        Properties.Settings.Default.Save();
    }
}
```