# WPF Tutorial – Script

Contents

[WPF Tutorial – Script 1](#_Toc494979387)

[Preface 2](#_Toc494979388)

[The Beginning 2](#_Toc494979389)

[Adding ViewModels & Bindings 3](#_Toc494979390)

[Binding error output 5](#_Toc494979391)

[User Control with ListBox 5](#_Toc494979392)

[Converter 9](#_Toc494979393)

[Selected Item Binding 11](#_Toc494979394)

[Commands 15](#_Toc494979395)

[Page handling 17](#_Toc494979396)

[ContentControl 20](#_Toc494979397)

[Attached Property 22](#_Toc494979398)

[Exception handling in WPF 23](#_Toc494979399)

# Preface

# The Beginning

First, we need to create a new wpf application with some simple demo data. Our goal is to create the “Lemonade Log Viewer (LLV)” app that display some log entries in a fancy GUI.

* Create new WPF application „WPFDemo\_LLV“
* Add Folder Images
  + Add image lemon.ico
  + Add image warning.ico
  + Add image lemonade.ico
* Add Folder ViewModels
* Add Folder MVVMBasics
* Add Folder Models
  + Add new class Log.cs

class Log {

    public string Name { get; set; }

    public List<LogEntry> LogEntries { get; set; }

    public Log() {

        LogEntries = new List<LogEntry>();

    }

}

* Add new class LogEntry.cs

class LogEntry {

    public int ID { get; set; }

    public DateTime Date { get; set; }

    public string Message { get; set; }

    public string Level { get; set; }

}

* Add new class LogFactory.cs

static class LogFactory {

    public static Log ProduceDemoLog() {

        Log log = new Log() { Name = "DemoLog" };

        for (int i=1;i<20;i++) {

            LogEntry entry = new LogEntry() {

                ID = i,

                Message = $"This is a Test Message #{i}",

                Date = DateTime.Now.AddHours(-i)

            };

            entry.Level = i % 3 == 0 ? "ERROR" : "WARN";

            log.LogEntries.Add(entry);

        }

        return log;

    }

}

* Change to MainWindow.xaml
  + Adjust Height to 550px (Width 525px)
  + Add XAML-Code

<Grid>

        <Grid.RowDefinitions>

            <RowDefinition Height="30" />

            <RowDefinition Height="60" />

            <RowDefinition Height="3\*" />

            <RowDefinition Height="\*" />

        </Grid.RowDefinitions>

        <DockPanel Grid.Row="1" HorizontalAlignment="Center">

            <Image Source="Images/lemonade.ico" Height="64" />

            <StackPanel VerticalAlignment="Center">

  <TextBlock Text="Lemonade Log Viewer" FontSize="18" VerticalAlignment="Center"/>

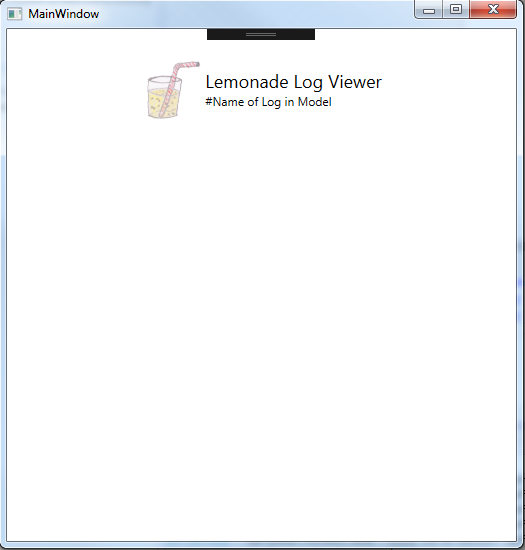
                <TextBlock Text="#Name of Log in Model" />

            </StackPanel>

        </DockPanel>

 </Grid>

* Hit F5 and show current state (not much to see here…)



# Adding ViewModels & Bindings

* Add new class ViewModel.cs in MVVMBasics

abstract class ViewModel : INotifyPropertyChanged {

   public event PropertyChangedEventHandler PropertyChanged;

   protected void OnPropertyChanged([CallerMemberName] string propertyName = "") {

          if (PropertyChanged != null)

              PropertyChanged(this, new PropertyChangedEventArgs(propertyName));

   }

}

abstract class ViewModel<T> : ViewModel {

      public T Model { get; private set; }

      public ViewModel(T model) {

          this.Model = model;

      }

  }

* Add new class LogViewModel.cs in ViewModels

class LogViewModel : ViewModel<Log> {

    public string Name {

        get {

            return Model.Name;

        }

        set {

            Model.Name = value;

            OnPropertyChanged();

        }

    }

   public List<LogEntryViewModel> LogEntries {

        get {

   return Model.LogEntries.Select(entry => new LogEntryViewModel(entry)).ToList();

        }

    }

    public LogViewModel(Log model) : base(model) { }

}

* Add new class LogEntryViewModel.cs in ViewModels

class LogEntryViewModel : ViewModel<LogEntry> {

    public int ID {

        get {

            return Model.ID;

        }

    }

    public DateTime Date {

        get {

            return Model.Date;

        }

        set {

            Model.Date = value;

            OnPropertyChanged();

        }

    }

    public string Message {

        get {

            return Model.Message;

        }

        set {

            Model.Message = value;

            OnPropertyChanged();

        }

    }

    public string Level {

        get {

            return Model.Level;

        }

        set {

            Model.Message = value;

            OnPropertyChanged();

        }

    }

    public LogEntryViewModel(LogEntry model) : base(model) { }

}

* Add class PresenterViewModel.cs

class PresenterViewModel : ViewModel{

    public LogViewModel DemoLog { get; set; }

    public PresenterViewModel() {

        Log demoLog = LogFactory.ProduceDemoLog();

        DemoLog = new LogViewModel(demoLog);

    }

}

* MainWindow.xaml
  + Change Text for TextBlock

<!--<TextBlock Text="#Name of Log in Model" />-->

<TextBlock Text="{ Binding Path=DemoLog.Name}" />

* Hit F5 and show current state -> there is no text. Something forgotten?
* Go into Maindow.xaml.cs (Code-behind!)
  + Add code to constructor

public MainWindow() {

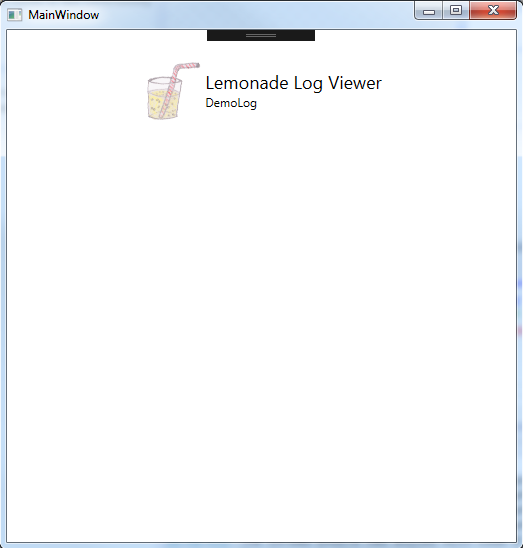
    InitializeComponent();

    PresenterViewModel presenter = new PresenterViewModel();

    this.DataContext = presenter;

}

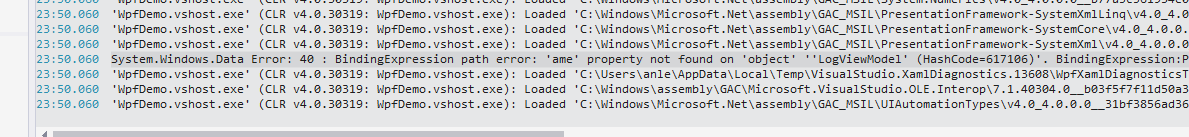
* Hit F5 and voilá



## Binding error output

This show where binding errors are logged

* Make binding “faulty” (delete a letter) and show error message in output

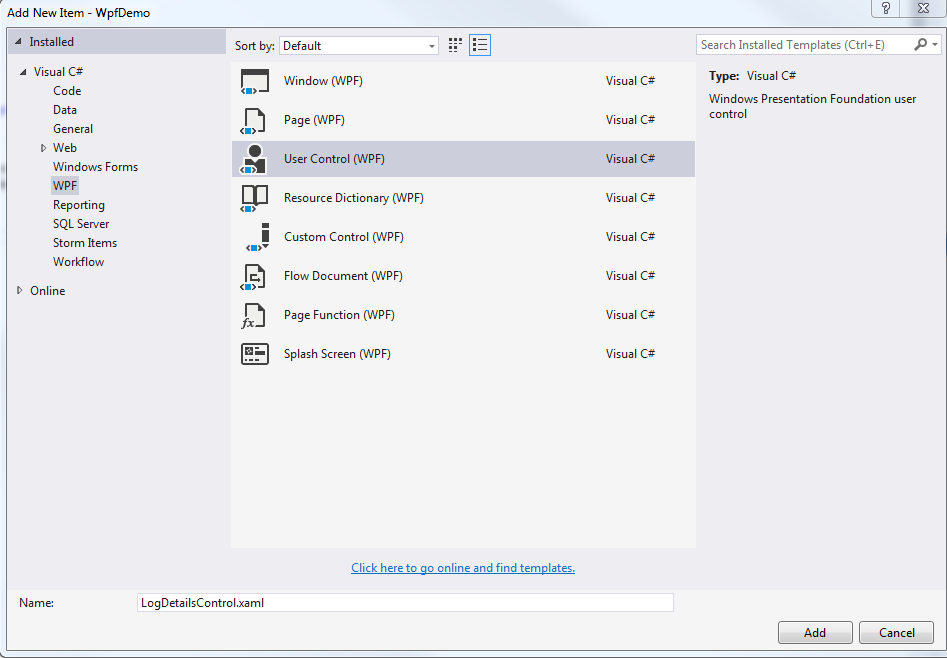


* Revert it back

## User Control with ListBox

Now we want to add a user control where we have some more sophistaced xaml. The reason why we choose a user control will be clarified later.

* Add Folder UserControls
  + Add into folder new UserControl: LogDetailsControl.xaml



* Change namespace (remove UserControls) in XAML and Code-Behind
* Change XAML

<Grid>

        <ListBox x:Name="LogList"

            ItemsSource="{Binding LogEntries}"

                 Height="200" Width="200"

                 HorizontalAlignment="Left">

        </ListBox>

    </Grid>

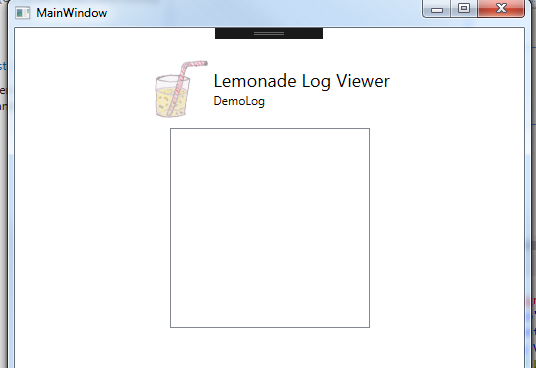
* Add new UserControl to MainWindow.xaml (after DockPanel with Header)

<StackPanel Grid.Row="2" HorizontalAlignment="Center" Margin="10">

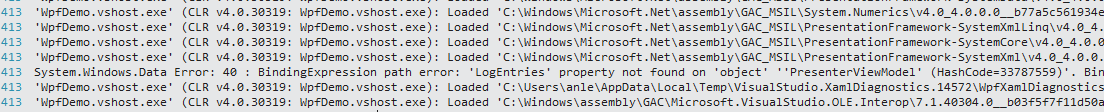
            <local:LogDetailsControl />

</StackPanel>

* Hit F5



* There is a list box, but it’s empty – what happened?
* Show binding error in output



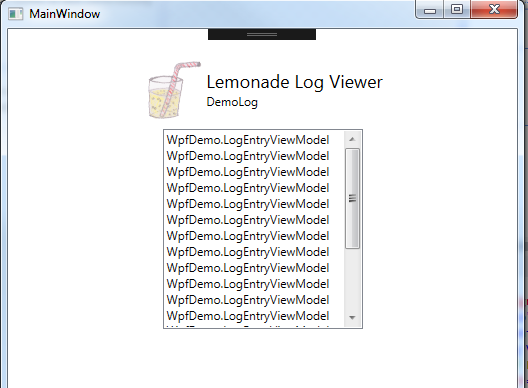
* Add DataContext in former StackPanel containing the UserControl:

<StackPanel Grid.Row="2" HorizontalAlignment="Center" Margin="10">

            <local:LogDetailsControl DataContext="{Binding DemoLog}"/>

  </StackPanel>

* Now the ListBox contains 19 items



No want to display the content in a fancy way.

* Adding an ItemTemplate to ListBox

<ListBox.ItemTemplate>

    <DataTemplate>

      <StackPanel Orientation="Horizontal" VerticalAlignment="Center" Height="32">

        <Image Source="pack://application:,,,/Images/lemon.ico" Height="32" />            <TextBlock Text="{Binding ID}" />

         <TextBlock Text=" | " />

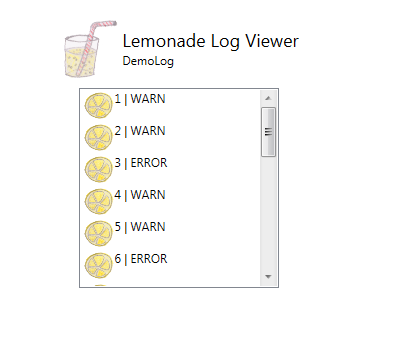
         <TextBlock Text="{Binding Level}" />

         </StackPanel>

     </DataTemplate>

 </ListBox.ItemTemplate>

* Hit F5



Better, but still needs some styling

* Add StackPanel.Resources to style TextBlocks:

<StackPanel.Resources>

           <Style TargetType="{x:Type TextBlock}">

                  <Setter Property="VerticalAlignment" Value="Center" />                      </Style>

</StackPanel.Resources>

* Hit F5

# Converter

No let’s demonstrate the use of a converter – first we want to display WARN log entries blue and ERROR log entries red. This can be done with use of the ViewModel.

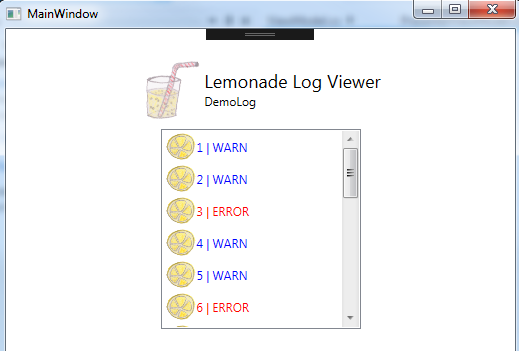
* Add Setter to Resources

<Setter Property="Foreground" Value="{Binding Path=LevelAsColor}" />

* Add to class LogEntryViewModel.cs

public string LevelAsColor => Model.Level == "ERROR" ? "Red" : "Blue";

* Hit F5



Now let’s do the same with a converter. (But instead of blue we want green – green is hope and a friendlier color)

* Uncomment Add Setter to Resources, add new setter

<!--<Setter Property="Foreground" Value="{Binding Path=LevelAsColor}" />-->

<Setter Property="Foreground" Value="{Binding Path=Level, Converter={StaticResource level2color}}" />

* Add new folder Converter
  + Add Level2ColorConverter.cs

[ValueConversion(typeof(string), typeof(SolidColorBrush))]

class LevelToColorConverter : IValueConverter {

    public object Convert(object value, Type targetType, object parameter, CultureInfo culture) {

        string level = value as string;

        if (String.IsNullOrEmpty(level)) return Binding.DoNothing;

        if (level == "ERROR")

            return new SolidColorBrush(Colors.Red);

        else

            return new SolidColorBrush(Colors.Green);

    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture) {

        return Binding.DoNothing;

    }

}

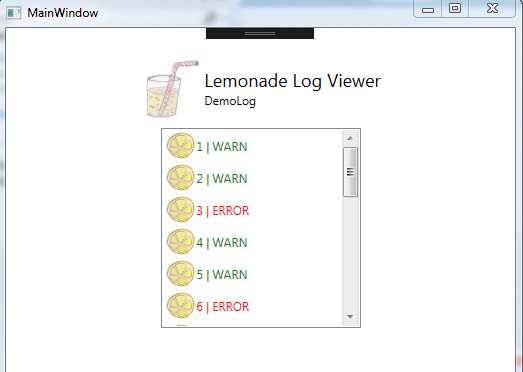
* Add to LogDetailsControl.xaml previously created Converter

<UserControl.Resources>

        <local:LevelToColorConverter x:Key="level2color" />

</UserControl.Resources>

* Hit F5



## Selected Item Binding

Now we need to have some kind of behavior when we click on an item in the list box.

* Change ListBox, add SelectItem Property in LogDetailsControl.xaml

<ListBox x:Name="LogList"

         ItemsSource="{Binding LogEntries}"

         Height="200" Width="200"

         HorizontalAlignment="Left"

         SelectedItem="{Binding SelectedEntry}"

         >

* Add Property to LogViewModel.cs

LogEntryViewModel \_selectedEntry;

public LogEntryViewModel SelectedEntry {

    get {

        return \_selectedEntry;

    }

    set {

        \_selectedEntry = value;

        OnPropertyChanged();

    }

}

* Set breakpoint in Setter, HIT F5 and demonstrate that breakpoint is reached

No we want to add something that displays log message

* In LogDetailsControl.xaml, add ColumnsDefinition to main grid

<Grid.ColumnDefinitions>

            <ColumnDefinition Width="\*" />

            <ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

* Add ContentControl

<ContentControl Grid.Column="1" Margin="10" Content="{Binding SelectedEntry}">

            <ContentControl.ContentTemplate>

               <DataTemplate>

                    <StackPanel>

                        <TextBlock Text="{Binding Message}" />

                        <TextBlock Text="{Binding Date}" />

    <Image Source="pack://application:,,,/Images/warning.ico" Height="64">

                    </StackPanel>

                </DataTemplate>

            </ContentControl.ContentTemplate>

        </ContentControl>

2 issues here – when selecting an item first time, the listbox will “jump” some pixels to the left. And the Error image is displayed when nothing is selected.

Fix issue 1 by changing MainWindow.xaml

* add width to StackPanel

<StackPanel Grid.Row="2" HorizontalAlignment="Center" Margin="10" Width="450">

            <local:LogDetailsControl DataContext="{Binding DemoLog}"/>

        </StackPanel>

Fix issue 2 is a bit more complicated. The problem is that SelectedEntry has a binding to a null-Object.

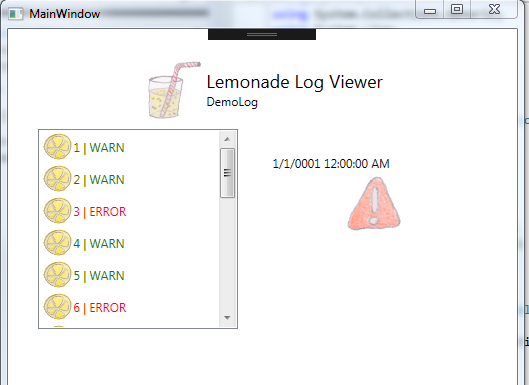
* First try by changing constructur in LogViewModel.cs

public LogViewModel(Log model) : base(model) {

    SelectedEntry = new LogEntryViewModel(new LogEntry());

}

* Hit F5



Better, but still not there yet. Let’s try a fix by using a BoolConverter

* Add BoolConverter.cs to folder Converter

[ValueConversion(typeof(object), typeof(object))]

class BoolConverter : IValueConverter {

    public object TrueValue { get; set; }

    public object FalseValue { get; set; }

    public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture) {

        bool typedValue = (bool)value;

        return typedValue ? this.TrueValue : this.FalseValue;

    }

    public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture) {

        return Binding.DoNothing;

    }

}

* Add static resource to UserControl.Resources

<UserControl.Resources>

        <local:LevelToColorConverter x:Key="level2color" />

        <local:BoolConverter TrueValue="Visible" FalseValue="Hidden" x:Key="Bool2VisibilityConverter"/>

    </UserControl.Resources>

* Add Property to LogEntryViewModel.cs

public bool IsError => Model.Level == "ERROR" ? true : false;

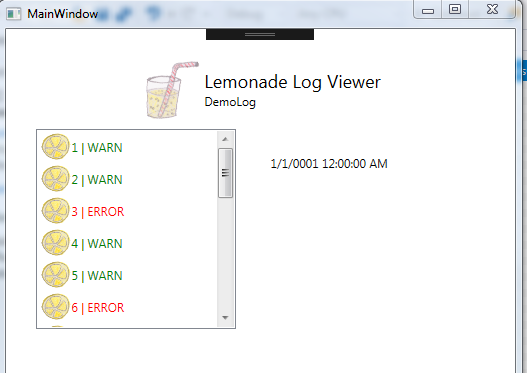
* Change Image Visibility by using Binding

<Image Source="pack://application:,,,/Images/warning.ico" Height="64"

                       Visibility="{Binding IsError, Converter={StaticResource Bool2VisibilityConverter}}">

Great! Now the  error icon is omitted.

But we have a new issue: Since we added a new empty object into SelectedEntry, there is a default date – 1/1/0001 12:00:00 AM.



We could make the DataType nullable, or

* we can add a trigger

<TextBlock Text="{Binding Date}">

    <TextBlock.Style>

      <Style TargetType="{x:Type TextBlock}">

        <Style.Triggers>

         <DataTrigger Binding="{Binding Date}" Value="1/1/0001 12:00:00 AM">

               <Setter Property="Visibility" Value="Hidden" />

              </DataTrigger>

        </Style.Triggers>

       </Style>

     </TextBlock.Style>

 </TextBlock>

* Hit F5

Works!

# Commands

Next let’s demonstrate the use of buttons and commands.

* Wrap the ContentControl within a StackPanel, in order to have a Button below

<StackPanel Grid.Column="1">

            <ContentControl Margin="10" Content="{Binding SelectedEntry}">

…

            <Button Width="70" Height="20" Content="Delete"

            Command="{Binding DeleteLogEntryCommand}" CommandParameter="{Binding SelectedEntry}"

            />

</StackPanel>

* Add class DelegateCommand.cs into folder MVVMBasics

class DelegateCommand : ICommand {

    private readonly Predicate<object> \_canExecute;

    private readonly Action<object> \_execute;

    public event EventHandler CanExecuteChanged;

    public DelegateCommand(Action<object> execute)

                   : this(execute, null) {

    }

    public DelegateCommand(Action<object> execute,

                   Predicate<object> canExecute) {

        \_execute = execute;

        \_canExecute = canExecute;

    }

    public bool CanExecute(object parameter) {

        if (\_canExecute == null) {

            return true;

        }

        return \_canExecute(parameter);

    }

    public void Execute(object parameter) {

        \_execute(parameter);

    }

    public void RaiseCanExecuteChanged() {

        if (CanExecuteChanged != null) {

            CanExecuteChanged(this, EventArgs.Empty);

        }

    }

}

* Add to LogViewModel.cs

public DelegateCommand DeleteLogEntryCommand { get; set; }

* Comment out current LogEntries-Property implementation (won’t work) and add observerable collection

//public List<LogEntryViewModel> LogEntries {

//    get {

//        return Model.LogEntries.Select(entry => new LogEntryViewModel(entry)).ToList();

//    }

//}

public ObservableCollection<LogEntryViewModel> LogEntries { get; set; }

* Add to constructor in LogViewModel.cs

public LogViewModel(Log model) : base(model) {

          SelectedEntry = new LogEntryViewModel(new LogEntry());

          DeleteLogEntryCommand = new DelegateCommand(DoDeleteLogEntry, CanDeleteLogEntry);

LogEntries = new ObservableCollection<LogEntryViewModel>();

foreach (LogEntry LogEntry in Model.LogEntries)

    LogEntries.Add(new LogEntryViewModel(LogEntry));

      }

* Add implementation of command

public bool CanDeleteLogEntry(object sender) {

          return true;

}

      public void DoDeleteLogEntry(object sender) {

          LogEntryViewModel selEntry = sender as LogEntryViewModel;

          Model.LogEntries = Model.LogEntries.Where(entry => entry.ID != selEntry.ID).ToList();

          LogEntries.Remove(selEntry);

      }

* Hit F5 and show that it works

Now we have the issue that Button is working, but always activated .

* To fix this we must Change CanDeleteLogEntry – Function.

public bool CanDeleteLogEntry(object sender) {

          //return true;

          return SelectedEntry?.ID > 0;

      }

* Hit F5 – now the Button is always disabled.

There must be some kind of Notification to the GUI

* Change Property SelectedEntry

public LogEntryViewModel SelectedEntry {

          get {

              return \_selectedEntry;

          }

          set {

              \_selectedEntry = value;

              OnPropertyChanged();

              DeleteLogEntryCommand?.RaiseCanExecuteChanged();

          }

      }

If we want to fix that the “Error sign” also disappears after an “ERROR Message” is deleted we need to set SelectedEntry back to an “empty” ViewMode

public void DoDeleteLogEntry(object sender) {

    LogEntryViewModel selEntry = sender as LogEntryViewModel;

    Model.LogEntries = Model.LogEntries.Where(entry => entry.ID != selEntry.ID).ToList();

    LogEntries.Remove(selEntry);

    SelectedEntry = new LogEntryViewModel(new LogEntry());

}

But this seems odd. Comment it out – instead we define a “Fallback Value” within the error sign image:

Visibility="{Binding IsError, Converter={StaticResource Bool2VisibilityConverter},FallbackValue=Hidden}">

* Hit F5

This would be the preferred way for the textblock as well, but we wanted to demonstrate the use of a trigger.

# Page handling

Now we want to add a “different” page.

* First, we add two buttons in MainWindow.xaml

<WrapPanel Grid.Row="3" HorizontalAlignment="Center">

            <Button Content="&lt;--Prev" Height="32" Margin="10" Command="{Binding PrevPageCommand}" />

            <Button Content="Next--&gt;" Height="32" Margin="10" Command="{Binding NextPageCommand}"/>

        </WrapPanel>

No we introduce a second view model and a second UserControl

* Add OnlyUIViewModel.cs

class OnlyUIViewModel {

    public string Title => "This is the next page!";

}

* Add SecondPageUserControl.xaml (as UserControl), adjust namespace to WpfDemo (also in Code-Begin)

<TextBlock Text="{Binding Title}" />

* Now we make the content decision depending on data templates within MainWindow.xaml

<Window.Resources>

        <DataTemplate DataType="{x:Type local:LogViewModel}">

            <local:LogDetailsControl />

        </DataTemplate>

        <DataTemplate DataType="{x:Type local:OnlyUIViewModel}">

            <local:SecondPageUserControl />

        </DataTemplate>

 </Window.Resources>

* Change the “fixed” UserControl to a content-Control

<StackPanel Grid.Row="2" HorizontalAlignment="Center" Margin="10" Width="450">

            <!--<local:LogDetailsControl DataContext="{Binding DemoLog}"/>-->

            <ContentControl Content="{Binding CurrentView}" />

        </StackPanel>

* Now we add the Command definition and CurrentView into PresenterViewModel.cs

public object CurrentView { get; set; }

public DelegateCommand NextPageCommand { get; set; }

public DelegateCommand PrevPageCommand { get; set; }

* Extend the constructor

public PresenterViewModel() {

    Log demoLog = LogFactory.ProduceDemoLog();

    DemoLog = new LogViewModel(demoLog);

    CurrentView = DemoLog;

    NextPageCommand = new DelegateCommand(

        nopar => {

            CurrentView = new OnlyUIViewModel();

            OnPropertyChanged("CurrentView");

            PrevPageCommand.RaiseCanExecuteChanged();

            NextPageCommand.RaiseCanExecuteChanged();

        },

        nopar => {

            //return true;

            return (CurrentView as OnlyUIViewModel == null);

        });

    PrevPageCommand = new DelegateCommand(

        nopar => {

            CurrentView = DemoLog;

            OnPropertyChanged("CurrentView");

            PrevPageCommand.RaiseCanExecuteChanged();

            NextPageCommand.RaiseCanExecuteChanged();

        },

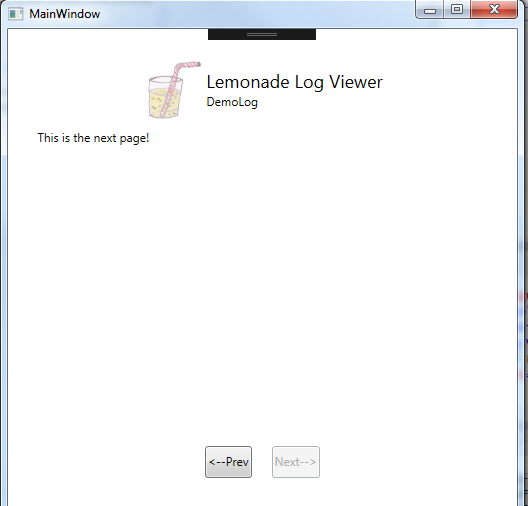
        nopar => {

            return (CurrentView as OnlyUIViewModel != null);

        });

}

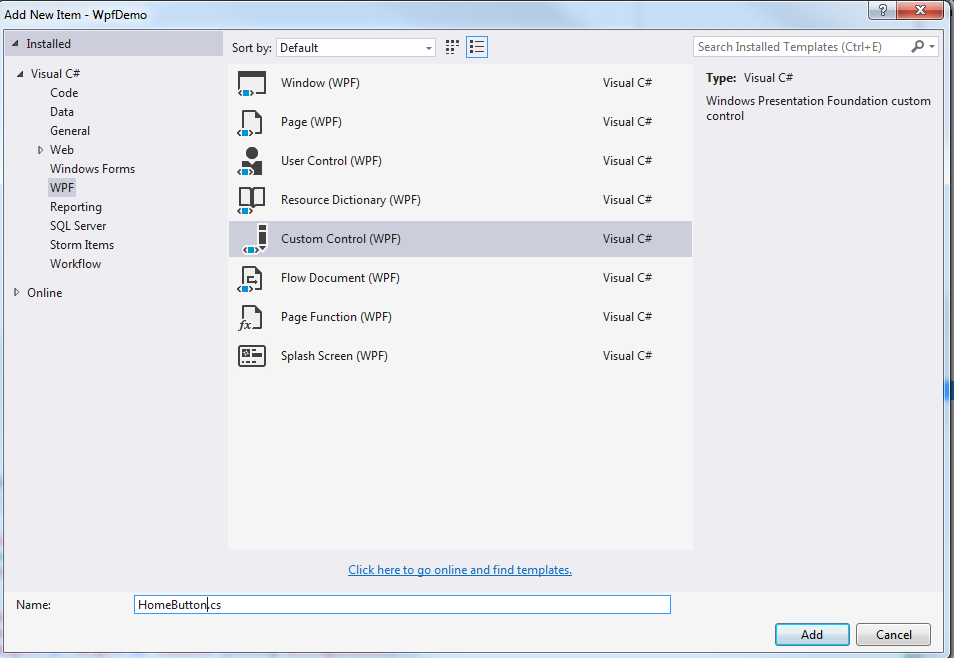
* Hit F5 – Taadaa



# ContentControl

Finally, we want to show a ContentControl – with a dependency Property and an attached Property. The Content Control is a home icon displaying an URL inside.

* Add home.ico to Images
* Add folder CustomControls
  + Add template for CustomControl: HomeButton.cs



* Add Dependency Property (type string with name URL, typeof(HomeButton) and “” as default

public string URL {

         get { return (string)GetValue(URLProperty); }

         set { SetValue(URLProperty, value); }

     }

     // Using a DependencyProperty as the backing store for URL.  This enables animation, styling, binding, etc...

     public static readonly DependencyProperty URLProperty =

         DependencyProperty.Register("URL", typeof(string), typeof(HomeButton), new PropertyMetadata(""));

Now we need a new template for this.

* Add a ResourceDictionary as HomeButton.xaml to CustomControls folder

<Style TargetType="{x:Type local:HomeButton}">

        <Setter Property="Template">

            <Setter.Value>

                <ControlTemplate TargetType="{x:Type local:HomeButton}">

                    <Grid HorizontalAlignment="Center" VerticalAlignment="Center">

                        <Image Source="pack://application:,,,/Images/home.ico" Width="64" />

                        <TextBlock Name="tb" VerticalAlignment="Center" Text="{TemplateBinding URL}" Foreground="Blue" TextDecorations="Underline" />

                    </Grid>

                    <ControlTemplate.Triggers>

                        <Trigger Property="IsMouseOver" Value="True">

                            <Setter Property="FontWeight" TargetName="tb" Value="Bold"/>

                        </Trigger>

                    </ControlTemplate.Triggers>

                </ControlTemplate>

            </Setter.Value>

        </Setter>

    </Style>

* Now we need to add the control to SecondPageUserControl.xaml
* First, we need the resources in the UserControl

<UserControl.Resources>

        <ResourceDictionary>

            <ResourceDictionary.MergedDictionaries>

                <ResourceDictionary Source="pack://application:,,,/CustomControls/HomeButton.xaml" />

            </ResourceDictionary.MergedDictionaries>

        </ResourceDictionary>

    </UserControl.Resources>

* Second, we need to add the control itself

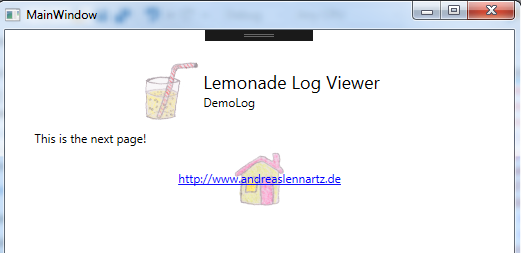
<StackPanel>

            <TextBlock Text="{Binding Title}" />

            <local:HomeButton URL="http://www.andreaslennartz.de" />

 </StackPanel>

* Hit F5



# Attached Property

Last but not least, we want a generic attribute for a TextBlock that makes Text Yellow that consists only of letters.

* Add a new class AttachedPropertyExample.cs in folder CustomControls

static class AttachedPropertyExample {

    public static bool GetAllowOnlyString(DependencyObject obj) {

        return (bool)obj.GetValue(AllowOnlyStringProperty);

    }

    public static void SetAllowOnlyString(DependencyObject obj, bool value) {

        obj.SetValue(AllowOnlyStringProperty, value);

    }

    // Using a DependencyProperty as the backing store for AllowOnlyString.  This enables animation, styling, binding, etc...

    public static readonly DependencyProperty AllowOnlyStringProperty =

        DependencyProperty.RegisterAttached("AllowOnlyString", typeof(bool), typeof(AttachedPropertyExample), new PropertyMetadata(false, MakeAlphaStringYellow));

    private static void MakeAlphaStringYellow(DependencyObject d, DependencyPropertyChangedEventArgs e) {

        if (d is TextBlock) {

            TextBlock txtObj = (TextBlock)d;

            if (Regex.IsMatch(txtObj.Text, @"^([a-zA-Z]| )\*$")) {

                txtObj.Foreground = Brushes.LightGoldenrodYellow;

            };

        }

    }

}

* Now add a new textblock to SecondPageUserControl.xaml

<StackPanel>

            <TextBlock Text="{Binding Title}"/>

            <local:HomeButton URL="http://www.andreaslennartz.de" />

            <TextBlock Text="Lemonade for everyone" local:AttachedPropertyExample.AllowOnlyString="True" />

        </StackPanel>

# Exception handling in WPF

Finally, another (last) hint for exception handling in WPF

* In Code-Behind of App.xaml.cs, add the following event

    public partial class App : Application {

        public App() {

            this.DispatcherUnhandledException += App\_DispatcherUnhandledException;

        }

        void App\_DispatcherUnhandledException(object sender, DispatcherUnhandledExceptionEventArgs e) {

#if !DEBUG

            System.Windows.MessageBox.Show(string.Format("An error occured: {0}", e.Exception.Message), "Error");

            e.Handled = true;

#endif

        }

    }

This is the same as a big try /catch block inside the main – but also catching events inside the Dispatcher Thread… (not all Exception will be triggered within the Dispatcher Thread… but this is advanced stuff)

Done! Thanks for the attention