WPF

Table of Contents

[NuGet 1](#_Toc27424441)

[App Events 1](#_Toc27424442)

[Window Events 1](#_Toc27424443)

[Controls 2](#_Toc27424444)

[Button content 2](#_Toc27424445)

[Bindings 2](#_Toc27424446)

[Between controls in XAML 2](#_Toc27424447)

[Data 2](#_Toc27424448)

[Collection 3](#_Toc27424449)

[XAML 3](#_Toc27424450)

[Resources 3](#_Toc27424451)

[Binary Resource 3](#_Toc27424452)

[Load Image 3](#_Toc27424453)

[Logical Resources 4](#_Toc27424454)

[Static Resource – cannot be change by C# 4](#_Toc27424455)

[Synamic Resource – can be change by C# 4](#_Toc27424456)

[Change resource by C# 4](#_Toc27424457)

[Dictionary 4](#_Toc27424458)

[Style 4](#_Toc27424459)

[RoundButtonTemplate.xaml 4](#_Toc27424460)

[Use 4](#_Toc27424461)

[Skins 5](#_Toc27424462)

[Change skin by C# 5](#_Toc27424463)

[Global Exceptions 5](#_Toc27424464)

# NuGet

Extented.WPF.ToolKit

# App Events

protected override void OnStartup(StartupEventArgs e) {

base.OnStartup(e);

}

protected override void OnActivated(EventArgs e) {

base.OnActivated(e);

}

protected override void OnLoadCompleted(NavigationEventArgs e) {

base.OnLoadCompleted(e);

}

protected override void OnSessionEnding(SessionEndingCancelEventArgs e) {

base.OnSessionEnding(e);

}

protected override void OnDeactivated(EventArgs e) {

base.OnDeactivated(e);

}

protected override void OnExit(ExitEventArgs e) {

base.OnExit(e);

}

# Window Events

public partial class MainWindow : Window {

protected override void OnInitialized(EventArgs e) {

base.OnInitialized(e);

}

protected override void OnActivated(EventArgs e) {

base.OnActivated(e);

}

protected override void OnContentRendered(EventArgs e) {

base.OnContentRendered(e);

}

private void MainWindow\_OnLoaded(object sender, RoutedEventArgs e) {

throw new **NotImplementedException**();

}

protected override void OnClosing(CancelEventArgs e) {

base.OnClosing(e);

}

protected override void OnClosed(EventArgs e) {

base.OnClosed(e);

}

protected override void OnDeactivated(EventArgs e) {

base.OnDeactivated(e);

}

}

# Controls

## Button content

<Button x:Name="OrdinaryButton" Height="30" Width="160" Click="OrdinaryButton\_OnClick">

<Button.Content>

<StackPanel Orientation="Horizontal">

<Ellipse Fill="Red" Width="15" Height="15" />

<TextBlock Text=" Click Me, I'm still a button" />

</StackPanel>

</Button.Content>

</Button>

# Bindings

## Between controls in XAML

<Slider Name="slider1" Value="{Binding ElementName=textBox1, Path=Text, Mode=TwoWay, Delay=10}"/>

<TextBox Name="textBox1" Text="TextBox"/>

OR

<Slider Name="slider2"/>

<TextBox Name="textBox2" Text="{Binding ElementName=slider2, Path=Value, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>

## Data

public class CraftsmanViewModel : INotifyPropertyChanged {

private int age;

public string Name { get; set; }

public string LastName { get; set; }

public int Age {

get {

return age;

}

set {

age = value;

OnPropertyChanged();

}

}

public string PicturePath { get; set; }

public event PropertyChangedEventHandler PropertyChanged;

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

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

}

## Collection

public ObservableCollection<Craftsman> Craftsmen { get; private set; }

## XAML

<Window x:Class="DataBinding.BindingToCustomType"

.

.

DataContext="{Binding RelativeSource = {RelativeSource Self}}">

<StackPanel>

<Image x:Name="CraftsmanPicture" Source="{Binding Craftsman.PicturePath}"/>

<TextBlock x:Name="Name" Text="{Binding Path=Craftsman.Name, Mode=OneWay}"/>

<TextBlock x:Name="LastName" Text="{Binding Path=Craftsman.LastName, Mode=OneWay}"/>

<TextBox x:Name="Age" Text="{Binding Path=Craftsman.Age, Mode=OneWay}"/>

</StackPanel>

</Window>

# Resources

## Binary Resource

<Image x:Name="CraftsmanPicture"/>

## Load Image

public BinaryResources() {

InitializeComponent();

var assembly = Assembly.GetExecutingAssembly();

var resourceName = "Resources.Seemann.jpeg";

Stream stream = assembly.GetManifestResourceStream(resourceName);

byte[] result = new byte[stream.Length];

stream.Read(result, 0, result.Length);

CraftsmanPicture.Source = LoadImage(result);

}

private static BitmapImage LoadImage(byte[] imageData) {

if (imageData == null || imageData.Length == 0) {

return null;

}

var image = new BitmapImage();

using (var mem = new MemoryStream(imageData)) {

mem.Position = 0;

image.BeginInit();

image.CreateOptions = BitmapCreateOptions.PreservePixelFormat;

image.CacheOption = BitmapCacheOption.OnLoad;

image.UriSource = null;

image.StreamSource = mem;

image.EndInit();

}

image.Freeze();

return image;

}

}

## Logical Resources

<Window.Resources>

<SolidColorBrush x:Key="CustomBrush">

<SolidColorBrush.Color>Red</SolidColorBrush.Color>

</SolidColorBrush>

</Window.Resources>

### Static Resource – cannot be change by C#

<Rectangle Width="120" Height="120" Fill="{StaticResource CustomBrush}" StrokeThickness="2"/>

### Synamic Resource – can be change by C#

<Rectangle Width="120" Height="120" Fill="{DynamicResource CustomBrush}" StrokeThickness="2"/>

### Change resource by C#

this.Resources["CustomBrush"] = new SolidColorBrush(Colors.Green);

## Dictionary

<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:Resources">

<Color x:Key="RedColor" A="255" R="255" G="0" B="0"/>

<Color x:Key="GreenColor" A="255" R="0" G="255" B="0"/>

<Color x:Key="BlueColor" A="255" R="0" G="0" B="255"/>

</ResourceDictionary>

# Style

## RoundButtonTemplate.xaml

<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

<ControlTemplate x:Key="RoundButton" TargetType="Button">

<Grid>

<Ellipse Stroke="Black" StrokeThickness="2">

<Ellipse.Fill>

<RadialGradientBrush>

<GradientStop Offset="0" Color="Aqua" />

<GradientStop Offset="1" Color="Yellow" />

<GradientStop Offset="1" Color="Aqua" />

<RadialGradientBrush.Transform>

<TransformGroup>

<ScaleTransform ScaleY="0.65" />

</TransformGroup>

</RadialGradientBrush.Transform>

</RadialGradientBrush>

</Ellipse.Fill>

</Ellipse>

<ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>

</Grid>

</ControlTemplate>

</ResourceDictionary>

## Use

<Button x:Name="Exit" Content="Exit" Click="Exit\_OnClick" Template="{StaticResource RoundButton}"/>

<ListBox ItemsSource="{Binding Craftsmen}" ItemTemplate="{StaticResource CraftsmanTemplate}" />

# Skins

## Change skin by C#

public enum AppSkin {

Default,

Green

}

public static class SkinManager {

public static AppSkin CurrentThemeType { get; private set; }

public static void SetSkin(AppSkin appSkin) {

CurrentThemeType = appSkin;

Application.Current.Resources.MergedDictionaries[0].MergedDictionaries.Clear();

var newDictionary = new ResourceDictionary();

switch (appSkin) {

case AppSkin.Default:

newDictionary.Source =

new Uri("pack://application:,,,/StylesTemplatesSkinsThemes;component/Skins/DefaultSkin.xaml");

break;

case AppSkin.Green:

newDictionary.Source =

new Uri("pack://application:,,,/StylesTemplatesSkinsThemes;component/Skins/GreenSkin.xaml");

break;

}

Application.Current.Resources.MergedDictionaries[0].MergedDictionaries.Add(newDictionary);

}

}

# Global Exceptions

public partial class App : Application {

protected override void OnStartup(StartupEventArgs e) {

DispatcherUnhandledException += MainUIThreadUnhandledException;

Current.Dispatcher.UnhandledException += DispatcherOnUnhandledException;

AppDomain.CurrentDomain.UnhandledException += CurrentDomainOnUnhandledException;

TaskScheduler.UnobservedTaskException += TaskSchedulerOnUnobservedTaskException;

}

private void MainUIThreadUnhandledException(object sender, DispatcherUnhandledExceptionEventArgs e) {

Debug.WriteLine("MainUIThreadUnhandledException");

e.Handled = true;

}

private void DispatcherOnUnhandledException(object sender, DispatcherUnhandledExceptionEventArgs e) {

Debug.WriteLine("DispatcherOnUnhandledException");

e.Handled = true;

}

private void CurrentDomainOnUnhandledException(object sender, UnhandledExceptionEventArgs e) {

Debug.WriteLine("CurrentDomainOnUnhandledException");

}

private void TaskSchedulerOnUnobservedTaskException(object sender, UnobservedTaskExceptionEventArgs e) {

Debug.WriteLine("TaskSchedulerOnUnobservedTaskException");

e.SetObserved();

}

}