

Module 1

"A Quick Tour of WPF Fundamentals"



TEKNOLOGISK
INSTITUT

Agenda

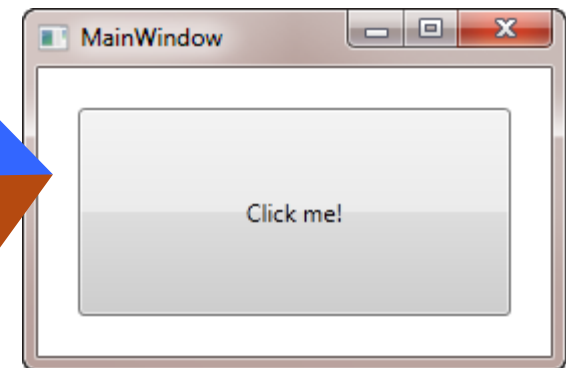
- ▶ **WPF = XAML + Code**
- ▶ Controls and Layout
- ▶ Resources
- ▶ Styles and Triggers
- ▶ Animations

WPF = XAML + Code

▶ XAML

```
<Window x:Class="MyWindow"  
        Title="MainWindow">  
    <Grid>
```

```
public partial class MyWindow : Window  
{  
    private void ButtonClick(  
        object sender, RoutedEventArgs e)  
    {  
        // Handle click  
    }  
}
```



▶ Complex compilation process

▶ *.g.cs

▶ Code-Behind



Dual Capabilities

```
<Button  
  RenderTransformOrigin="0.5,0.5"  
  Width="200" Height="100"  
  Content="Click me!">  
  <Button.RenderTransform>  
    <RotateTransform Angle="315" />  
  </Button.RenderTransform>  
</Button>
```

```
Button btn = new Button();  
btn.Width = 200;  
btn.Height = 100;  
btn.Content =  
  "Click me!";  
btn.RenderTransformOrigin =  
  new Point(0.5,0.5);
```

```
Transform rt =  
  new RotateTransform();  
rt.Angle = 315;  
btn.RenderTransform = rt;
```



Introducing Dependency Properties

- ▶ New type of property specific to WPF
 - Rich functionality directly from XAML
 - Change notification
 - Property value inheritance
 - Depend on multiple providers
- ▶ Dependency properties =
.NET properties + additional WPF infrastructure

Dependency Property Example

```
public class Button : ButtonBase
{
    public static readonly DependencyProperty IsDefaultProperty; // The dependency property

    static Button()
    {
        // Register the property
        Button.IsDefaultProperty = DependencyProperty.Register(
            "IsDefault", typeof(bool), typeof(Button),
            new FrameworkPropertyMetadata(false,
                new PropertyChangedCallback(OnIsDefaultChanged)));
    }

    public bool IsDefault // A .NET property wrapper (optional)
    {
        get { return (bool)GetValue(Button.IsDefaultProperty); }
        set { SetValue(Button.IsDefaultProperty, value); }
    }

    // A property changed callback (optional)
    private static void OnIsDefaultChanged( DependencyObject o,
        DependencyPropertyChangedEventArgs e) { ... }
}
```

Dependency Property Features

- ▶ Many features of WPF are only for dependency properties
 - Animations
 - Triggers
 - ...
- ▶ Attached properties
 - E.g. **DockPanel.Dock**



Agenda

- ▶ WPF = XAML + Code
- ▶ **Controls and Layout**
- ▶ Resources
- ▶ Styles and Triggers
- ▶ Animations

Introducing Content Controls

- ▶ All content controls derive from **ContentControl**
- ▶ Contain single nested element
 - **Content** property
 - Content can be any type...!
 - **UIElements** are rendered
 - Other elements are rendered as **TextBlock** via **object.ToString()**

Introducing Items Controls

- ▶ All item controls derive from **ItemsControl**
- ▶ Contains a collection of elements
 - **Items** property of type **ItemCollection**
 - **ItemsSource**
 - Items can be of any type...!
 - **UIElements** are rendered
 - Other elements are rendered as **TextBlock** via **object.ToString()**

Virtualizing Item Controls

- ▶ Some item controls can be virtualized for performance reasons
- ▶ Set **ItemsControl.ItemsPanel** to e.g. **VirtualizingStackPanel**
 - Only creates the items necessary!
- ▶ Turn virtualization on/off via
 - **VirtualizingStackPanel.IsVirtualizing**
- ▶ Note:
 - Virtualization in fact only happens when the **VirtualizingStackPanel** itself creates its own item containers! (e.g. when data binding)
- ▶ See <https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/optimizing-performance-controls>



WPF Layout System

- ▶ Two-pass layout system in WPF
 - 1. Measurement Pass
 - Evaluate **Children** for **DesiredSize**
 - 2. Arrangement Pass
 - Determine final size of each child and place in layout control

- ▶ Two distinct transformation properties on all **FrameworkElements**
 - **LayoutTransform**
 - **RenderTransform**
 - **RenderTransformOrigin**



Control Properties for Layout

▶ FrameworkElement

- Margin
- HorizontalAlignment
- VerticalAlignment
- FlowDirection

"spacing outside"

▶ Control : FrameworkElement

- Padding
- HorizontalContentAlignment
- VerticalContentAlignment

"spacing inside"

- ▶ **FrameworkElement.HorizontalAlignment** must be set in order to size to content in e.g. **StackPanel**



Layout Controls (or "Panels")

- ▶ StackPanel
- ▶ DockPanel
- ▶ WrapPanel
- ▶ Grid
- ▶ UniformGrid
- ▶ Canvas
- ▶ ...



WPF Control Hierarchy

▶ object

- DispatcherObject

- DependencyObject

- Freezable

- Visual

Has 2D representation, drawing etc.

- UIElement

Has routed events, layout, command bindings etc.

- FrameworkElement

Has styles, data binding, resources etc.

- Control

Has Control templates, Foreground, Background etc.

- Visual3D

- UIElement3D

- ContentElement

Parallel to **UIElement** (for Content)

- FrameworkContentElement

▶ See e.g.

- [Nathan, Chapter 3]

- <http://2000thingswpf.files.wordpress.com/2010/12/classhierarchy.png>

Agenda

- ▶ WPF = XAML + Code
- ▶ Controls and Layout
- ▶ **Resources**
- ▶ Styles and Triggers
- ▶ Animations

Introducing Logical Resources

- ▶ Logical resources can be defined in
 - `App.Resources` or
 - `FrameworkElement.Resources`
 - `FrameworkContentElement.Resources`
- ▶ Declare a logical resource with **x:Key** in the resource dictionary
- ▶ Resource Lookup
 - "Cascading" fashion
 - **StaticResource** or **DynamicResource**



Static and Dynamic Resources

- ▶ Two markup extensions
 - **StaticResource**
 - The resource is applied only once
 - **DynamicResource**
 - The resource is reapplied every time it changes

 - ▶ Programmatic access in code as well
 - Access Resources directly
 - `Lookup()`
 - `Add()`
 - `FindResource()`, `TryFindResource()`
 - `SetResourceReference()`
- static
dynamic



Resource Dictionaries

- ▶ The **Resources** property is in fact a **ResourceDictionary** object
- ▶ **ResourceDictionary**
 - Can be defined in separate XAML files
 - "Add ResourceDictionary"
 - Can be merged when needed
 - Merge rules apply
- ▶ System Resources
 - Use **x:Static** to refer
 - Make reference dynamic!
- ▶ Note
 - Resource dictionaries can be changed without recompiling the application by loading them dynamically with **XamlReader.Load()** in **System.Windows.Markup** and setting resources programmatically



Agenda

- ▶ WPF = XAML + Code
- ▶ Controls and Layout
- ▶ Resources
- ▶ **Styles and Triggers**
- ▶ Animations

Introducing Styles

- ▶ A style is basically a group of property values

- ▶ **Style**
 - **Setters**
 - (Property) Setter
 - Event Setter
 - **Triggers**
 - Property Trigger
 - Data Trigger
 - Event Trigger

Setters

- ▶ Styles can be defined as resource or set directly
 - Work for heterogenous elements
 - Can be overridden locally...
- ▶ **Setters**
 - (Property) **Setter**
 - Property, Value
 - **EventSetter**
 - Event, Handler



Implicit and Inherited Styles

- ▶ Styles can be set
 - Declaratively
 - Programmatically
- ▶ Styles can be defined
 - Explicitly
 - **Style** property
 - Implicitly
 - **Style.TargetType**
 - Inherited
 - **BasedOn**



Introducing Triggers

► **Style.Triggers** contain a "triggered" collection of setters and actions

- (Property)Trigger *WPF dependency properties*

- Property, Value

- Setters

- EnterActions

- ExitActions

- DataTrigger

CLR properties

- Binding, Value

- Setters

- EventTrigger

Event occurrences

- RoutedEvent

- Actions



Agenda

- ▶ WPF = XAML + Code
- ▶ Controls and Layout
- ▶ Resources
- ▶ Styles and Triggers
- ▶ **Animations**

Introducing Animations

- ▶ Animations
 - Can vary the value of dependency properties
- ▶ Three categories
 - Linear animations, i.e. *TypeName**Animation***,
 - **DoubleAnimation**
 - **ColorAnimation**
 - ...
 - Key frame-based animations
 - Path-based animations

Animation Basics

▶ Animation

- BeginTime
- Duration
- RepeatBehavior
- AutoReverse
- AccelerationRatio
- DecelerationRatio
- ...

"Forever", "3x", ...

▶ Linear animations

- From
- To, By

▶ Can programmatically be defined and started directly on controls



Storyboards

- ▶ **Storyboard** structures sets of animations in XAML
 - Attached **TargetName**
 - Attached **TargetProperty**
- ▶ Storyboards are controlled by **Trigger** actions in
 - Styles (as seen earlier)
 - Element Triggers
- ▶ Storyboard-related actions include e.g.
 - **BeginStoryboard**
 - **PauseStoryboard**
 - **ResumeStoryboard**
 - **StopStoryboard**
 - ...
- ▶ Note: "target" storyboard must be defined in same **Triggers** collection



More Triggers and Styles

- ▶ Storyboard actions can also be triggered by
 - (Property)Trigger
 - MultiTrigger
 - DataTrigger
 - MultiDataTrigger
- ▶ Common example: Triggers in styles



Easing Functions

- ▶ Easing Functions help creating good-looking animations in a pre-built manner
 - [http://msdn.microsoft.com/en-us/library/ee308751\(VS.100\).aspx](http://msdn.microsoft.com/en-us/library/ee308751(VS.100).aspx)
- ▶ **IEasingFunction**
 - QuadraticEase, CubicEase, QuarticEase, QuinticEase, PowerEase
 - BackEase
 - BounceEase
 - CircleEase
 - ElasticEase
 - ExponentialEase
 - SineEase
- ▶ **EasingMode**
 - EaseIn default
 - EaseOut
 - EaseInOut



Summary

- ▶ WPF = XAML + Code
- ▶ Controls and Layout
- ▶ Resources
- ▶ Styles and Triggers
- ▶ Animations



WINCUBATE

Jesper Gulmann Henriksen

PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31

Email : jgh@wincubate.net

WWW : <http://www.wincubate.net>

Hasselvangel 243

8355 Solbjerg

Denmark