BUILDING XAML/C# APPLICATIONS FOR DESKTOP, STORE, AND PHONE

Philip Japikse (@skimedic)
skimedic@outlook.com
www.skimedic.com/blog
Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, CSP
Principal Architect, InterKnowlogy

Delivering the Art of Software
About()

- Principal Architect, InterKnowlogy, Inc.
- http://www.interknowlogy.com
- Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, CSP
- Founder, Agile Conferences, Inc.
- President, Cincinnati .NET User’s Group
- Co-host, Hallway Conversations
  - www.hallwayconversations.com
INTERKNOWLOGY

• Training – XAML (WPF, Windows 8.1, Windows Phone 8.1), Agile
• Natural User Interfaces (NUI) - Incorporating Voice, Touch, and Gesture
• Interaction Design (IxD) - Designing software to actual users
• Enterprise Mobile Apps (EMA)
  • Expanding your brand, Increasing Productivity
• App Renovation (AR) - Add NUI and IxD to existing software

• Contact me at philj@interknowlogy.com
AGENDA

• XAML Basics
• Layouts
• Styles, Resources, Behaviors
• Binding & Commands

• Animations and Transforms
• MVVM
• Windows 8.1
• Windows Phone
WHAT IS WPF?

• Windows Presentation Foundation
• Next Generation Rich User Application Platform
  • Declarative UI (XAML)
• Based on DirectX (not GDI/GDI+)
  • Vector Based
• Supports true Hardware Acceleration
• Resolution Independence
• Leaps ahead with binding
• Data and other Elements
WHAT ELSE IS IN WPF?

• Flexible flow layout model
• Styles and Templates
• OOTB document handling features
• True animation – not timer based
• A/V support (through WMP)
• Commands
• Routed Events
WHAT’S NEW IN .NET 4?

• DataGrid, DatePicker, Calendar
• Visual State Manager
• Layout Rounding
• Multi-Touch support
• Custom Dictionaries
• Behaviors*
• Animation Easing
• Binding String Format
WHAT’S NEW IN .NET 4.5

• Ribbon Control
• Improved Performance with large data
• Binding to Static Properties
• Accessing Collections on non-UI Threads
• Asynchronous Validation
• Delay Updating Data Source
• Retrieving Data Binding Info from Expression
• And more…

• http://tinyurl.com/wpf4-5
WPF ROADMAP

• http://bit.ly/wpfroadmap
• Performance
  • App startup, scrolling and virtualization
• DirectX Interoperability
  • Seamless integration with future versions of DirectX
• Modern Hardware Support
• Tooling
  • Visual diagnostics, Timeline tools, Blend improvements
XAML BASICS
WHAT IS XAML

- eXtensible Application Markup Language
- Based on XAML
- XAML Documents define the arrangement of:
  - Panels, Buttons, Controls, UIElements
- Used to create UIs for
  - Windows Presentation Foundation (WPF)
  - Windows 8.x apps (WIN8)
  - Windows Phone apps (WP)
  - Windows Workflow Foundation (WF)
XAML BENEFITS

- Declarative User Interface
- Separation of Design/Markup from programming
- Flexible flow layout model
- Styles and Templates
- Leaps ahead with binding
- Data and other Elements
- Styles and Templates
- True storyboard-based animations
THE FOUNDATION

- Every XAML elements maps to a .NET class
- Elements can be nested
- Properties can be set through:
  - Attributes
  - Tag content (for the Content property)
  - Nested tags (with special syntax)
XAML DOCUMENTS

• Can only have one top level element
• Typically
  • Window (WPF)
  • Page (WIN8, WP8)
  • UserControl
• Application
MENUS

• Use “_” instead of “&” for Hot-Key
• Beware for XAML order vs Grid Order
• Similar to WinForms except
  • MenuItem is a container control
  • Use <Separator> instead of “-”
XAML *IS* XML AT ITS CORE

• All tags must be closed
  • <Grid> <!-- stuff here --> </Grid>
  <Image/>

• Special Characters must be escaped
  • &lt; &gt; &amp; &quot;

• Elements can have attributes
  • <Image Source="Assets/Logo.png" />

• Elements can have nested tags
  • <Image>
    <Image.Source>Assets/Logo.png</Image.Source>
  </Image>
ELEMENT NAMING

• Named elements can be referred to in code
• Full intellisense support
• Two ways of naming
  • XAML Name
    • x:Name="Foo"
  • Element Name Property
    • Name="Foo"
• Both generate the same code
  • internal System.Windows.Controls.Image Foo;
<Window x:Class="WPFDemo.Window1“

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

Title="Window1" Height="300" Width="300">

<Grid>

</Grid>

</Window>
NAMESPACE

• Microsoft namespaces do not map to specific .NET namespaces
• Creators of XAML combined them under /presentation

• Your namespaces will
  • xmlns:Prefix="clr-namespace:Namespace[;assembly=AssemblyName]"

• Two prefixes that are “reserved” (by convention)
  • xmlns:sys="clr-namespace:System;assembly=mscorlib"
  • xmlns:local="clr-namespace:WPFDemo"
PROPERTIES

• Simple properties
  • Set with Attributes or as tag content
  • Always set as a string
  • Converted using a typeconverter
  • Declared on the property or the class

• Complex Properties
  • Set with property-element syntax and nested elements
  • `<Grid ShowGridLines="True" UseLayoutRounding="True">`<br>    `<Grid.RowDefinitions>`<br>    `<RowDefinition/>`
MARKUP EXTENSIONS

• Used to set properties that can’t be set any other way
• Dynamically setting properties
• Setting to other elements
• Example:
  • Foreground="{x:Static SystemColors.ActiveCaptionBrush}"
  • <x:Static Member="SystemColors.ActiveCaptionBrush"/>
ATTACHED PROPERTIES

- Properties defined in one element used in another
  - `Grid.Row/Grid.Column`
- Translated into method calls
  - `Grid.SetRow(control,x)`
DEPENDENCY PROPERTIES

• Use more efficient storage
• Support additional features
  • Change notification
  • Dynamic value resolution
• Required for key WPF/WIN8/WP8 features
  • Animations
  • Databinding
  • Styles
• Wrapped with “normal” .NET properties
FROM XAML TO EXECUTABLE

- InitializeComponent()
  - In the code behind file
  - THIS MUST STAY IN

- Compiles to intermediate file
  - <filename>.g.cs

- Gets added to the .exe
TOOLS FOR CREATING XAML APPS

• Visual Studio
  • 2013 needed for Universal Apps
• Blend
  • Now free with Visual Studio
• Your favorite editor
  • Anything you can do in XAML you can do in code
Questions?
LAYOUTS AND LAYOUT CONTAINERS
LAYOUT

• “Web”-like layout model with Flexible Flow
  • Gone are explicit sizing, absolute positioning, etc
  • Containers try to allocate element’s requested space

• Layout Process
  • Measure
    • Loop through all elements asking for preferred size
  • Arrange
    • Container places elements in appropriate position
LAYOUT CONTAINERS

- All derive from the Panel class
- Standard
  - Grid, UniformGrid
    - Most commonly used (default Window Panel)
  - StackPanel, WrapPanel, DockPanel
    - Used for UI Tweaks
- Canvas
  - Absolute (think WinForms) positioning
  - InkCanvas - Supports Digital Ink
LAYOUT PROPERTIES

• Alignment (Horizontal || Vertical)
  • Center, Left/Top, Right/Bottom, Stretch
• Margin || Padding
  • Left, Top, Right, Bottom
• MinWidth/MinHeight
  • Requested Minimum dimension
• MaxWidth/MaxHeight
  • Maximum dimensions (trumps Stretch)
• Width/Height – (note: shouldn’t really do this)
  • Dimensions (trumps Stretch, trumped by Min/Max settings)
• ActualHeight/ActualWidth (read only)
MEASURE AND ARRANGE

- Containers check (in order):
  - Minimum Size
  - Maximum Size
  - The content
  - Size of the container (clipping if necessary)
  - Alignment
BORDER

• Not a true layout panel
• Provides:
  • Border
  • Background
  • Rounded Corners

```xml
<Border Grid.Row="1" Background="Aqua"
CornerRadius="45" BorderBrush="BlueViolet">
  <Grid>
  
  </Grid>
</Border>
```
STACK PANEL

• Stacks children in a single row or column
• Typically used nested in another panel

```xml
<StackPanel Name="MyStackPanel"
Orientation="Vertical" >
  <!-- Children -->
</StackPanel>
```
WRAP PANEL

• Stacks children in row or column
• Wraps children around who don’t fit on one row
• Typically used nested in another panel

```
<WrapPanel Name="MyWrapPanel" Orientation="Horizontal">
  <!-- Children -->
</WrapPanel>
```
DOCK PANEL

- Stretches controls against its edges
- Top, Right, Bottom, Left
- Use LastChildFill to fill the middle

```xml
<DockPanel LastChildFill="True">
  <Button DockPanel.Dock="Top">First</Button>
  <Button DockPanel.Dock="Left">Second</Button>
  <Button DockPanel.Dock="Right">Third</Button>
  <Button DockPanel.Dock="Bottom">Fourth</Button>
  <Button DockPanel.Dock="Bottom">Fifth</Button>
  <Button Name="button5">Last</Button>
</DockPanel>
```
CANVAS

• Only use for special use cases
• Ink (w/ InkCanvas)
• Games

```xml
<Canvas>
    <Button Content="Button"
        Canvas.Left="80" Canvas.Top="42"
        Width="75"/>
    <Button Content="Button"
        Canvas.Left="63" Canvas.Top="214"
        Width="75"/>
    <Label Content="Label"
        Canvas.Left="155" Canvas.Top="121"/>
</Canvas>
```
GRID

- Most powerful and versatile layout
- Default layout panel when you add a new window
- Carves up the window into sections for other layout containers
- Convention is to place one element in each cell
- Can be split with the GridSplitter
GRID

• Three ways to size
  • Proportional
  • Automatic
• Can span rows and columns
• Use Layout Rounding

```xml
<Grid UseLayoutRounding="True">
  <Grid.RowDefinitions>
    <RowDefinition Height="100" />
    <RowDefinition Height="*" />
    <RowDefinition Height="Auto" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="Auto" />
  </Grid.ColumnDefinitions>
  <Label Grid.Row="0" Grid.Column="0">Home:</Label>
  <TextBox Grid.Column="1" Margin="3,0"
           Grid.ColumnSpan="3" />
</Grid>
```
SHARED SIZE GROUPS

- Ensures that Rows or Columns in different grids stay the same size
NESTING LAYOUT CONTAINERS

• Layout Panels can (and should) be nested
• Combine layouts to create compelling user interfaces
  • Typically will start with a Grid
• Remember to keep the flow flexible
• Avoid hard coding size (except for main margins)
NESTED EXAMPLE

- Create a Dialog by combining StackPanel and DockPanel

```xml
<DockPanel LastChildFill="True">
  <StackPanel DockPanel.Dock="Bottom"
              HorizontalAlignment="Right"
              Orientation="Horizontal">
    <Button Content="OK"/>
    <Button Content="Cancel"/>
  </StackPanel>
  <TextBox DockPanel.Dock="Top">
    Here's the Dialog
  </TextBox>
</DockPanel>
```
DEMO

Layouts & Containers
ELEMENTS, CONTROLS, AND MORE
ELEMENTS & CONTROLS

• Everything is an Element
  • Controls are elements that can:
    • Receive Focus, Accept User Input
• Content Controls
  • Hold a single element
    • Window, Page, UserControl (Special)
    • Labels, Buttons, CheckBox, RadioButton, ToolTips
    • ScrollViewer, Border, Expander, etc
• Text Controls
  • TextBox, PasswordBox – Strings Only
  • RichTextBox – Sophisticated Content – FlowDocument
LIST CONTROLS

- List Controls (ItemsControl)
- Selectors
  - ListBox, ComboBox, TabControl
- Non-Selectors
  - Menus, ListView, GridView, TreeView, ToolBar
- Range Based Controls
  - ScrollBar, ProgressBar, Slider
CONTENT CONTROLS

• Can only hold one element
• Element can be a layout control
• Use this wisely…
SPECIALIZED CONTENT CONTROLS
SCROLLVIEWER

• Used to fit large amounts of data in a tight area
• Typically used to wrap a layout panel

<ScrollViewer VerticalScrollBarVisibility="Auto" HorizontalScrollBarVisibility="Hidden" CanContentScroll="True" />
</ScrollViewer>
GROUPBOX

- Often used with RadioButtons
- Still needs a Panel to layout its controls

```xml
<GroupBox Header="GroupBox">
  <StackPanel>
    <RadioButton/>
    <RadioButton/>
    <RadioButton/>
    <RadioButton/>
  </StackPanel>
</GroupBox>
```
EXPANDER

• Wraps content that user can show or hide

```xml
<Expander Header="Region One">
  <Button/>
</Expander>
<Expander Header="Region Two" IsExpanded="True">
  <TextBox>
  </TextBox>
</Expander>
<Expander Header="Region Three">
  <Button>Hidden Button Two</Button>
</Expander>
```
LIST CONTROLS
LIST CONTROLS

- Add data to the Items property
- Bind to the ItemsSource property
- The item controls derive from ContentControl
  - ListBoxItem, TabItem
- Selectors have SelectItem and SelectedIndex
  - ListBox, ComboBox, TabControl
- Containers don’t support selection
  - Menus, ToolBars, Trees
LISTBOX, COMBOBOX

• Support
• SelectedItem
• SelectedIndex
• Can build complex lists through nested controls
• Better with Binding and Styles
• Coming later today

```xml
<ListBox Grid.Row="0">
  <ListBoxItem>
    <StackPanel Orientation="Horizontal">
      <CheckBox />
      <Label Content="First Item"/>
    </StackPanel>
  </ListBoxItem>
</ListBox>
```
TABCONTROL

• List control for TabItems
• Tab string can be placed on any side
• TabItem is a container control

```xml
<TabControl TabStripPlacement="Left">
  <TabItem Header="Scrolling"
    IsSelected="False">
    <!-- Content -->
  </TabItem>
  <TabItem Header="Expander"
    IsSelected="True">
    <!-- Content -->
  </TabItem>
</TabControl>
```
Controls
QUESTIONS?
XAML RESOURCES

• Reusable markup stored in a resource collection
  • Think function or stored procedure
• Provide for consistency throughout your app
  • E.g. Font Family, size, style
• UI Elements
• Allows for updates at runtime
  • E.g. Skinning your app, Localization
• Separate design from code
DEFINING RESOURCES

• Resources can be defined at many levels
  • Controls
  • Page
  • Application

• Controls look up the Element Tree to find resources
  • Must be declared in the XAML *before* they are called if static

• Resources by default are shared
  • Change to non shared with x:Shared="False"
DEFINING AND USING RESOURCES

<Window.Resources>
    <ImageBrush x:Key="TileBrush" TileMode="Tile"
                ViewportUnits="Absolute" Viewport="0 0 32 32"
                ImageSource="happyface.jpg" Opacity="0.3"></ImageBrush>
</Window.Resources>

<Button Background="{StaticResource TileBrush}" Margin="5" Padding="5" FontWeight="Bold" FontSize="14">
    A Tiled Button
</Button>
STATIC VS DYNAMIC RESOURCES

- Static Resources are applied when the element is first rendered
  - Must be defined before referenced
  - Changes don’t affect controls unless the resources derive from Freezable (e.g. Brushes)

- Dynamic Resources look up the details every time they are needed
  - Can change entire object at runtime
  - *Can* add significant overhead
SYSTEM RESOURCES

• Used to get system settings and apply them to your application
  • System.Windows is for WPF, System.Drawing is for WinForms
• SystemFonts
  • Provides access to user’s font settings
• SystemColors
  • Provides access to user’s color settings
• SystemParameters
  • Wraps list of settings, including standard sizes, keyboard & mouse settings, and status of graphical effects such as drop shadows
DEFINING AND USING SYSTEM RESOURCES

• Use Dynamic
  • Otherwise changes won’t get picked up by your application
• The resources are system defined, so use x:Static

  <Label Foreground="{DynamicResource {x:Static SystemColors.WindowTextBrushKey}}">
    Ordinary text
  </Label>
RESOURCE DICTIONARIES

• Allow for sharing resources between projects/assemblies
• Resource Dictionary is a XAML document
  
  <ResourceDictionary
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

• Resource Dictionaries are key to localization
• To share between applications and load with markup
• Place resource dictionary in a file called generic.xaml
USING RESOURCE DICTIONARIES

• Must be merged into a resource collection in your application

<Application.Resources>
  <ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>
      <ResourceDictionary Source="AppBrushes.xaml"/>
      <ResourceDictionary Source="WizardBrushes.xaml"/>
    </ResourceDictionary.MergedDictionaries>
  </ResourceDictionary>
</Application.Resources>
Resources
XAML STYLES

• Conceptually the same as CSS styles
  • Enables applying design choices consistently throughout the app or page
  • Can work automatically across the app or target specific types
  • Can set any dependency property
  • Support triggers
• Can be defined at the Control, Page, or Application level
STYLE ELEMENTS

• Setters
  • Collection of Setter or EventSetter objects that set dependency property values or event handlers

• Triggers
  • Allow changing styles based on an external value

• Resources
  • Resource Collection to be used with the style

• BasedOn
  • Provides way to derive from another style

• TargetType
  • If set, the style is automatically applied to the type specified.
**STYLE EXAMPLES**

```xml
<Style x:Key="BigFontButtonStyle">
<Setter Property="Control.FontFamily" Value="Times New Roman" />
<Setter Property="Control.FontSize" Value="18" />
<Setter Property="Control.FontWeight" Value="Bold" />
</Style>

<Style x:Key="BigFontButtonStyle">
<Setter Property="Button.FontFamily" Value="Times New Roman" />
<Setter Property="Button.FontSize" Value="18" />
<Setter Property="Button.FontWeight" Value="Bold" />
</Style>

<Style x:Key="BigFontButtonStyle" TargetType="Button">
<Setter Property="FontFamily" Value="Times New Roman" />
<Setter Property="FontSize" Value="18" />
<Setter Property="FontWeight" Value="Bold" />
</Style>
```
TRIGGERS

• Allow for declaratively handling style changes based on dependency property values
• Dependency Property Triggers
  • Trigger
  • MultiTrigger
• Binding Triggers
  • DataTrigger
  • MultiDataTrigger
• Event Triggers - Animations
  • EventTrigger
TRIGGERS IN ACTION

• Triggers wait for the condition to be true
• Apply the correct value
• When condition is false
• Resets the value
• Last Trigger wins
• In XAML order

```xml
<Style x:Key="BigFontButton">
    <Style.Setters>
        ...
    </Style.Setters>
    <Style.Triggers>
        <Trigger Property="Control.IsFocused" Value="True">
            <Setter Property="Control.Foreground" Value="DarkRed" />
        </Trigger>
        <Trigger Property="Control.IsMouseOver" Value="True">
            <Setter Property="Control.Foreground" Value="LightYellow" />
            <Setter Property="Control.FontWeight" Value="Bold" />
        </Trigger>
        <Trigger Property="Button.IsPressed" Value="True">
            <Setter Property="Control.Foreground" Value="Red" />
        </Trigger>
    </Style.Triggers>
</Style>
```
MULTITRIGGERS

• All conditions must be met before the trigger will call the setters
• Since all must be true, XAML order doesn’t matter

```xml
<Style x:Key="BigFontButton">
  <Style.Setters>
    ...
  </Style.Setters>
  <Style.Triggers>
    <MultiTrigger>
      <MultiTrigger.Conditions>
        <Condition Property="Control.IsFocused" Value="True"/>
        <Condition Property="Control.IsMouseOver" Value="True"/>
      </MultiTrigger.Conditions>
      <MultiTrigger.Setters>
        <Setter Property="Control.Foreground" Value="DarkRed"/>
      </MultiTrigger.Setters>
    </MultiTrigger>
  </Style.Triggers>
</Style>
```
Styles
EVENTTRIGGERS

• Launch an animation based on an event
• Event Triggers need to be reversed
• Triggers can fire EventTriggers
• Useful for launching animations

```xml
<Style x:Key="BigFontButtonStyle">
    ...
    <Style.Triggers>
        <EventTrigger>
            <EventTrigger.RoutedEventArgs>
                <BeginStoryboard>
                    <Storyboard>
                        <DoubleAnimation ... />
                    </Storyboard>
                </BeginStoryboard>
            </EventTrigger.RoutedEventArgs>
        </EventTrigger>
        <EventTrigger>
            <EventTrigger.RoutedEventArgs>
                <Mouse.MouseLeave>
                    ...
                </Mouse.MouseLeave>
            </EventTrigger.RoutedEventArgs>
        </EventTrigger>
    </Style.Triggers>
</Style>
```
BEHAVIORS
BEHAVIORS

• Introduced in Expression Blend v3
• Encapsulate functionality into reusable components
  • Drag & Drop
  • Pan and Zoom
  • Input Validation
  • Watermark Text
  • InvokeCommand
• Additional Behaviors are available from the Expression Gallery
  • http://msdn.microsoft.com/en-us/expression/jj873995
Behaviors
Questions?
BINDING IN XAML

• Set a target dependency property to the value of a public property on a source object

• Dependency properties can bind to
  • Other UIElements
  • Data objects
  • Lists

• Binding failures are “silent”
BINDING MODES

- **OneWay**
  - Target property is updated when the source property changes

- **TwoWay**
  - Target property is updated when the source changes and the source property is updated when the target property changes

- **OneTime**
  - OneWay that only fires when the XAML is rendered

- **OneWayToSource**
  - OneWay in reverse

- **Default**
  - Base on the target property. User settable are two way.
BINDING UPDATES

• PropertyChanged
  • Updates when the property is changed
  • Default for checkbox, radiobutton, etc.

• LostFocus
  • Updates when the property loses focus
  • Default for a text box

• Explicit
  • Only updates when called through code

• Default
SETTING THE BINDING TARGET

• ElementName
  • UIElement in the same XAML tree

• Source
  • Points directly to a source object

• RelativeSource
  • Used to locate another object in a relative position to the element being bound
  • Used mainly in data templates

• DataContext
  • Set the source for multiple elements in one statement
THE BINDING EXPRESSION

- Starts with Binding
- Then can contain
  - Target
  - Path
  - Mode
  - UpdateSourceTrigger
  - TargetNullValue
  - Delay

Text={Binding ElementName=Foo,
Path=Value, Mode=TwoWay,
UpdateSourceTrigger=
    PropertyChanged,
TargetNullValue=[N/A]}

Delivering the Art of Software
SETTING BINDINGS IN CODE

- Everything you can do in XAML you can do in code
- Create a Binding object
- Set the properties
- Call SetBinding on the Element
- Pass in the Binding and the DependencyProperty

```csharp
Binding binding = new Binding();
binding.Source = <some object>;
binding.Path = <some property>;
binding.Mode = BindingMode.TwoWay;
myControl.SetBinding(
    TextBox.Text, binding);
```
GETTING BINDING INFO FROM CODE

- New in WPF 4.5
- Call GetBinding on BindingOperations
- Pass in
  - Element name
  - Dependency Property
- Can also call GetBindingExpression
- Returns the ResolvedSource property

```csharp
BindingOperations.
    GetBinding(  
        myTextBox,TextBox.Text);

BindingOperations.
    GetBindingExpression(  
        myTextBox,TextBox.Text);
```
ELEMENT BINDING

• Allows for powerful UI constructs all defined in XAML
• Can bind elements value directly to another elements
  • No more coding “onChanged” events!
• Example: Master Detail layouts
ELEMENT BINDING

- Target property doesn’t know it’s being bound to
- Multiple bindings can be created

```xml
<Label Grid.Row="4" Name="label1" Content="{Binding
    ElementName=textBox1,
    Path=Text, Mode=TwoWay,
    UpdateSourceTrigger=PropertyChanged}"
    FontSize="{Binding
        ElementName=FontSlider,
        Path=Value, Mode=TwoWay}"
    Foreground="{Binding
        ElementName=lstColors,
        Path=SelectedItem.Tag, Mode=TwoWay}">
</Label>
```
DATABINDING
BINDING TO OBJECTS

- Can bind to
  - Static objects
  - Objects defined as resources
  - Any object that is “reachable”

```xml
Font="{Binding
    Source={x:Static
        SystemFonts.IconFamily},
    Path=Source}"
```
RELATIVE SOURCE

• Select mode
• Self
• FindAncestor
• Select Type
• Select Level
• PreviousData
• TemplatedParent

Text = "{Binding Path=Title,
RelativeSource=
{RelativeSource
    FindAncestor,
    AncestorType= {x:Type Window},
    AncestorLevel=2}}"
DATACONTEXT

- Sets the target for all child elements
- If target isn’t set on an element, it will look up the tree to find the first DataContext setting

DataContext="{Binding ElementName=cboProducts, Path=SelectedItem}"
DATABINDING

• Use Observables
  • Implement INotifyPropertyChanged
  • Use ObservableCollection
• Use Validation
  • INotifyDataErrorInfo
• Best to bind to data objects
  • Use an ORM (like EF) to get the data
  • Return an object graph via MVVM (more on this later today)
OBSERVABLES

• Models
  • Leverage INotifyPropertyChanged
  • Beware of magic strings

• Collections
  • Leverage ObservableCollections
  • Implements
    • INotifyCollectionChanged
    • INotifyPropertyChanged
INOTIFYPROPERTYCHANGED (PRIOR TO 4.5)

```csharp
public class Product : INotifyPropertyChanged
{
    private string _modelName;
    public string ModelName
    {
        get { return _modelName; }
        set
        {
            if (_modelName == value) return;
            _modelName = value;
            OnPropertyChanged(FieldNames.ModelName);
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
    private void OnPropertyChanged(Enum fieldName)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(string.Empty));
            //PropertyChanged(this, new PropertyChangedEventArgs(fieldName.ToString()));
        }
    }
}
```
public class Product : INotifyPropertyChanged
{
    private string _modelName;
    public string ModelName
    {
        get { return _modelName; }
        set
        {
            if (_modelName == value) return;
            _modelName = value;
            OnPropertyChanged();
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
    private void OnPropertyChanged([CallerMemberName] string fieldname = "")
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(string.Empty));
            //PropertyChanged(this, new PropertyChangedEventArgs(fieldName));
        }
    }
}
```csharp
public class ProductList : IProductList
{
    private readonly IList<Product> _products;
    public ProductList(IList<Product> products)
    {
        _products = products;
    }
    public void Add(Product item)
    {
        _products.Add(item);
        notifyCollectionChanged(new NotifyCollectionChangedEventArgs(NotifyCollectionChangedAction.Add, item));
    }
    public void Clear()
    {
        _products.Clear();
        notifyCollectionChanged(new NotifyCollectionChangedEventArgs(NotifyCollectionChangedAction.Reset));
    }
    // Ommitted for brevity
    public event NotifyCollectionChangedEventHandler CollectionChanged;
    private void notifyCollectionChanged(NotifyCollectionChangedEventArgs args)
    {
        if (CollectionChanged != null) { CollectionChanged(this, args); }
    }
}
```
OBSERVABLE COLLECTIONS – THE EASY WAY

- Use the built in ObservableCollection class
- Constructor takes an IEnumerable

```csharp
Products = new ObservableCollection<Product>(new ProductService().GetProducts());
```
WHERE TO IMPLEMENT INotifyPropertyChanged FOR MVVM?

- Anemic Model
  - Implemented in ViewModel
  - “Cleaner” model
  - Lots of code duplication
- Anemic ViewModel
  - Implemented in Model
  - Less code duplication
  - Mixing of concerns?
Observables
VALIDATION
VALIDATION METHODS

• Raise an error on your data object
  • Normally errors are ignored on bindings
  • Set NotifyOnValidationError
• Use INotifyDataErrorInfo or IDataErrorInfo
  • INotifyDataErrorInfo is new in WPF 4.5
• Define validation at the binding level
  • Note: Validation only occurs with TwoWay or OneWayToSource binding modes
VALIDATESONEXCEPTION

• Raises exceptions from bound property
• Sets Validation.HasError = True
• Creates ValidationError object
• If NotifyOnValidationError = true
  • WPF raises Validation.Error
IDATAERRORINFO

• Add an string indexer for the Item property
• Error property is not used by WPF
• Relies on INotifyPropertyChanged
• Requires ValidatesOnDataErrors in binding
public class Product : IDataErrorInfo
{
    public string this[string columnName]
    {
        get
        {
            var field = (FieldNames)Enum.Parse(typeof(FieldNames), columnName);
            switch (field)
            {
                case FieldNames.Inventory:
                    if (Inventory < 0) { return "Inventory can not be less than zero"; }
                    break;
                case FieldNames.Price:
                    if (Price < 0) { return "Price can not be less than zero"; }
                    break;
            }
            return string.Empty;
        }
    }

    public string Error { get { throw new NotImplementedException(); } }
}
LEVERAGING VALIDATION IN XAML

<TextBox Text="{Binding ElementName=ProductSelector, Path=SelectedItem.ModelName, ValidatesOnExceptions=true, ValidatesOnDataErrors=true}" />

<Style.Triggers>
  <Trigger Property="Validation.HasError" Value="true">
    <Setter Property="Background" Value="Pink" />
    <Setter Property="Foreground" Value="Black" />
  </Trigger>
</Style.Triggers>
<Setter Property="Validation.ErrorTemplate">
  <Setter.Value>
    <ControlTemplate>
      <!-- Omitted for brevity -->
    </ControlTemplate>
  </Setter.Value>
</Setter>
INOTIFYDATAERRORINFO

• New in WPF 4.5
• Contains
  • ErrorsChanged event
  • HasError Boolean
  • GetErrors() method to return the errors
• Relies on INotifyPropertyChanged
private Dictionary<string, List<string>> errors = new Dictionary<string, List<string>>();
public event EventHandler<DataErrorsChangedEventArgs> ErrorsChanged;

public bool HasErrors { get { return (errors.Count > 0); } }
public IEnumerable GetErrors(string propertyName)
{
    if (propertyName.IsNullOrEmpty()) { return errors.Values; }
    else {
        if (errors.ContainsKey(propertyName)) { return (errors[propertyName]); }
        else { return null; }
    }
}
private void RaiseErrorsChanged(string propertyName)
{
    if (ErrorsChanged != null) {
        ErrorsChanged(this, new DataErrorsChangedEventArgs(propertyName));
    }
}
private void SetErrors(List<string> propertyErrors, [CallerMemberName] string propertyName = "")
{
    errors.Remove(propertyName);
    errors.Add(propertyName, propertyErrors);
    RaiseErrorsChanged(propertyName);
}

private void AddError(string error, [CallerMemberName] string propertyName = "")
{
    if (!errors.ContainsKey(propertyName)) { errors.Add(propertyName, new List<string>()); }
    if (!errors[propertyName].Contains(error))
    {
        errors[propertyName].Add(error);
        RaiseErrorsChanged(propertyName);
    }
}

private void ClearErrors([CallerMemberName] string propertyName = "")
{
    errors.Remove(propertyName);
    RaiseErrorsChanged(propertyName);
}
public decimal Price
{
    //Getter Omitted
    set
    {
        if (_price == value)
            return;
        _price = value;
        if (Price < 0)
        {
            var errors = new List<string>() { "Price can not be less than zero" };  
            SetErrors(errors);
        }
        else
        {
            ClearErrors();
        }
        OnPropertyChanged();
    }
}
CUSTOM VALIDATION RULES

• Derive from ValidationRule
• Add additional properties for custom configuration
• Override Validate()
  • Return new ValidationResult(true || false, [Error Message])
• Add to Binding.ValidationRules collection
VALUE CONVERSION
CONVERTING DATA

• String Formatting
  • Convert data represented as text
  • Use Binding.StringFormat property

• Value Converters
  • Convert any type of data to any other type of data
  • Use
    • IValueConverter
    • IMultiValueConverter (WPF only)
STRING FORMATTING

• Use the standard .NET formatting strings
• Escape with {}
  • Text="{Binding Path=SalePrice, StringFormat={0:C}}"
• Or use a string literal to preface the format
  • Text="{Binding Path=SalePrice, StringFormat=SalePrice{0:C}}"
• When editing, must clear out the string literal
STANDARD FORMATTERS

- Currency: C
- Scientific: E
- Percentage: P
- Fixed Decimal: Fx

- Short Date: d
- Long Date: D
- Long Date, Short Time: f
- Long Date, Long Time: F
VALUE CONVERTERS

• Converts source data just before it’s displayed and converts back just before its applied to the source
  • Similar to Parse and Format methods in Windows Forms
• Can create custom objects
  • E.g. Command Parameters
• Can alter properties on elements
  • E.g. Background Color
public class LoginMultiConverter : IMultiValueConverter
{
    public object Convert( object[] values, Type targetType,
                               object parameter, System.Globalization.CultureInfo culture)
    {
        var param = new LoginParameter();
        //Omitted for brevity
        return param;
    }

    public object[] ConvertBack(object value, Type[] targetTypes,
                                 object parameter, System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
<Button Command="{Binding Path=LoginCmd}">
  <Button.CommandParameter>
    <MultiBinding Converter="{StaticResource LoginMultiConverter}">
      <MultiBinding.Bindings>
        <Binding Path="UserName" />
        <Binding Path="Password" />
        <Binding ElementName="pgLogin" />
      </MultiBinding.Bindings>
    </MultiBinding>
  </Button.CommandParameter>
</Button>
Value Conversion
BINDING TO COLLECTIONS
THE BASICS

• All controls that derive from ItemsControl can bind to a collection
  • ListBox, ComboBox, ListView, DataGrid, Menu, TreeView

• Three key properties (more later today)
  • ItemsSource
  • DisplayMemberPath
  • ItemTemplate

• Can bind to anything that derives from IEnumerable
SIMPLE BINDING

- Set
  - ItemsSource
- Then either
  - Set
  - DisplayMemberPath
- Or Override
  - ToString()

```csharp
products = FakeRepo.GetAllProducts();
lstProducts.ItemsSource = products;
```
OVERRIDING TOSTRING()

• Convenient if you just want to combine some properties together
• But –
  • No flexibility
  • Limited formatting

```csharp
public override string ToString()
{
    return ModelName + " (" + UnitCost.ToString() + ")";
}
```
LIST STYLES

• Provide mechanism for more advanced rendering of list items
• ItemContainerStyle
  • Uses Setters and Triggers to alter
    • DisplayMemberPath
• ItemTemplate (next section)
ITEMCONTAINERSTYLE

- Each list control has an ItemContainer
- ListBoxItem
- ComboBoxItem
- Etc.

```xml
<ListBox Name="lstProducts" Margin="5"
    DisplayMemberPath="ModelName">
    <ListBox.ItemContainerStyle>
        <Style TargetType="{x:Type ListBoxItem}">
            <Setter Property="Background"
                Value="LightSteelBlue" />
            <Setter Property="Margin"
                Value="5" />
            <Style.Triggers>
                <Trigger Property="IsSelected"
                    Value="True">
                    <Setter Property="Background"
                        Value="DarkRed" />
                    <Setter Property="Foreground"
                        Value="White" />
                </Trigger>
            </Style.Triggers>
        </Style>
    </ListBox.ItemContainerStyle>
</ListBox>
```
UPDATING THE ITEMTEMPLATES

• Create new ControlTemplate for the Item
  • <ControlTemplate TargetType="{x:Type ListBoxItem}">

• Add Content Controls
  • <RadioButton Focusable="False" IsChecked="{Binding Path=IsSelected, RelativeSource={RelativeSource TemplatedParent}, Mode=TwoWay}">

• Add ContentPresenter to get original content
  • <ContentPresenter> </ContentPresenter>
SETTING THE CONTROL TEMPLATE

• Setting the ControlTemplate like any other property
• For reuse, place into a resource

```xml
<Setter Property="Template">
  < Setter.Value >
    <ControlTemplate
      TargetType="{x:Type ListBoxItem}"
    >
      <RadioButton ...
        >
        <ContentPresenter>
        </ContentPresenter>
      </RadioButton>
    </ControlTemplate>
  </Setter.Value>
</Setter>
```
SETTING ALTERNATING ITEM STYLE

• Set AlternationCount in the ListControl
  • <ListBox … AlternationCount="2">

• Add Trigger for AlternationIndex in the Style
  • <Trigger Property="ItemsControl.AlternationIndex" Value="1">

• AlternationIndex gets trumped by IsSelected
STYLE SELECTORS

• Can only be created in code
• Derive from StyleSelector
  • Override SelectStyle(object item, DependencyObject container)
• Styles must be accessible through code
  • Either Control, Windows, or Global resources
• Better: Create properties for the styles on the StyleSelector
DATA TEMPLATES

• XAML Used to represent an item in the list
• ControlTemplate used for Content property
• ItemTemplates used for ItemsControls
• Based on ControlTemplates
• Each item is wrapped in an ItemControl
SAMPLE DATA TEMPLATE

```xml
<ListBox.ItemTemplate>
  <DataTemplate>
    <Grid Margin="0" Background="White">
      <Border Margin="5" BorderThickness="1" BorderBrush="SteelBlue" CornerRadius="4">
        <HorizontalAlignment="Stretch" Background="{Binding Path=Background, RelativeSource={RelativeSource Mode=FindAncestor,AncestorType={x:Type ListBoxItem}}}">
          <Grid Margin="3" Width="Auto" ShowGridLines="True" HorizontalAlignment="Stretch">
            <TextBlock FontWeight="Bold" Width="Auto" Text="{Binding Path=ModelNumber}"></TextBlock>
            <TextBlock Grid.Row="1" Width="Auto" Text="{Binding Path=ModelName}"></TextBlock>
            <TextBlock Grid.Row="2" Width="Auto" Text="{Binding ElementName=lbProducts,Path=ActualWidth}"/>
          </Grid>
        </Border>
      </Grid>
    </DataTemplate>
  </ListBox.ItemTemplate>
```
Collection Binding
HANDLING LARGE LISTS
VIRTUALIZATION

• UI Virtualization
  • List control only creates containers for the visual items
  • Implemented by the VirtualizationStackPanel
  • Default in ListBox, ListView, DataGrid automatically support it
  • ComboBox does not – need to add an ItemsPanelTemplate
  • TreeView supports it, but disabled for backward compat
VIRTUALIZATION CONSIDERATIONS

• Breaking Virtualization
• Using a ScrollView
• Not using an ItemsPresenter in a control template
• Not using DataBinding
• Enable Container Recycling
• Reuses controls and loads with new data
  • VirtualizationStackPanel.VirtualizationMode="Recycling"
CONTROLLING THE VIRTUALIZATION CACHE

- CacheLength
  - How many additional items are created (before, after)
    - `VirtualizingStackPanel.CacheLength="10,50"`
- CacheLengthUnit
  - Page, Item, Pixel
    - `VirtualizingStackPanel.CacheUnit="Page"`
AND FINALLY

• Deferred Scrolling
  • Doesn’t refresh list until user stops scrolling
    • ScrollViewer.IsDeferredScrollingEnabled = "true"
  • Items scroll at the item level
    • Doesn’t clip partial items
    • Enable clipping by setting the ScrollUnit to Pixel
      • VirtualizingStackPanel.ScrollUnit = "Pixel"
Virtualization
THE COMMAND PATTERN
ENCAPSULATING LOGIC

• Wire commands through markup
• Reuse command code (where appropriate)
• Leverage converters
  • IValueConverter
  • IMultiValueConverter
IMPLEMENTING ICommand BY HAND - WPF

```csharp
public class DoSomethingCommand : ICommand
{
    public DoSomethingCommand()
    {
        // do something if necessary
    }
    public void Execute(object parameter)
    {
        _messenger.Show("Clicked!");
    }
    public bool CanExecute(object parameter)
    {
        return (parameter != null) && ((bool) parameter);
    }
    public event EventHandler CanExecuteChanged
    {
        add { CommandManager.RequerySuggested += value; }
        remove { CommandManager.RequerySuggested -= value; }
    }
}
```
The Command Pattern
Questions?
THE MODEL VIEW VIEW-MODEL PATTERN
MODELS

- The data of the application
- (Can be) Optimized for storage
VIEWS

• Accepts Interactions from User
• Returns results of interactions back to user
VIEW MODEL – JOB 1

- Façade for individual models
- Transport mechanism for models
VIEW MODEL – JOB 2

• Process Incoming requests
• Perform changes to the model
WHY MVVM?
DON’T REPEAT YOURSELF
SEPARATION OF CONCERNS

DAMMIT, JIM!
I'M A DOCTOR.
NOT...
A.) A BRICKLAYER
B.) AN ESCALATOR
C.) A MECHANIC
D.) AN ENGINEER
E.) A COAL MINER
F.) ALL OF THE ABOVE
AND THEN SOME
MVVM != 0 CODE BEHIND

• Temper your actions with wisdom
• Code Behind is hard to test
• Complicated markup is hard to understand
IMPLEMENTING MVVM IN WPF
MVVM By Hand
UI INTERACTION
CHALLENGES

• Challenges
  • Must keep separation of concerns
  • Must still be testable
• One suggestion
  • Create Interfaces representing UI
  • Code to UI instead of element
• Another suggestion:
  • Create notification system
  • More complicated, more powerful
CREATE AND LEVERAGE INTERFACES

```csharp
public interface IAnimationController
{
    void StartAnimation();
    void EndAnimation();
}

public override void Execute(object parameter)
{
    _param = parameter as LoginParameter;
    if (_param == null) { return; }
    _param.AnimationController.StartAnimation();
    _bgw.RunWorkerAsync(_param);
}
```
UI Interaction
REVIEW

• MVVM is a pattern
  • Not a specific implementation or code sample
• Goal is to increase
  • Single Responsibility
  • Code Re-use (DRY)
  • Testability
  • SOLID
• Many frameworks available
  • Pick the one that best matches your needs
CUSTOM DICTIONARIES
CUSTOM DICTIONARIES

• Enable by setting SpellCheck.IsEnabled="True"
• Text File (<name>.lex) for custom words
• Add as a resource, assign to the CustomDictionaries collection, copy to output directory
• Four languages supported
  • English, Spanish, German, French
Custom Dictionaries
ASYNCHRONOUS EVERYWHERE
**ASYNC - AWAIT**

`await` is the easiest way to run asynchronously and *in order*

```csharp
private async void DownloadPage()
{
    HttpClient client = new HttpClient();
    string bing = await client.GetStringAsync("http://www.bing.com");
    string ms = await client.GetStringAsync("http://www.microsoft.com");
}
```
Tasks provide more control, like running in parallel

```csharp
private async void DownloadPage()
{
    HttpClient client = new HttpClient();
    var tasks = new Task<string>[
    {
        client.GetStringAsync("http://www.bing.com"),
        client.GetStringAsync("http://www.microsoft.com"),
    };
    int first = await Task.WaitAny(tasks);
    string firstPage = tasks[first].Result;
}
```
WINDOWS 8.1 IS WINDOWS VNEXT

Units Since Launch of Windows 7 as of March 2012, IDC
USER INTERFACES REIMAGINED
TRAITS OF MODERN APPLICATION DESIGN

1. Be Fast and fluid
2. Size beautifully
3. Use the right contracts
4. Invest in a great Tile
5. Feel connected & alive
6. Roam to the cloud
7. Embrace Modern Design principles
MODERN DESIGN PRINCIPLES

• Show pride in Craftsmanship
• Be authentically digital
• Do more with less
• Win as One
Skeuomorphism *(noun)* - an object or feature which imitates the design of a similar artefact in another material.

Windows 8.1 Goal – anti-skeuomorphism

Common Chrome

- Layout
- Interactions
- Navigation
SCALE BEAUTIFULLY

- Orientation
  - Landscape
  - Portrait
- Edges
  - Left
  - Right
  - Both
- Scaling
  - Windows 8.1 supports many form factors – so should your app
WINDOWS 8.1 – (C# || VB.NET)/XAML
WINDOWS 8.1 CONTROLS
MANY CONTROLS OUT OF THE BOX

• Designed for
  • Touch
  • Mouse
  • Keyboard
• Follow Modern Application Design Principles
• Native to XAML/HTML
DATA VIEWS

List View

Grid View

Flip View
DEMO

Windows 8.1 Application
WHY ADD XAML BASIC PAGE?

• Updates the layout of your application
• Adds framework classes (including)
  • Navigation
  • RelayCommands
  • SuspensionManager
  • Observable Dictionary
• For more information see ReadMe.txt (also added to the project)
WHY USE XAML RESOURCES?

• Resources are
• Reusable
• Easier to maintain
• Easier to localize
APPLICATION ARCHITECTURE
ARCHITECTURE – (C# || VB.NET)/XAML

App.xaml + App.cs

Frame

pagex.xaml + pagex.cs
(injected into Frame as navigation occurs)
NAVBARS, APPBARS, COMMANDBARS
APPBARS AND COMMANDBARS (CS)
APPBARS SIMPLIFIED IN WINDOWS 8.1 (CS)

```xml
<Page.TopAppBar>
  <CommandBar>
    <CommandBar.SecondaryCommands>
      <AppBarButton Label="AppBarButton" Icon="Filter"/>
    </CommandBar.SecondaryCommands>
    <AppBarButton x:Name="SearchNav" Label="Search" Icon="Zoom"
      Click="SearchNav_Click"/>
    <AppBarButton x:Name="ShareSourceNav" Label="Share (Source)"
      Icon="Forward" Click="ShareSourceNav_Click"/>
  </CommandBar>
</Page.TopAppBar>
```
private void ppBarButton_Click(object sender, RoutedEventArgs e)
{
    this.Frame.Navigate(typeof(PhotoPage)[, myArgs]);
}

protected override void OnNavigatedTo(NavigationEventArgs e)
{
    if ((SearchActivatedEventArgs)e.Parameter != null)
    {
        //Do something
    }
}
USE VISUAL STATE GROUPS (CS)

- Create a Visual State for each layout (XAML)
  - Add Storyboard for each control
- Handle the size changed event (Code)
  - Use portrait Visual State Group
    - Height > Width
    - ApplicationViewOrientation == ApplicationViewOrientation.Portrait
  - Use VisualStateManager.GoToState
<VisualStateManager.VisualStateGroups>
  <VisualStateGroup>
    <VisualState x:Name="DefaultLayout">
      <Storyboard>
        </Storyboard>
    </VisualState>
    <VisualState x:Name="Portrait">
      <Storyboard>
        <ObjectAnimationUsingKeyFrames Storyboard.TargetName="imagePanel">
          <DiscreteObjectKeyFrame KeyTime="0">
            <Orientation>Vertical</Orientation>
            <DiscreteObjectKeyFrame.Value>
            </DiscreteObjectKeyFrame.Value>
          </DiscreteObjectKeyFrame>
        </ObjectAnimationUsingKeyFrames>
      </Storyboard>
    </VisualState>
  </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Page
    x:Name="pageRoot"
    x:Class="Lab3.PhotoPage"
    RequestedTheme="Light"
    SizeChanged="pageRoot_SizeChanged"/>

private void pageRoot_SizeChanged(object sender, SizeChangedEventArgs e)
{
    //if (ApplicationView.GetForCurrentView().Orientation ==
    //   ApplicationViewOrientation.Poratrait)
    if (e.NewSize.Height / e.NewSize.Width >= 1)
    {
        VisualStateManager.GoToState(this, "Portrait", true);
    }
    else
    {
        VisualStateManager.GoToState(this, "DefaultLayout", true);
    }
}
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    navigationHelper.OnNavigatedTo(e);
    this.SizeChanged += PhotoPage_SizeChanged;
}
protected override void OnNavigatedFrom(NavigationEventArgs e)
{
    navigationHelper.OnNavigatedFrom(e);
    this.SizeChanged -= PhotoPage_SizeChanged;
}
void PhotoPage_SizeChanged(object sender, SizeChangedEventArgs e)
{
    if (ApplicationView.GetForCurrentView().Orientation == ApplicationViewOrientation.Portrait)
        if (e.NewSize.Height / e.NewSize.Width >= 1)
        {
            VisualStateManager.GoToState(this, "Portrait", true);
        }
    else
    {
        VisualStateManager.GoToState(this, "DefaultLayout", true);
    }
}
TESTING LAYOUT IN SIMULATOR
DEMO

Navigation, Layout, and Views
KEY POINTS - XAML

• Navigation commands belong in NavBar
• Navigate through Frame object
• BackButton is handled through Basic Page
• XAML Layout containers flow
  • Grids and StackPanels
• Use Document Outline to wrap controls in a layout
KEY POINTS (CONTINUED)

• Create Applications that Snap and Scale
• Device Panel
  • Change layout and dimensions
  • Record changes based on layout
WINDOWS PHONE 8.1
GETTING STARTED
Create universal Windows apps across phones, tablets and PCs

Free tools for a fast start

Samples to get you coding

Windows Dev Center: HTTP://DEV.WINDOWS.COM
GETTING THE TOOLS

• The Windows Developer Center is your base for all things Windows and Windows Phone related
  • http://dev.windows.com    http://dev.windowsphone.com

• FREE download: Visual Studio Express 2013 for Windows
  • Enables development for Windows 8.1 and Windows Phone 8.1

• In Visual Studio 2013 Professional or higher, install Update 2 to add in Windows Phone 8.1 dev tools
DEVELOPMENT PC REQUIREMENTS

• Your computer must meet the following system requirements to run Windows Phone SDK 8.1:

<table>
<thead>
<tr>
<th>Supported operating systems</th>
<th>Windows 8.1 (x86 or x64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>4 GB of free disk space</td>
</tr>
<tr>
<td></td>
<td>1.5 GB of RAM (recommended)</td>
</tr>
<tr>
<td></td>
<td>64-bit (x64) motherboard</td>
</tr>
<tr>
<td>To use the Windows Phone Emulators</td>
<td>Windows 8.1 Pro or higher (for Hyper-V) and Second Level Address Translation (SLAT) Min 4GB RAM</td>
</tr>
</tbody>
</table>
GETTING A WINDOWS DEVELOPER ACCOUNT

• You do not need a Windows Developer account to download the SDK and start developing apps
• You need a developer account to publish Windows Phone and/or Windows Store apps, to use Push Notifications and to unlock more than one phone for development

• To get a Developer Account:
  • Included if you have an MSDN subscription
  • Free to students who have a Dreamspark
  • $19 charge per annum for individual developers, $99 for Company accounts

• New!
  • No Credit card required to sign up
  • Paypal supported
NATIVE APP DEVELOPMENT ON WP8.1

Gaming

DirectX/
Direct3D
C++

HTML

Windows Phone Store Apps with HTML
JavaScript
WinJS

XAML

Windows Runtime
XAML
C#/VB
or C++

Windows Phone
Silverlight XAML
C#/VB

Windows Runtime

* Apps written for Windows Phone 7.x/8.0 all run on Windows Phone 8.1

Delivering the Art of Software
<table>
<thead>
<tr>
<th>Approach</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keep using Silverlight 7.x/8 (Windows Phone 8 platform)</td>
<td>Will run on Windows Phone 8.1 (app compatibility), still runs on earlier Windows Phone devices</td>
<td>Will not take advantage of new platform capabilities</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Will not enable targeting of Windows</td>
</tr>
<tr>
<td>Use Silverlight 8.1 (Move Silverlight app into new execution stack and app package)</td>
<td>Enables existing Silverlight apps to take advantage of nearly all new APIs and platform capabilities</td>
<td>Will not enable targeting of Windows</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Will not run on Windows Phone 8 devices</td>
</tr>
<tr>
<td>Use Windows XAML platform</td>
<td>Enables targeting of Windows and Windows Phone</td>
<td>Will not run on Windows Phone 8 devices</td>
</tr>
<tr>
<td></td>
<td>Takes full advantage of new APIs and platform capabilities</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Better performance &amp; reduced memory use</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ONLY AVAILABLE ON SILVERLIGHT

- CameraCaptureTask
- Camera Lenses
- Lockscreen background image provider
- Runs under Lock
- Background Audio Agent
- Alarms/Reminders

- SocialRT (Silverlight 8.1 only)
- VoIP
- Continuous background location tracking (SL 8.0 only)
- Wallet agents
- System.ServiceModel (WCF/SOAP)
WHAT’S NEW IN WP8.1
WHAT’S NEW IN WINDOWS PHONE 8.1

- Action center to quick launch settings and view notifications
- New Skype app integrated with Phone dialer
- New Calendar app with new Week view
- Fast typing with new Word Flow keyboard
- Buy app once: use on PC and on Phone
- App data can roam among devices
- Start screen backup and restore
- App data backup
- Faster app updates
- Remote viewing of Phone screen
- Install apps to SD card
GETTING WINDOWS PHONE OS 8.1

• New devices sold into market from June 2014 will have Windows Phone 8.1 pre-installed
• All devices running Windows Phone 8.0 are capable of being upgraded to Windows Phone 8.1
  • Availability of upgrades for contract devices is under the control of the Mobile Operator
• Registered developers can upgrade their phone through the ‘Preview for Developers’ program
  • Developers who have signed up for a developer account at http://dev.windowsphone.com
  • Anyone who has signed up (free!) for Microsoft’s App Studio tool: http://appstudio.windowsphone.com
WP8.1 PREVIEW FOR DEVELOPERS

• Preview for Developers
  • Installs OS updates on your device
  • Does not install OEM firmware updates
  • When the ‘proper’ update is released by your MO/OEM, you will get that update as normal
WINDOWS AND WINDOWS PHONE CONVERGENCE
BRINGING ONE WINDOWS TO DEVs

- Designed once; engaging everywhere
  - Converged developer platform
  - Converged app model
  - Shared app identities + entitlement
  - Unified push services (via WNS)
- Apps come to life on Windows
  - Live tile improvements
  - Action Center for smart notifications
  - Background execution + triggers
  - Bluetooth-LE support for wearables + beacons
  - Internet Explorer 11 improvements
PLATFOM CONVERGENCE IS A JOURNEY…

Windows Phone 7.5
Convergence Begins with IE
- WP 7.5 shipped with IE9
- Same IE codebase as Windows
- Same JavaScript engine as Windows

Converged Core OS
- Common NT kernel, file system and core networking
- Kernel mode driver f/work
- Secure boot & BitLocker

Developer Platform
- Partial API convergence (focus on sensors & IAP)
- Native Code (C++) and DirectX
- IE10

Windows Phone 8.0

Converged Dev Platform
- More skillset reuse
- More code reuse
- More seamless app experiences

Aligning the Stores
- Shared dev registration
- Shared entitlement

Common Core Platform
- Proximity & Location frameworks
- Security & identity
- Task scheduler

Windows Phone 8.1
API CONVERGENCE BETWEEN WINDOWS

• The Windows Runtime (WinRT) is the shared runtime and API space used by store apps across the Windows platform (phone and client)

• Dramatic convergence in 8.1
  • Goal is 100% convergence for dev scenarios
  • In 8.0, we had ~30% API convergence
  • With 8.1, we move well past 90%+ convergence
**WINDOWS PHONE 8.1 PLATFORM PRINCIPLES**

<table>
<thead>
<tr>
<th>Primary features:</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Existing Windows Phone 8 apps run flawlessly</td>
</tr>
<tr>
<td>✓ Windows Runtime XAML platform and HTML/WinJS support brings convergence with Windows 8.1, and is faster and uses less memory than Silverlight</td>
</tr>
<tr>
<td>✓ The core of an app can be built once for Windows and Windows Phone; you can share elements of UI, though it’s best to tailor the user experience for each platform</td>
</tr>
<tr>
<td>✓ Convergence is not just in the developer platform: it spans tools, store, commerce</td>
</tr>
<tr>
<td>✓ Developer investment in the existing Silverlight platform is protected: Existing Silverlight apps can be upgraded to access new Windows Phone 8.1 platform capabilities</td>
</tr>
</tbody>
</table>

**Now is the time to build for Windows and Windows Phone!**
WHAT’S NEW FOR DEVELOPERS
WHAT’S NEW FOR DEVELOPERS

• Developers can create a single app that targets Windows Phone and Windows
• One Visual Studio ‘Universal app’ project template for your app
• Shared dev and test tooling for both platforms
• Developers can use their HTML/CSS and JavaScript skills to create native apps that run on Windows Phone, similar to Windows
COMMON CANVAS FOR SCREEN DIVERSITY

• Shared virtual layout canvas
• Provides a common layout coordinate system that extends along the x and y axis into infinity, allowing you to scale up
CALCULATING SCALE FACTOR

• Scale Factor
  • Pixels / Width X Platform Constant
  • Minimum of 1.0, rounded to nearest 0.1
  • Platform Constant depends on viewing distance of device
    • Phone/Tablet/PC

• Applying Scale Factor
  • Raw Pixels / Scale Factor = View Pixels
STEPS TO HANDLING DIFFERENT SCREENS

• Provide bitmap images to work with scaling system
  • Provide image.scale-xxx.png
  • If no exact match, system falls back to next best factor
• Build responsive UI that adapts to usable screen size and Orientation
  • Subscribe to OrientationChanged event (not SizeChanged)
• Make use of DisplayInformation for granular display
CONVERGED CONTROLS

What’s it really mean?

80% exact same XAML

20% custom

Common
- Button
- CheckBox
- RadioButton
- ToggleSwitch
- Slider
- ProgressBar

Optimized
- DatePicker
- TimePicker
- CommandBar
- AppBar

Signature
- Hub
- Pivot
- ListView
- GridView
CONTROLS FOR TAILORING UI TO EACH PLATFORM

On Windows Phone 8.1

… and on Windows 8.1
ANIMATIONS

• Signature Phone animations built in
  • Page Navigation
  • PointerDown/Up (Tilt)
  • Page Rotation
  • Escalator for MenuFlyout (context menu)
  • Readerboard
• ThemeTransitions / ThemeAnimations
  • Same API as on Windows
  • Key animations updated to match Phone UX
TEXT HANDLING CONTROLS

- **TextBlock**
  - Displays text
  - Supports line breaks and word wrapping

- **RichTextBlock**
  - Paragraphs, spans, and runs allow for formatting sections of the rich text

- **TextBox**
  - Fully templatable Header property
  - Supports spell-checking, placeholder text, text-prediction and input scope for on-screen keyboards, multi-line support

- **PasswordBox**
  - Obscures text entry
  - Supports placeholder text and templateable Header
INPUT SCOPE

• Sets layout of on-screen keyboard
  • Many different types available – see documentation for full list

• Does not provide any data validation

• Implement to provide the most efficient experience to your users.
INPUT SCOPE IN ACTION

<TextBox InputScope= "EmailSmtpAddress"/>

<TextBox InputScope= "CurrencyAmountAndSymbol"/>

<TextBox InputScope= "Number"/>
BUTTON, HYPERLINK BUTTON

• Standard XAML Button
• Customizable through
  • Properties
  • Styles
• Custom Content
• Support Events and Commands
OTHER TYPES OF BUTTON CONTROLS

• **ToggleButton**
  • Behaves like a CheckBox, looks like a button.

• **CheckBox and RadioButton**
  • Inherited from ToggleButton to provide the expected functionality
  • Supports tri-state through null or, in XAML, `{x:Null}`

• **AppBarButton, AppBarToggleButton**
  • Provides buttons for use in standard app bars, and also on the main canvas. We’ll cover this when we discuss app bars.
PROGRESSRING AND PROGRESSBAR

• Progress Ring
  • Set IsActive=True to spin

• Progress Bar
  • Value based
  • Indeterminate

• Important!
  • Deactivate when not visible
  • Otherwise, Performance suffers
DATE/TIME PICKERS

• Same API on WP8.1 and Win8.1
• UI Adjusts for platform

• Allows restricting segments
• E.g. Show only month and year
FLYOUTS

• Converged with Windows 8.1
• MenuFlyout used to create context menu
• New Phone-only flyouts
  • List Picker Flyouts
  • Date/TimePicker Flyouts
  • Generic Picker Flyouts
• All are “light-dismiss”:
  • Dismiss by Back button, or for non full-screen flyouts such as MenuFlyout, dismiss by tapping outside of the control
CONTENT DIALOG

• A custom message box to place arbitrary content

• Supports both full and partial screen
AUTOSUGGESTBOX

• App code provides suggestions

• Automatic positioning to maximize space for suggestions

• Fully re-templatable
APPBAR

• Top AppBar not supported
• Bottom AppBar
  • Created the same as Win8.1
  • AppBar not shown by gesture
  • Can display minimized
  • AppBarSeparator not shown
  • IsSticky ignored on WP8.1
• Opened/Closed events ignored on WP8.1
APP BAR BUTTONS MENU FLYOUTS

• Content is MenuFlyout
• Contains MenuFlyoutItems
STATUS BAR

• Status bar is visible by default
• Background color is same as the containing page
  • Can program BackGroundColor and BackgroundOpacity
  • Can’t control through XAML
• Can show a progress indicator
APPLICATION CONTENT AREA

• Content resizes based on visibility of AppBar and StatusBar
  • Can override this behavior and disable automatic window resizing
  • If AppBar and/or StatusBar are transparent they overlay the content
LAYOUT AND NAVIGATION
WINNDS, FRAMES AND PAGES

The window contains a single frame, sized at 100% of area. The frame contains pages, also typically sized at 100% of the area available to the window. In Windows Store Apps for Tablet/PC, apps may have more than one window. On phone, apps have only one window.
• Standard Windows Phone UX is to use Back key to navigate back or close transient UI
• By default, Back key causes a navigation back to the previous App – not Page!
• In Windows Phone Store apps, you must include code to override this to cause a backwards navigation within the app
• Differs from Windows Phone Silverlight – within an app, backwards page navigation is default for that framework

• With correct Back key handling, pressing the Back key on the launch page suspends the current app and navigates back to previous app
• Differs from Windows Phone Silverlight – Closes the current app in that framework
BACK KEY HANDLING

- New Project templates:
  - Blank App does not include Back key handling
  - Hub App, Pivot App does
- Included in /Common/NavigationHelper class
- Causes a backwards Page navigation
- If you need to override this, replace with your own code for custom navigation handling
public sealed partial class SecondPage : Page
{
    protected override void OnNavigatedTo(NavigationEventArgs e)
    {
        Windows.Phone.UI.Input.HardwareButtonsBackPressed += HardwareButtonsBackPressed;
    }
    protected override void OnNavigatedFrom(NavigationEventArgs e)
    {
        Windows.Phone.UI.Input.HardwareButtonsBackPressed -= HardwareButtonsBackPressed;
    }
    async void HardwareButtonsBackPressed(object sender, Windows.Phone.UI.InputBackPressedEventArgs e)
    {
        e.Handled = true; // We've handled this button press
        // If the secondary UI is visible, hide it
        if (SecondaryUI.Visibility == Visibility.Visible)
        {
            FadeOutSecondaryStoryboard.Begin();
        }
        else
        {
            // Standard page backward navigation
            if (Frame.CanGoBack) Frame.GoBack();
        }
    }
}
PAGE CACHE MODE

• When you navigate to a Page type the first time, a new instance is created.
  • Whether that instance is cached or destroyed when you navigate away is determined by
    the NavigationCacheMode property of the Page

• NavigationCacheMode.Disabled
  • You get a new instance of the page whether you navigate forward or backward to the page

• NavigationCacheMode.Enabled
  • The page is cached, but the cached instance is discarded when the size of the cache for
    the frame is exceeded (determined by Frame.CacheSize property – default on Windows
    Phone is 10).

• NavigationCacheMode.Required
  • The page is cached and the cached instance is reused for every visit regardless of the
    cache size for the frame
RESOURCES

• Pro WPF 4.5 in C#
  • Mathew McDonald
  • Apress
  • http://bit.ly/1wk4AaM
• Microsoft Virtual Academy
  • http://www.microsoftvirtualacademy.com/
Questions?
Contact Me
philj@interknowlogy.com
www.interknowlogy.com
skimedic@gmail.com
www.skimedic.com/blog
www.twitter.com/skimedic
www.hallwayconversations.com
www.about.me/skimedic

Thank You!