Windows Programming

Windows Presentation Foundation

Karol Walędzik, MSc
k.waledzik@mini.pw.edu.pl
http://www.mini.pw.edu.pl/~kwaledzik

Introduction

WPF History

- **2001**
  - New team formed by Microsoft to build a unified presentation platform that could eventually replace User32/GDI32, Visual Basic, DHTML, and Windows Forms
- **2003**
  - Avalon project announced at Professional Developer Conference
- **2006**
  - WPF released as a part of the .NET Framework 3.0
  - VS 2005 Extensions for .NET 3.0 (CTP)

WPF
Aims & Principles

- Platform for rich presentation
  - programmable
  - declarative
- Integrate UI, documents, and media
- Incorporate the best of the Web, and the best of Windows
- Integrate developers and designers

WPF
Features & Selling Points

- Interoperability
  - Win32 interoperability
  - Windows Forms – WPF interoperability:
    - ElementHost and WindowsFormsHost classes
- 2D Graphics improvements
  - DirectX based, retained rendering system with advanced graphical features
  - Graphics Processing Unit utilization
  - vector-based graphics with lossless scaling
  - built-in set of brushes, pens, geometries, and transforms
  - support for most common image formats

WPF
Features & Selling Points cont'd

- 3D Graphics and multimedia:
  - 3D capabilities as a subset of the full feature Direct3D's set
    - incl. 3D model rendering
  - support for WMV, MPEG and some AVI films
  - support for Windows Media Player codecs
- Animations:
  - time-based animations
  - triggered by other external events, including user action
  - animation effects applied on a per-object (or even per-property) basis
  - set of predefined animation effects

WPF

FEATURES & SELLING POINTS CONT'D

- Data binding:
  - flexible declarative definition
    - in-code definition available as well
  - support for two-way changes notifications
- Styling and templating:
  - ability to declaratively (or programmatically) change the visuals of each and every control
- Accessibility:
  - Microsoft UI automation

WPF

FEATURES AND SELLING POINTS CONT'D

	Windows Forms	PDF	Windows Forms/ GDI+	Windows Media Player	Direct3D	WPF
Graphical Interface, e.g., Forms and Controls	X					X
On-Screen Documents	X					X
Fixed-Format Documents		X				X
Images			X			X
Video and Audio				X		X
Two-Dimensional Graphics			X			X
Three-Dimensional Graphics					X	X

http://windowsclient.net/learn/techarticle.aspx?a=WPF
Introducing WPF 3.5

XAML

Extensible Application Markup Language (XAML)

- Markup language for declarative application programming
- Not specific to WPF (or even .NET)
- Describes behavior and integration of components without the use of procedural programming
- In WPF:
  - intended to facilitate a separation of model and view
  - direct equivalent of analogous code in C#
  - ultimately compiled into a managed assembly in the same way all .NET languages are

XAML Namespaces

- XML namespaces declared in **xmlns** attributes
  - can be placed inside any element start tag
    - usually declared in the very first tag
- Two basic namespaces:
  - http://schemas.microsoft.com/winfx/2006/xaml
    - XAML namespace including various XAML utility features
    - usually mapped to prefix x
  - http://schemas.microsoft.com/winfx/2006/xaml/presentation
    - core WPF namespace
    - encompasses all the WPF classes, incl. all controls
    - usually declared as the default namespace for the entire document

XAML

- Object instances
  - created by including XML nodes in XAML
- Property values set:
  - via attributes
    - only for primitives and simple (string-convertible) types
  - via nested elements
    - allows declaring complex types

```
<TextBox Background="Red" Text="Text"/>


<TextBox Text="Text">
    <TextBox.Background>
        <SolidColorBrush Color="Red“/>
    </TextBox.Background>
</TextBox>


<TextBox Text="Text">
    <TextBox.Background>
        <SolidColorBrush>
            <SolidColorBrush.Color>
                <Color R="255" A="255"/>
            </SolidColorBrush.Color>
        </SolidColorBrush>
    </TextBox.Background>
</TextBox>
```

XAML EXAMPLES

```
<Application x:Class="WpfApp.App"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   StartupUri="Window1.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>
```

```
<Window x:Class="WpfApp.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentati
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Window1" Height="300" Width="300">
    <Grid>
    </Grid>
</Window>
```

WPF APPLICATIONS

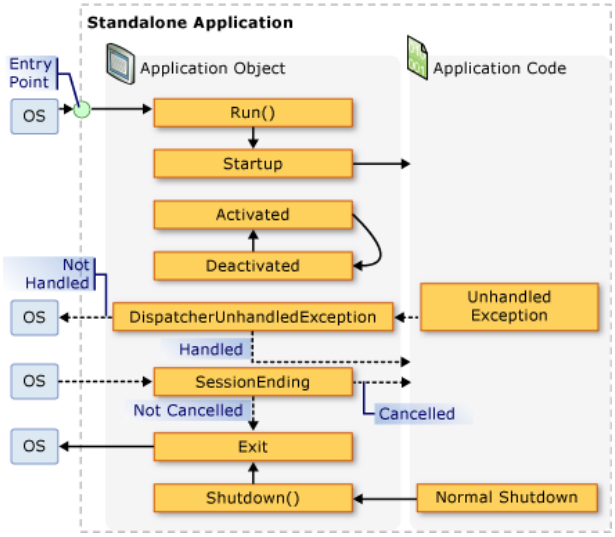
WPF APPLICATION TYPES

- Standalone applications:
  - create typical **Window**s
  - may make use of all the common dialog boxes
- Browser-hosted applications (XBAPs)
  - hosted in browsers (IE6+ or Firefox)
  - consisting of **Page**s and **PageFunction**s
    - navigatable via **Hyperlink**s and/or **NavigationService**
  - not to be confused with Silverlight applications
    - (entirely different technology)
- Hybrid:
  - standalone applications making use of **NavigationWindow**s and/or **Frame**s to host **Page**s

APPLICATION CLASS

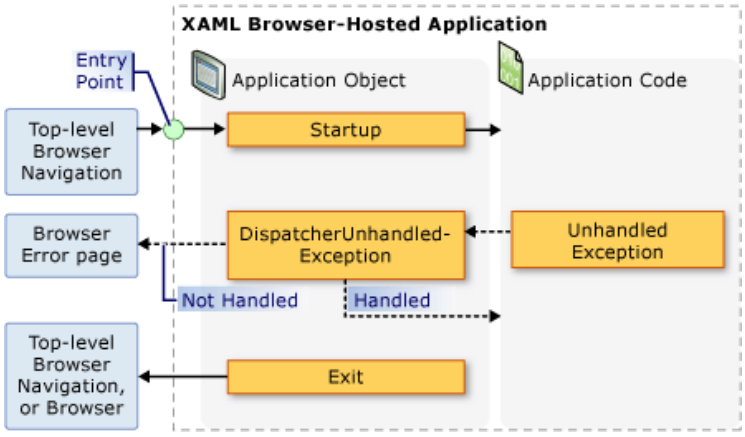
- encapsulates WPF application-specific functionality, incl.:
  - application lifetime
  - application-scope window, property, and resource management
  - command-line parameter and exit code processing
  - navigation
- can be implemented using markup, code or mix of both
- singleton
  - only one instance can be created per **AppDomain**
  - **Application.Current**
- not required for simple standalone applications

APPLICATION LIFETIME



http://msdn.microsoft.com/en-us/library/ms743714.aspx#Application_Lifetime

APPLICATION LIFETIME



http://msdn.microsoft.com/en-us/library/ms743714.aspx#Application_Lifetime

APPLICATION MARKUP

```
<Application
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentati
on”
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml”
    x:Class="SDKSample.App”
    StartupUri="MainWindow.xaml”
    Startup="App_Startup" Exit="App_Exit”
    ShutdownMode="OnMainWindowClose“
    DispatcherUnhandledException=
           "App_DispatcherUnhandledException">

...

</Application>
```


APPLICATION CODE-BEHIND

```
using System.Windows; using System.Windows.Threading;

namespace SDKSample
{
   public partial class App : Application
   {

      ...

      void App_DispatcherUnhandledException(
           object sender, DispatcherUnhandledExceptionEventArgs e)
      {
          // Process unhandled exception
          ...
          // Prevent default unhandled exception processing
          e.Handled = true;
      }

   }
}
```

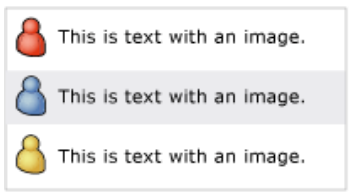
CONTROLS

CONTENT MODELS

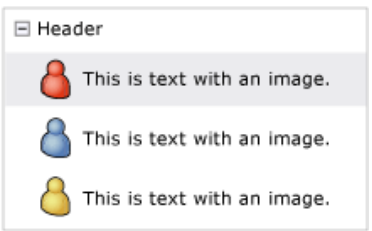
A Button, which is a ContentControl.

A GroupBox, which is a HeaderedContentControl.

A ListBox, which is an ItemsControl.



A TreeViewItem, which is a HeaderedContentControl.



http://msdn.microsoft.com/en-us/library/bb613548.aspx

Single Child Content Model

- **ContentPropertyAttribute**

```
<Button>This is string content of a Button</Button>

<Button Content="More string content"/>

<Button xmlns:sys="clr-namespace:System;assembly=mscorlib">
   <sys:DateTime>2004/3/4 13:6:55</sys:DateTime>
</Button>

<Button>
   <Rectangle Height="40" Width="40" Fill="Blue"/>
</Button>

<Button>
   <StackPanel>
        <Ellipse Height="40" Width="40" Fill="Blue"/>
        <TextBlock TextAlignment="Center">Button</TextBlock>
   </StackPanel>
</Button>
```


Single Child Content Model cont'd

- **ContentControl**
  - **ContentPresenter** – responsible for displaying control's content; its algorithm:
    - Content is **UIElement** → add it to the display tree
    - **ContentTemplate** is set → use it to create a **UIElement**
    - **ContentTemplateSelector** is set → use it to find a template
    - a data template is associated with the content data type → use it
    - **TypeConverter** is associated with the content data type and can convert it to **UIElement** → use it
    - **TypeConverter** is associated with the content data type and can convert it to string → use it to create a **TextBlock** with the string value
    - use **ToString()** method on the content object

Common Controls

- **UserControl**
- **TextBlock**
  - block-level text
- **Run**
  - inline-level text
- **ButtonBase**
  - **Button**
    - **ToggleButton**
      - **CheckBox**
      - **RadioButton**

Lists

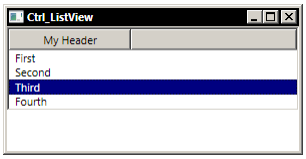
- ListBox and ComboBox
  - **ItemsSource**
    - any type implementing **IEnumerable** acceptable
    - **ObservableCollection<T>** provides change notifications
      - it implements **INotifyCollectionChaned**
  - ItemsPanel
    - of type: **ItemsPanelTemplate**
    - template for the layout panel that will be used to display the items in the list

LISTS

```
<StackPanel>
    <ComboBox>
        <ComboBoxItem>First</ComboBoxItem>
        <ComboBoxItem>Second</ComboBoxItem>
        <ComboBoxItem>Third</ComboBoxItem>
    </ComboBox>
    <Rectangle Height="100"></Rectangle>
    <ListBox>
        <ListBox.ItemsPanel>
            <ItemsPanelTemplate>
                <UniformGrid Columns='2'/>
            </ItemsPanelTemplate>
        </ListBox.ItemsPanel>
        <ListBoxItem>First</ListBoxItem>
        <ListBoxItem>Second</ListBoxItem>
        <ListBoxItem>Third</ListBoxItem>
        <ListBoxItem>Fourth</ListBoxItem>
    </ListBox>
</StackPanel>
```

LISTVIEW

- **ListView** control
  - derives from **ListBox**
  - supports separation of view and control properties
  - View
    - possible to create custom view
    - predefined: **GridView**




```
<StackPanel>
    <ListView>
        <ListViewItem>First</ListViewItem>
        <ListViewItem>Second</ListViewItem>
        <ListViewItem>Third</ListViewItem>
        <ListViewItem>Fourth</ListViewItem>
        <ListView.View>
            <GridView>
                <GridViewColumn
          Header="My Header"
          Width="200"/>
            </GridView>
        </ListView.View>
    </ListView>
</StackPanel>
```

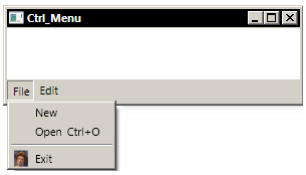
TreeView

- **TreeView** control
  - displays a hierarchical structure of collapsible nodes
    - hierarchy of **TreeViewItem** controls
    - **TreeViewItem** is a **HeaderedItemsControl**


```
<TreeView>
    <TreeViewItem Header="1"
        IsExpanded="True">
        <TreeViewItem>
            <TreeViewItem.Header>
                <DockPanel>
                    <CheckBox/>
                    <TextBlock>11</TextBlock>
                </DockPanel>
            </TreeViewItem.Header>
        </TreeViewItem>
        <TreeViewItem Header="12“>
            <TreeViewItem Header="121"/>
            <TreeViewItem Header="122"/>
        </TreeViewItem>
    </TreeViewItem>
</TreeView>
```


Menu

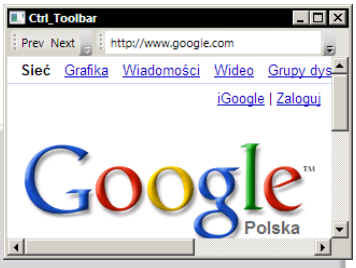
- Logically nothing more than a **TreeView** with special template
  - both **TreeViewItem** and **MenuItem** derive from **HeaderedItemsControl**
- Typically used with commands




```
<Menu DockPanel.Dock="Bottom">
    <MenuItem Header='_File'>
        <MenuItem Header='_New'/>
        <MenuItem Header='_Open‘
                InputGestureText="Ctrl+O" />
        <Separator/>
        <MenuItem Header='E_xit‘
                Click="ExitMenuItem_Click">
            <MenuItem.Icon>
                <Image
                 Source="Images/Fiona_67x77.gif”
                 Width="16" Height="16">
                </Image>
            </MenuItem.Icon>
        </MenuItem>
    </MenuItem>
    <MenuItem Header='_Edit'>
        <MenuItem Header='_Cut' />
        <MenuItem Header='C_opy' />
        <MenuItem Header='_Paste' />
    </MenuItem>
</Menu>
```

TOOLBAR

- Supports only one level of nesting
  - however: anything can be an item
- Special container: **ToolBarTray**




```
<DockPanel>
    <ToolBarTray DockPanel.Dock='Top'>
        <ToolBar>
            <Button>Prev</Button>
            <Button>Next</Button>
        </ToolBar>
        <ToolBar>
            <TextBox Name="AddressTextBox" Width='200'
                     Text="http://www.google.com" />
            <Button Width='23' Click="GoButton_Click">Go</Button>
        </ToolBar>
    </ToolBarTray>
    <WebBrowser Name="ContentWebBrowser"></WebBrowser>
</DockPanel>
```


CONTAINERS

- **TabControl** – provides traditional tab-style UI
  - derives from **Selector** (and therefore **ItemsControl**)
- **Expander** – offers Windows XP-style functionality known from the file explorer
- **GroupBox** – provides a simple visual containment for separating parts of the UI
- **TabItem**, **Expander**, and **GroupBox** derive from **HeaderedContentControl**

CONTAINERS CONT'D

```
<TabControl>
    <TabItem Header='Tab 1'>
        <StackPanel>
            <Expander Header='Expander 1' IsExpanded='True'>
                <GroupBox Header='GroupBox 1'>
                    <StackPanel Orientation="Horizontal">
                        <Label>Something 1</Label>
                        <Image Source="Images/Fiona_67x77.gif"></Image>
                    </StackPanel>
                </GroupBox>
            </Expander>
            <Expander Header='Expander 2'>
                <StackPanel>
                    <GroupBox Header='GroupBox 2'>
                        <Label>Something 2</Label>
                    </GroupBox>
                    <GroupBox Header="GroupBox 3">
                        <Label>Something 3</Label>
                    </GroupBox>
                </StackPanel>
            </Expander>
        </StackPanel>
    </TabItem>
    <TabItem Header='Tab 2' />
</TabControl>
```

RANGE CONTROLS

- **Slider**
- **ProgressBar**
- **ScrollBar**

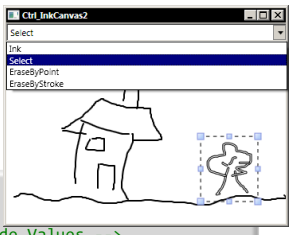
```
<UniformGrid Columns="2">
    <Label Name="SliderLabel">0</Label>
    <Slider Name="MySlider"
    ValueChanged="MySlider_ValueChanged"
    TickPlacement="TopLeft" />
    <Label Name="ScrollBarLabel">0</Label>
    <ScrollBar Name="MyScrollBar"
        ValueChanged="MyScrollBar_ValueChanged"
        Orientation="Horizontal" />
    <Button Name="ProgressButton"
    Click="ProgressButton_Click">
        Move progress
    </Button>
    <ProgressBar Name="MyProgressBar"/>
</UniformGrid>
```

```
private void MySlider_ValueChanged(object sender,
                                   RoutedPropertyChangedEventArgs<double> e)
{ SliderLabel.Content = MySlider.Value; }

private void MyScrollBar_ValueChanged(object sender,
                                      RoutedPropertyChangedEventArgs<double> e)
{ ScrollBarLabel.Content = MyScrollBar.Value; }

private void ProgressButton_Click(object sender, RoutedEventArgs e)
{ MyProgressBar.Value += MyProgressBar.SmallChange * 10; }
```

EDITORS

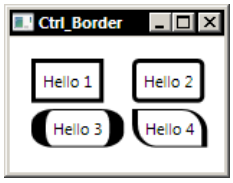
- **PasswordBox**
  - secure password storage
    - **SecurePassword** retrieves password as **SecureString**
- **TextBox**
- **RichTextBox**
  - hosts a **FlowDocument** with editable content
- **InkCanvas**
  - receives and displays ink input

INKCANVAS




```
<Canvas>
    <Canvas.Resources>
        <!--Define an array containing the InkEditingMode Values.-->
        <x:Array x:Key="MyEditingModes„
                 x:Type="{x:Type InkCanvasEditingMode}">
            <x:Static Member="InkCanvasEditingMode.Ink"/>
            <x:Static Member="InkCanvasEditingMode.Select"/>
            <x:Static Member="InkCanvasEditingMode.EraseByPoint"/>
            <x:Static Member="InkCanvasEditingMode.EraseByStroke"/>
        </x:Array>
    </Canvas.Resources>
    <StackPanel>
        <ComboBox Name="cmbEditingMode”
                  ItemsSource="{StaticResource
                       MyEditingModes}" />
        <InkCanvas
                  EditingMode="{Binding
                     ElementName=cmbEditingMode,
                     Path=SelectedItem}">
        </InkCanvas>
    </StackPanel>
</Canvas>
```

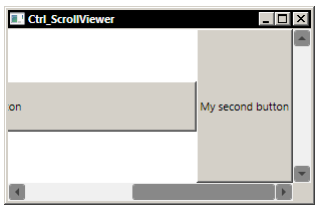
BORDER




```
<Canvas>
    <Border Canvas.Left='15' Canvas.Top='15' BorderThickness='3‘
            CornerRadius='0' BorderBrush='Black' Padding='5'>
        <TextBlock>Hello 1</TextBlock>
    </Border>
    <Border Canvas.Left='85' Canvas.Top='15' BorderThickness='3’
            CornerRadius='3' BorderBrush='Black' Padding='5'>
        <TextBlock>Hello 2</TextBlock>
    </Border>
    <Border Canvas.Left='15' Canvas.Top='50' BorderThickness='10,1,10,1’
            CornerRadius='10' BorderBrush='Black' Padding='5'>
        <TextBlock>Hello 3</TextBlock>
    </Border>
    <Border Canvas.Left='85' Canvas.Top='50' BorderThickness='4,1,4,1’
            CornerRadius='0,15,0,15' BorderBrush='Black' Padding='5'>
        <TextBlock>Hello 4</TextBlock>
    </Border>
</Canvas>
```


SCROLLVIEWER

- Provides a convenient way to enable scrolling of content
  - encapsulates horizontal and vertical **ScrollBar** elements and a content container
- Can only have one child
  - typically a Panel



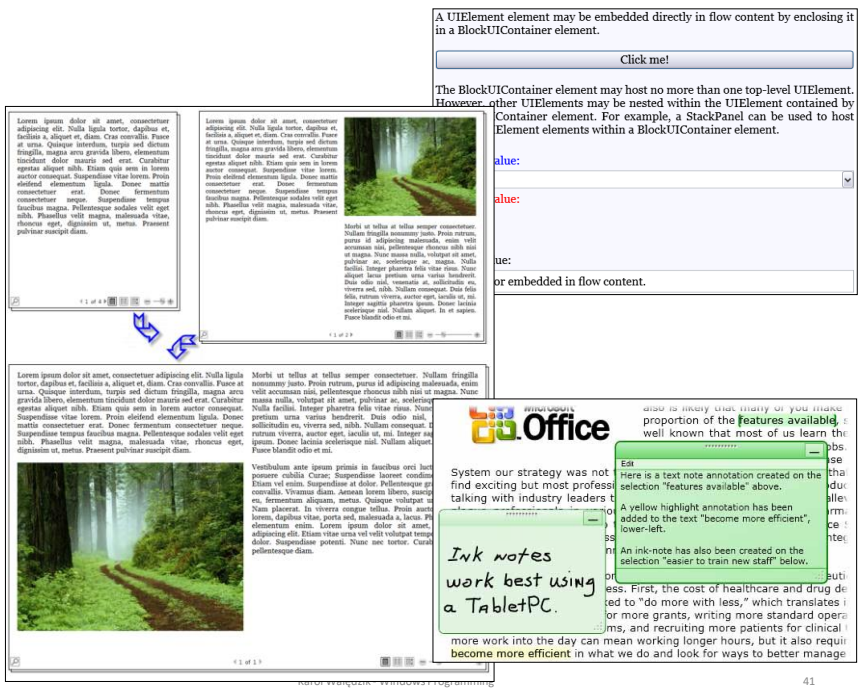

```
<ScrollViewer HorizontalScrollBarVisibility="Visible">
    <DockPanel>
        <Button Width="400" Height="50">My button</Button>
        <Button>My second button</Button>
    </DockPanel>
</ScrollViewer>
```

VIEWBOX

- content decorator that can stretch and scale a single child to fill the available space

```
<StackPanel Orientation="Vertical">
    <StackPanel Margin="0,0,0,10" HorizontalAlignment="Center"
        Orientation="Horizontal" DockPanel.Dock="Top">
        <Button Name="btn1" Click="stretchNone">Stretch="None"</Button>
        <Button Name="btn2" Click="stretchFill">Stretch="Fill"</Button>
        <Button Name="btn3" Click="stretchUni">Stretch="Uniform"</Button>
        <Button Name="btn4" Click="stretchUniFill">Stretch="UniformToFill"</Button>
    </StackPanel>
    <StackPanel Margin="0,0,0,10" HorizontalAlignment="Center"
        Orientation="Horizontal" DockPanel.Dock="Top">
        <Button Name="btn5" Click="sdUpOnly">StretchDirection="UpOnly"</Button>
        <Button Name="btn6" Click="sdDownOnly">StretchDirection="DownOnly"</Button>
        <Button Name="btn7" Click="sdBoth">StretchDirection="Both"</Button>
    </StackPanel>
    <TextBlock DockPanel.Dock="Top" Name="txt1" />
    <TextBlock DockPanel.Dock="Top" Name="txt2" />
    <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
        <Viewbox MaxWidth="500" MaxHeight="500" Name="vb1">
            <Image Source="Images/Fiona_67x77.gif"/>
        </Viewbox>
    </StackPanel>
</StackPanel>
```

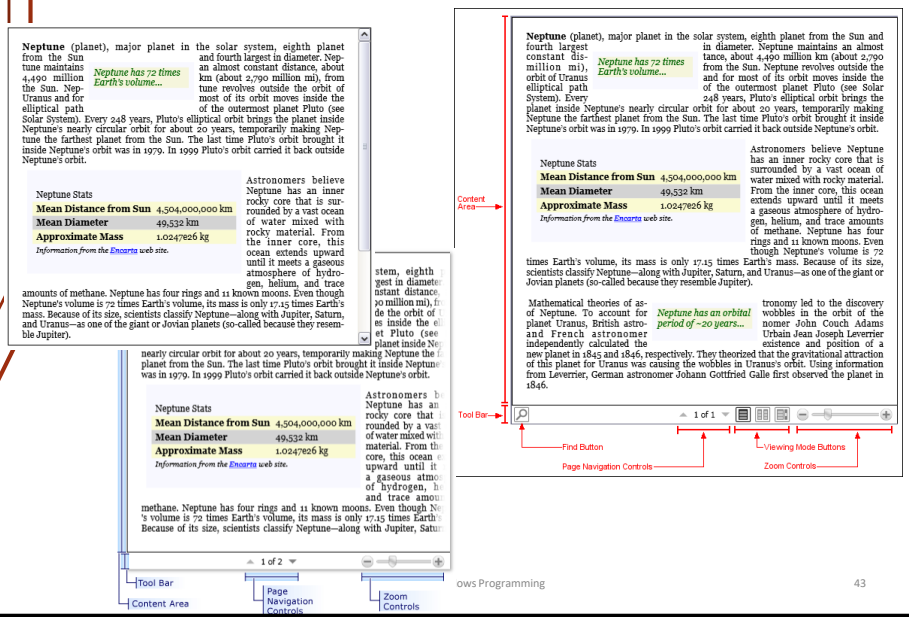
```
public void stretchNone(object sender, RoutedEventArgs e)
{
    vb1.Stretch = System.Windows.Media.Stretch.None;
    txt1.Text = "Stretch is now set to None.";
}
```



FlowDocument viewers

- **FlowDocumentScrollViewer** – continuous scrolling view
- **FlowDocumentPageViewer** – paginated view, one page at a time
- **FlowDocumentReader** – single-page view, multi-page view, or scrolling view

FlowDocument viewers cont'd



LAYOUT

LAYOUT SYSTEM

- Bounding box
  - rectangle surrounding each element
  - includes its padding and margin
  - **LayoutInformation.GetLayoutSlot**(element)
- Layout system
  - process of measuring and arranging members of any **Panel**'s **Children** collection

Layout System Steps

1. Measurement
   - an implicit call to Measure()
     - it evaluates **UIElement**'s properties (e.g. **Clip**, **Visbility**)
     - **MeasureCore**() is invoked with calculated **constraintSize**
       - processes **FrameworkElement** sizing properties (e.g. **Width**, **Height** etc.)
       - **MeasureOverride**() is invoked with the updated **contraintSize**
       - ultimately **DesiredSize** is calculated
2. Application of panel-specific logic

Layout System Steps cont'd

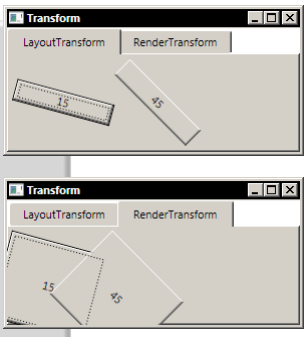
3. Arranging of all children elements
   - implicit call to **Arrange**() method with panel-generated bounding rectangle
     - internally **ArrangeCore**() is invoked
       - **DesiredSize** and margins are evaluated to create **arrangeSize**
       - **arrangeSize** is passed to **ArrangeOverride**
       - offset properties (e.g. alignment) are finally applied
4. Drawing of all the children

- Process is repeated for each and every change to **Children** collection or application of **LayoutTransform** (or call to **UpdateLayout**)

TRANSFORMS & Z-INDEX

- **RenderTransform** – applied immediately before rendering
- **LayoutTransform** – applied before layout
- **z-index** – equal to 0 by default

```
<TabControl>
    <TabItem Header="LayoutTransform">
        <StackPanel Orientation="Horizontal">
            <Button Width="100">15
                <Button.LayoutTransform>
                    <RotateTransform Angle="15"/>
                </Button.LayoutTransform>
            </Button>
            <Button Width="100">45
                <Button.LayoutTransform>
                    <RotateTransform Angle="45"/>
                </Button.LayoutTransform>
            </Button>
        </StackPanel>
    </TabItem>
    <!-- Analogously, but using the RenderTransform -->
</TabControl>
```



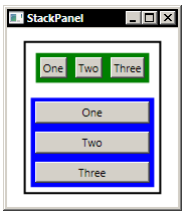
PANELS

CANVAS

- Simplest panel
- **Canvas.Left**, **Canvas.Right**, **Canvas.Top**, **Canvas.Bottom**

```
<Canvas Width='200' Height='100' Background="AntiqueWhite" >
    <Button Canvas.Right='10' Canvas.Top='10'>Top, Right</Button>
    <Button Canvas.Left='10' Canvas.Top='10'>Top, Left</Button>
    <Button Canvas.Right='10' Canvas.Bottom='10'>Bottom,
Right</Button>
    <Button Canvas.Left='10' Canvas.Bottom='10'>Bottom, Left</Button>
    <Button Canvas.Left='30' Canvas.Bottom='30'
     Canvas.Right='30' Canvas.Top='30'>
        All Four
    </Button>
</Canvas>
```

STACKPANEL

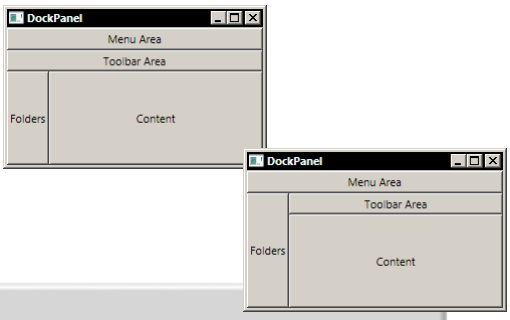


- **Orientation**
- always stretches to encompass all children
  - therefore: putting ScrollViewer inside it has no effect

```
<Border BorderBrush='Black' BorderThickness='2'
  HorizontalAlignment='Center' VerticalAlignment='Center'>
    <StackPanel Orientation='Vertical'>
        <StackPanel Margin='10' Background='Green' Orientation='Horizontal'>
            <Button Margin='4'>One</Button>
            <Button Margin='4'>Two</Button>
            <Button Margin='4'>Three</Button>
        </StackPanel>
        <StackPanel Margin='5' Background='Blue' Orientation='Vertical'>
            <Button Margin='4'>One</Button>
            <Button Margin='4'>Two</Button>
            <Button Margin='4'>Three</Button>
        </StackPanel>
    </StackPanel>
</Border>
```

DockPanel

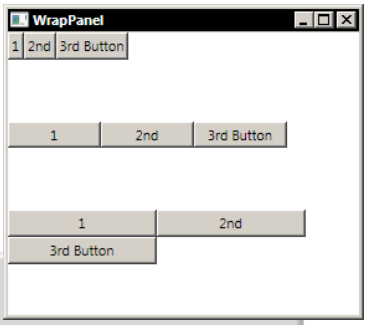
- DockPanel.Dock
- LastChildFill




```
<DockPanel>
    <Button DockPanel.Dock='Top'>Menu Area</Button>
    <Button DockPanel.Dock='Top'>Toolbar Area</Button>
    <Button DockPanel.Dock='Left'>Folders</Button>
    <Button>Content</Button>
</DockPanel>
```

```
<DockPanel>
    <Button DockPanel.Dock='Top'>Menu Area</Button>
    <Button DockPanel.Dock='Left'>Folders</Button>
    <Button DockPanel.Dock='Top'>Toolbar Area</Button>
    <Button>Content</Button>
</DockPanel>
```

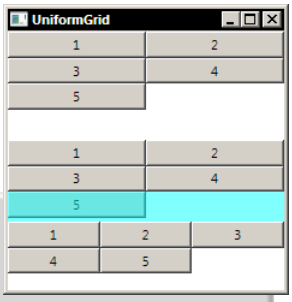

WrapPanel




```
<StackPanel>
    <WrapPanel>
        <Button>1</Button>
        <Button>2nd</Button>
        <Button>3rd Button</Button>
    </WrapPanel>
    <Rectangle Height="50"/>
    <WrapPanel ItemWidth="75">[...]</WrapPanel>
    <Rectangle Height="50"/>
    <WrapPanel ItemWidth="120">[...]</WrapPanel>
</StackPanel>
```

UniformGrid

- Columns, Rows



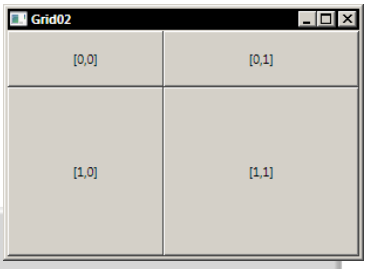

```
<StackPanel>
    <UniformGrid Columns="2">
        <Button>1</Button>
        <Button>2</Button>
        <Button>3</Button>
        <Button>4</Button>
        <Button>5</Button>
    </UniformGrid>
    <Rectangle Height="25"/>
    <UniformGrid Columns="2" Rows="2">[...]</UniformGrid>
    <Rectangle Height="25" Opacity="0.5" Fill="Cyan" />
    <UniformGrid Rows="2">[...]</UniformGrid>
</StackPanel>
```

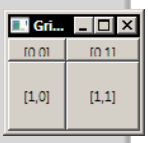

Grid

- RowDefinitions, ColumnDefinitions
- Grid.Row, Grid.Column


```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Button Grid.Row='0' Grid.Column='0' >[0,0]</Button>
    <Button Grid.Row='1' Grid.Column='1' >[1,1]</Button>
    <Button Grid.Row='0' Grid.Column='1' >[0,1]</Button>
    <Button Grid.Row='1' Grid.Column='0' >[1,0]</Button>
</Grid>
```

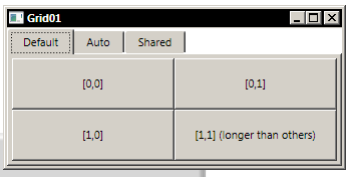
Grid cont'd
Star Sizing

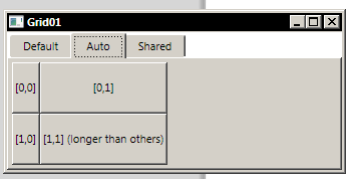


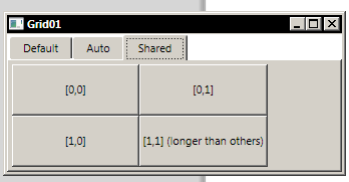



```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="1*"/>
        <RowDefinition Height="3*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="40*"/>
        <ColumnDefinition Width="50*"/>
    </Grid.ColumnDefinitions>
    <Button Grid.Row='0' Grid.Column='0' >[0,0]</Button>
    <Button Grid.Row='1' Grid.Column='1' >[1,1]</Button>
    <Button Grid.Row='0' Grid.Column='1' >[0,1]</Button>
    <Button Grid.Row='1' Grid.Column='0' >[1,0]</Button>
</Grid>
```


Grid cont'd
Size-Sharing





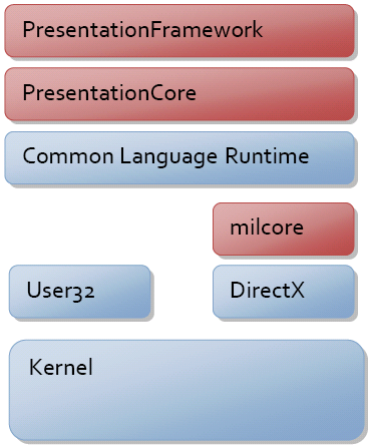



```
<Grid>
    [...]
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    [...]
</Grid>
[...]
<Grid>
    [...]
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="Auto"/>
    </Grid.ColumnDefinitions>
    [...]
</Grid>
[...]
<Grid Grid.IsSharedSizeScope="True">
    [...]
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" SharedSizeGroup="ssg01"/>
        <ColumnDefinition Width="Auto" SharedSizeGroup="ssg01"/>
    </Grid.ColumnDefinitions>
    [...]
</Grid>
```

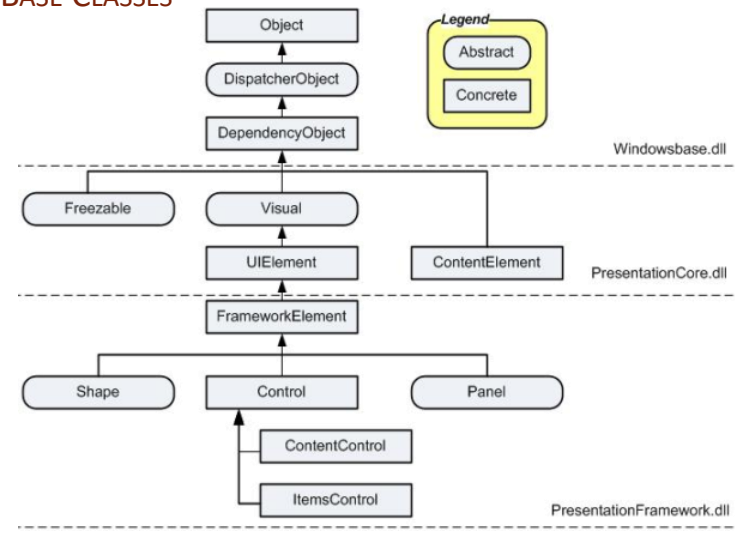
WPF Architecture

WPF Architecture

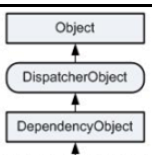
- **PresentationFramework**
  - end-user presentation features (incl. layouts, animations, and data-binding)
- **PresentationCore**
  - managed wrapper for MIL
  - implements the core services
- milcore – Media Integration Layer
  - native, performance-critical component
  - interfaces directly with DirectX



WPF Base Classes

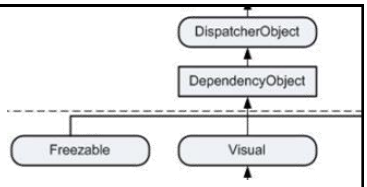


WPF Base Classes cont'd



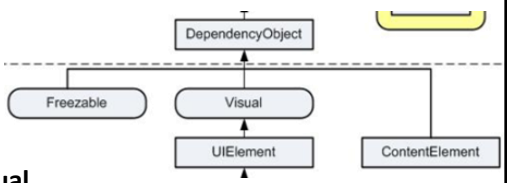
- System.Threading.**DispatcherObject**
  - provides STA (single thread affinity) behavior
    - each control is associated with a single thread and cannot be manipulated from any other thread
  - provides basic constructs for dealing with concurrency and threading
  - cross-thread message dispatching system, with multiple prioritized queues
    - actual implementation bases on User32 calls
  - allows execution of a method on a thread associated with any particular WPF object

WPF Base Classes cont'd



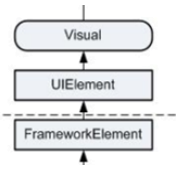
- System.Windows.**DependencyObject**
  - provides new mechanism for storing data in properties
    - change notifications
    - sparse storage
    - attached properties
    - makes data binding possible

WPF Base Classes cont'd



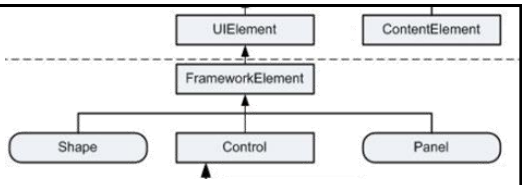
- System.Windows.Media.**Visual**
  - provides for building a tree of visual objects
    - each containing drawing instructions and metadata
  - point of connection between managed code and milcore
  - internally visual tree is mapped into an unmanaged composition tree
    - makes use of retained rendering system
    - painter's algorithm used instead of forced clipping employed typically by User32 and GDI
    - declarative rather than imperative drawing model

WPF Base Classes cont'd



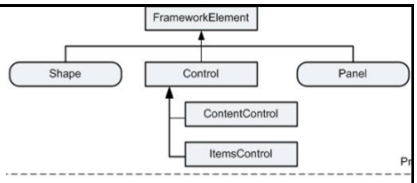
- System.Windows.**UIElement**
  - defines the 3 core subsystems for the WPF object:
    - **Layout**
      - 2-phase model:
        - Measure
        - Arrange
    - **Input**
      - signal on a kernel model driver is routed to the correct process and thread via Windows kernel un User32 messages
        - one signal may result in several events
    - **Events**
      - routed events-based system

WPF Base Classes cont'd



- System.Windows.**FrameworkElement**
  - provides common API to services offered higher in the object hierarchy
    - e.g.: HorizontalAlignment, MinWidth, Margin etc.
    - e.g.: BeginStoryboard()
  - adds support for data binding
  - adds support for styles

WPF Base Classes cont'd



- System.Windows.Controls.**Control**
  - unlike in Windows.Forms not every visual element is a control
  - provides full support for 3 main models of each control:
    - data model
      - properties of the control: common to all controls (e.g. Foreground, Background, Padding etc.) and control-specific
      - allows customization of interaction model and display model
    - interaction model
      - set of commands, events and actions possible
    - display model
      - fully customizable via use of templates

WPF Property System

- Typically:
  - simple properties exposed as regular CLR properties

  but:
  - backed up by dependency properties
- Dependency properties:
  - may be computed based on other inputs (themes, styles, templates, data binding, animations)
  - may provide validation, default values, change notifications, value coercion based on runtime state etc.
  - may have metadata attached
  - get better support from WPF designers, support property value inheritance in the element tree

Event Routing

- 'A routed event is a type of event that can invoke handlers on multiple listeners in an element tree, rather than just on the object that raised the event.'
- Routing strategies:
  - Direct
    - event originates in one element and is not passed to any other (as in Windows Forms)
    - e.g.: MouseEnter, MouseLeave
  - Bubbling
    - event travels up the hierarchy
    - most input or state change events e.g. MouseDown
  - Tunneling
    - event travels down the tree towards its source
    - preview events for most input event, e.g.: PreviewKeyDown
      - preview events can be suppressed or replaced by custom events by controls (e.g.: Click)
      - if preview event is marked as handled, normal event will not fire

RoutedEventArgs

- Base class for event arguments of all WPF events
- Properties:
  - **Source**
    - object that raised the event
    - may be different from sender – i.e. the object that is currently handling the event
  - **OriginalSource**
    - object that originally raised the event
    - read-only
    - may be different from Source if source adjustment has been performed
      - e.g. ListBox will report ListItem as source, OriginalSource will point to the element inside ListItem

UI Resources

UI Resources

- **FrameworkElement.Resources**, **FrameworkContentElement.Resources**, **Application.Resources**
  - type: **ResourceDictionary**
  - each resource identified by a unique key
- Can be stored at any level in hierarchy
  - typically: in **Window.Resources**, **Page.Resources** or **UserControl.Resources**
- In markup:
  - key defined via **x:Key** attribute
  - resources accessible via **StaticResource** and **DynamicResource** markup extensions

UI Resources cont'd

```
<Window
   xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
   xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
   x:Class="SDKSample.ResourcesWindow" Title="Resources Window">

   <Window.Resources>
      <SolidColorBrush x:Key="defaultBackground" Color="Red" />
   </Window.Resources>

   ...

   <Button Background="{StaticResource defaultBackground}">
      One Button
   </Button>

   <Label Background="{StaticResource defaultBackground}">
      One Label
   </Label>

   ...
</Window>
```

```
btn.Background = this.Resources["background"]
```

http://msdn.microsoft.com/en-us/library/aa970268.aspx

UI Resources cont'd

- Resources defined at application level are available to all other elements

```
<Application
   x:Class="WpfApplication1.App"
   xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
   xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
   StartupUri="Data/Resources.xaml">
    <Application.Resources>
        <SolidColorBrush x:Key="MainBrush">
       Black
     </SolidColorBrush>
    </Application.Resources>
</Application>
```

```
<Window [...]>
    <Ellipse DockPanel.Dock="Top" HorizontalAlignment="Left" Width="100"
          Height="100" Fill="{StaticResource MainBrush}" Margin="40" />
</Window>
```

UI Resources cont'd

- It is possible to store resources in separate files and attach them as needed

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

   <SolidColorBrush x:Key="defaultBackground" Color="Red" />
   ...
</ResourceDictionary>
```

```
<Application.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary Source="BackgroundColorResources.xaml"/>
        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</Application.Resources>
```

http://msdn.microsoft.com/en-us/library/aa970268.aspx

Dynamic resources

- Can be modified at runtime
  - however: introduce some performance overhead

```
<Ellipse DockPanel.Dock="Top" HorizontalAlignment="Left"
         Width="100" Height="100" Margin="40"
         Fill="{DynamicResource MainBrush}"
         MouseDown="Ellipse_MouseDown" />
```

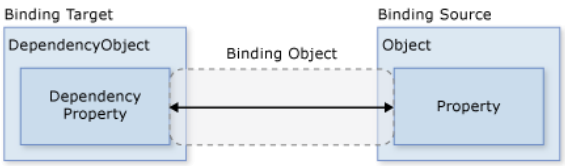
```
private void Ellipse_MouseDown(object sender,
                               MouseButtonEventArgs e)
{
    Application.Current.Resources["MainBrush"] =
                        new SolidColorBrush(Colors.Blue);
}
```

```
button1.SetResourceReference(Button.BackgroundProperty, "MainBrush");
```

DATA BINDING

DATA BINDING

- "Data binding is the process that establishes a connection between the application UI and business logic"
- 4 components:
  - binding source
  - path to the value in the binding source
  - target
  - target dependency property



http://msdn.microsoft.com/en-us/library/ms752347.aspx

Binding Source

- Different ways to define the binding source:
  - ElementName
    - binding to other UI element
  - Source
    - binding to public property of any object
  - RelativeSource
    - search relative to the target object
  - DataContext
    - context set once for a whole hierarchy of elements

Source

```
MyData myDataObject = new MyData(DateTime.Now);
Binding myBinding = new Binding("MyDataProperty");
myBinding.Source = myDataObject;
myText.SetBinding(TextBlock.TextProperty, myBinding);
```

```
<TextBlock Text="{Binding
                Source={x:Static SystemFonts.IconFontFamily}}" />
```

```
<DockPanel.Resources>
    <c:MyData x:Key="myDataSource"/>
</DockPanel.Resources>

<Button Width="150" Height="30”
   Background="{Binding Source={StaticResource myDataSource},
                        Path=ColorName}">
    I am bound to be RED!
</Button>
```


http://msdn.microsoft.com/en-us/library/ms752347.aspx

ELEMENTNAME

- Warning: binding errors do not cause application to fail
  - when developing in Visual Studio: visible in the output window

```
<TextBlock Text="{Binding ElementName=textBox, Path=Text}”
       FontSize="{Binding ElementName=sizeSlider, Path=Value}“
       FontFamily= "{Binding ElementName=fontFamilyComboBox,
                                    Path=SelectedItem}" >
    <TextBlock.Foreground>
      <SolidColorBrush Color="{Binding ElementName=colorsComboBox,
                                    Path=SelectedItem}" />
    </TextBlock.Foreground>
</TextBlock>
```


http://msdn.microsoft.com/en-us/library/ms752347.aspx

RELATIVESOURCE

- **RelativeSourceMode**
  - { PreviousData, TemplatedParent, Self, FindAncestor }

```
<TextBlock>
  <TextBlock.Text>
      <Binding Path="Title">
        <Binding.RelativeSource>
          <RelativeSource Mode="FindAncestor”
                  AncestorType="{x:Type Window}" />
        </Binding.RelativeSource>
      </Binding>
    </TextBlock.Text>
</TextBlock>
```

```
<TextBlock Text="{Binding Path=Title,
        RelativeSource= {RelativeSource FindAncestor,
                                    AncestorType={x:Type Window}}}">
</TextBlock>
```

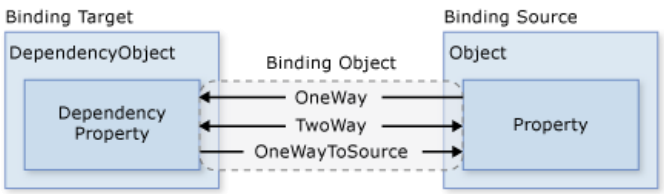
DataContext

```
<DockPanel
   xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
   xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
   xmlns:c="clr-namespace:SDKSample„>
   <DockPanel.Resources>
      <c:MyData x:Key="myDataSource"/>
   </DockPanel.Resources>
   <DockPanel.DataContext>
      <Binding Source="{StaticResource myDataSource}"/>
   </DockPanel.DataContext>
   <Button Background="{Binding Path=ColorName}”
           Width="150" Height="30">
         I am bound to be RED!
   </Button>
</DockPanel>
```

http://msdn.microsoft.com/en-us/library/ms752347.aspx

Data Flow Direction

- **INotifyPropertyChanged**
- **BindingMode**
  - { OneWay, TwoWay, OneWayToSource, OneTime, Default }




```
<TextBlock Text="{Binding Path=TotalIncome, Mode=OneTime}"/>
```

INotifyPropertyChanged

- INotifyPropertyChanged
  - PropertyChanged event

```
public class Customer : INotifyPropertyChanged
{
   private string _firstName;
   public string FirstName
   {
      get { return _firstName; }
      set { _firstName = value;  NotifyPropertyChanged("FirstName"); }
   }

   public event PropertyChangedEventHandler PropertyChanged;
   private void NotifyPropertyChanged(string propertyName)
   {
   if (PropertyChanged != null)
      PropertyChanged(this,
         new PropertyChangedEventArgs(propertyName));
   }

...
```

Source Update Triggers

- **UpdateSourceTrigger**
  - { Default, PropertyChanged, LostFocus, Explicit }

```
<TextBox>
  <TextBox.Text>
    <Binding Source="{StaticResource myDataSource}“
             Path="Name”
             UpdateSourceTrigger="PropertyChanged"/>
  </TextBox.Text>
</TextBox>
```

Other Configuration Possibilities

- BindingGroupName
- FallbackValue
- StringFormat
- TargetNullValue

Data Conversion

```
[ValueConversion(typeof(DateTime), typeof(String))]
public class DateConverter : IValueConverter
{
   public object Convert(object value, Type targetType,
                         object parameter, CultureInfo culture)
   {
      DateTime date = (DateTime)value;
      return date.ToShortDateString();
   }

   public object ConvertBack(object value, Type targetType,
                             object parameter, CultureInfo culture)
   {
      string strValue = value as string;
      DateTime resultDateTime;
      if (DateTime.TryParse(strValue, out resultDateTime))
      {
         return resultDateTime;
      }
      return DependencyProperty.UnsetValue;
   }
}
```

http://msdn.microsoft.com/en-us/library/system.windows.data.ivalueconverter.aspx

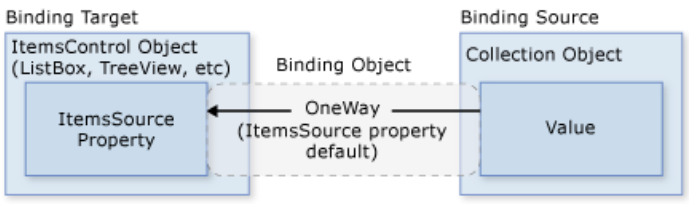
Data Conversion cont'd

```
...

<Window.Resources>
  <src:DateConverter x:Key="dateConverter"/>
<Window.Resources>

...

<TextBlock   Margin="0,0,8,0" Name="startDateTitle”
         Style="{StaticResource smallTitleStyle}">
  Start Date:
</TextBlock>

<TextBlock Name="StartDateDTKey”
         Text="{Binding Path=StartDate,
       Converter={StaticResource dateConverter}}”/>
```

http://msdn.microsoft.com/en-us/library/system.windows.data.ivalueconverter.aspx

Binding to Collections

- **INotifyCollectionChanged**
  - **ObservableCollection<T>**



http://msdn.microsoft.com/en-us/library/ms752347.aspx

Collection Views

- Layer on top of a binding source collection
  - allows sorting, filtering and grouping
  - can be used as ItemsSource
    - when collection used directly as ItemsSource, a default collection view is automatically created (one per collection)
      - CollectionViewSource.GetDefaultView()
  - INotifyCollectionChanged events are propagated to views
  - each collection can have multiple associated views
  - several actual classes dedicated to various collections:
    - internal type for IEnumerable
    - ListCollectionView for IList
    - BindingListCollectionView for IBindingList
  - easiest way to create: use CollectionViewSource

Collection Views cont'd

```
<Window.Resources>
   <CollectionViewSource
      Source="{Binding Source={x:Static Application.Current},
                                    Path=AuctionItems}"
      x:Key="listingDataView" />
</Window.Resources>

...

<ListBox Name="Master" Grid.Row="2" Grid.ColumnSpan="3" Margin="8"
      ItemsSource="{Binding Source={StaticResource listingDataView}}">
```

```
listingDataView.SortDescriptions.Add( new SortDescription("StartDate",  ListSortDirection.Ascending));
listingDataView.Filter += new FilterEventHandler(ShowOnlyBargainsFilter);

...

private void ShowOnlyBargainsFilter(object sender, FilterEventArgs e)
{
      AuctionItem product = e.Item as AuctionItem;
      if (product != null)   { e.Accepted = product.CurrentPrice < 25; }
}
```

http://msdn.microsoft.com/en-us/library/ms752347.aspx

Current Item Pointers

```
<Button Content="{Binding }" />

<Button Content="{Binding Path=/}" />

<Button Content="{Binding Path=/Description}" />

<Button Content="{Binding /Offices/}" />
```

http://msdn.microsoft.com/en-us/library/ms752347.aspx

Master-Detail Scenario

```
<ListBox Name="Master”
     ItemsSource=
          "{Binding Source={StaticResource listingDataView}}">
...
</ListBox>

...

<ContentControl Name="Detail”
   Content="{Binding Source={StaticResource listingDataView}}”
   ContentTemplate=
          "{StaticResource detailsProductListingTemplate}”/>
```

BINDINGOPERATIONS & BINDINGEXPRESSION

- **Binding**
  - high-level class for binding declaration
  - same object may be reused for several bindings
- **BindingExpression**
  - represents a single instance of **Binding**
- **BindingOperations**
  - static class for bindings manipulation

```
MyData myDataObject = new MyData(DateTime.Now);
Binding myBinding = new Binding("MyDataProperty");
myBinding.Source = myDataObject;
myText.SetBinding(TextBlock.TextProperty, myBinding);

BindingExpression bindingExpression =
   BindingOperations.GetBindingExpression(myText, TextBox.TextProperty);
```


MULTIBINDING

```
public class NameConverter : IMultiValueConverter
{
    public object Convert(object[] values, Type targetType,
                          object parameter, CultureInfo culture)
    { return name = values[0] + " " + values[1]; }

    public object[] ConvertBack(object value, Type[] targetTypes,
                          object parameter, CultureInfo culture)
    { return ((string)value).Split(' ');  }
}
```

```
<TextBlock Name="textBox2„
     DataContext="{StaticResource PersonData}">
  <TextBlock.Text>
    <MultiBinding
      Converter="{StaticResource myNameConverter}">
      <Binding Path="FirstName"/>
      <Binding Path="LastName"/>
    </MultiBinding>
  </TextBlock.Text>
</TextBlock>
```

```
public class PersonData {
    public string FirstName {get; set;}
    public string LastName {get; set;} }
```

PRIORITYBINDING

```
<StackPanel HorizontalAlignment="Center„
  VerticalAlignment="Center„
  DataContext="{Binding Source={StaticResource AsyncDS}}">
  <TextBlock Background="Honeydew„
      Width="100" HorizontalAlignment="Center">
    <TextBlock.Text>
      <PriorityBinding FallbackValue="defaultvalue">
        <Binding Path="SlowestDP" IsAsync="True"/>
        <Binding Path="SlowerDP" IsAsync="True"/>
        <Binding Path="FastDP" />
      </PriorityBinding>
    </TextBlock.Text>
  </TextBlock>
</StackPanel>
```

http://msdn.microsoft.com/en-us/library/system.windows.data.prioritybinding.aspx

STYLES

STYLES

```
<Window.Resources>
   <!--A Style that affects all TextBlocks-->
   <Style TargetType="TextBlock">
      <Setter Property="FontFamily" Value="Comic Sans MS"/>
      <Setter Property="FontSize" Value="14"/>
   </Style>

   <Style BasedOn="{StaticResource {x:Type TextBlock}}”
        TargetType="TextBlock" x:Key="TitleText„>
      <Setter Property="FontSize" Value="26"/>
         <Setter Property="Foreground„>
            <Setter.Value>
               <LinearGradientBrush StartPoint="0.5,0“
                                    EndPoint="0.5,1„>
                  ...
               </LinearGradientBrush>
            </Setter.Value>
         </Setter>
         <EventSetter Event=“MouseEnter" Handler="b1SetColor"/>
      </Style>
</Window.Resources>
```

http://msdn.microsoft.com/en-us/library/ms745683.aspx

TRIGGERS

```
<Style x:Key="Triggers" TargetType="Button">
    <Style.Triggers>
        <Trigger Property="IsPressed" Value="true">
            <Setter Property="Foreground" Value="Green"/>
        </Trigger>
    </Style.Triggers>
</Style>
```

```
<Trigger Property="IsMouseOver" Value="True">
    <Trigger.EnterActions>
        <BeginStoryboard>
            <Storyboard>
                <DoubleAnimation Storyboard.TargetProperty="Opacity" To="1„
                                                        Duration="0:0:1" />
            </Storyboard>
        </BeginStoryboard>
    </Trigger.EnterActions>
    <Trigger.ExitActions>
        <BeginStoryboard>
            <Storyboard>
                <DoubleAnimation Storyboard.TargetProperty="Opaci
                                 To="0.25„ Duration="0:0:1" />
            </Storyboard>
        </BeginStoryboard>
    </Trigger.ExitActions>
</Trigger>
```

http://msdn.microsoft.com/en-us/library/system.windows.triggerbase.enteractions.aspx

DATA TRIGGERS

```
<Style TargetType="ListBoxItem">
   <Style.Triggers>
      <DataTrigger Binding="{Binding Path=State},,
                                                Value="WA">
         <Setter Property="Foreground" Value="Red" />
      </DataTrigger>
      <MultiDataTrigger>
         <MultiDataTrigger.Conditions>
            <Condition Binding="{Binding Path=Name},,
                                       Value="Portland" />
            <Condition Binding="{Binding Path=State},,
                                                Value="OR" />
         </MultiDataTrigger.Conditions>
         <Setter Property="Background" Value="Cyan" />
      </MultiDataTrigger>
   </Style.Triggers>
</Style>
```

http://msdn.microsoft.com/en-us/library/system.windows.datatrigger.aspx

MULTITRIGGERS

```
<Style.Triggers>
  <Trigger Property="IsEnabled" Value="false">
    <Setter Property="Background" Value="#EEEEEE" />
  </Trigger>
  <MultiTrigger>
    <MultiTrigger.Conditions>
      <Condition Property="HasItems" Value="false" />
      <Condition Property="Width" Value="Auto" />
    </MultiTrigger.Conditions>
    <Setter Property="MinWidth" Value="120"/>
  </MultiTrigger>
  <MultiTrigger>
    <MultiTrigger.Conditions>
      <Condition Property="HasItems" Value="false" />
      <Condition Property="Height" Value="Auto" />
    </MultiTrigger.Conditions>
    <Setter Property="MinHeight" Value="95"/>
  </MultiTrigger>
</Style.Triggers>
```

http://msdn.microsoft.com/en-us/library/system.windows.multitrigger.aspx

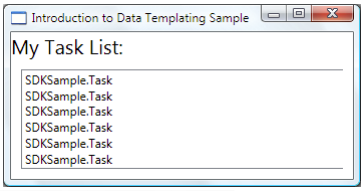
TEMPLATES

CONTROL TEMPLATES

- **Control.Template**
  - of type **ControlTemplate**

```
<Style TargetType="Button„>
   <!--Set to true to not get any properties from the themes.→
   <Setter Property="OverridesDefaultStyle" Value="True"/>
   <Setter Property="Template„>
      <Setter.Value>
         <ControlTemplate TargetType="Button„>
            <Grid>
               <Ellipse Fill="{TemplateBinding Background}"/>
               <ContentPresenter HorizontalAlignment="Center”
                  VerticalAlignment="Center"/>
            </Grid>
         </ControlTemplate>
      </Setter.Value>
   </Setter>
</Style>
```

Data Templates

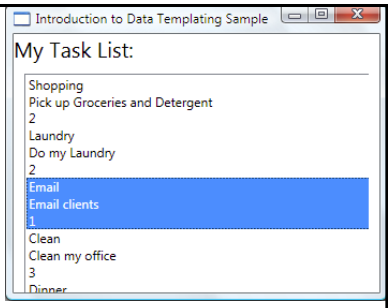



```
<Window x:Class="SDKSample.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="clr-namespace:SDKSample" Title="Introduction to Data
Templating Sample„>

   <Window.Resources>
     <local:Tasks x:Key="myTodoList"/>
     ...
   </Window.Resources>

   <StackPanel>
     <TextBlock Name="blah" FontSize="20" Text="My Task List:"
       <ListBox Width="400" Margin="10”
          ItemsSource=
             "{Binding Source={StaticResource myTodoList}}"/>
        ...
       </StackPanel>
</Window>
```

http://msdn.microsoft.com/en-us/library/ms742521.aspx

Data Templates

- **ContentControl.ContentTemplate, ItemsControl.ItemTemplate**
- **DataTemplate, HierarchicalDataTemplate**
- TemplateBinding markup extension
- Template can be created as a resource




```
<ListBox Width="400" Margin="10”
   ItemsSource="{Binding Source={StaticResource myTodoList}}„>
   <ListBox.ItemTemplate>
     <DataTemplate>
       <StackPanel>
         <TextBlock Text="{Binding Path=TaskName}" />
         <TextBlock Text="{Binding Path=Description}"/>
         <TextBlock Text="{Binding Path=Priority}"/>
       </StackPanel>
     </DataTemplate>
   </ListBox.ItemTemplate>
</ListBox>
```

http://msdn.microsoft.com/en-us/library/ms742521.aspx

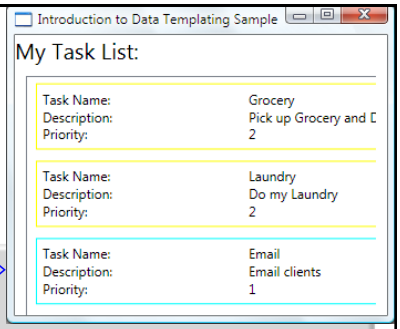
Type-Specific Templates

- Same template for many controls
- Allows displaying heterogeneous lists

```
<DataTemplate DataType="{x:Type local:Task}”>
  <StackPanel>
    <TextBlock Text="{Binding Path=TaskName}" />
    <TextBlock Text="{Binding Path=Description}"/>
    <TextBlock Text="{Binding Path=Priority}"/>
  </StackPanel>
</DataTemplate>
```

http://msdn.microsoft.com/en-us/library/ms742521.aspx

Data Triggers




```
<DataTemplate x:Key="myTaskTemplate">

...

  <DataTemplate.Triggers>
    <DataTrigger Binding="{Binding Path=TaskType}„>
      <DataTrigger.Value>
        <local:TaskType>Home</local:TaskType>
      </DataTrigger.Value>
      <Setter TargetName="border”
           Property="BorderBrush" Value="Yellow"/>
    </DataTrigger>
  </DataTemplate.Triggers>

...

</DataTemplate>
```

http://msdn.microsoft.com/en-us/library/ms742521.aspx

TEMPLATE SELECTORS

```
public class TaskListDataTemplateSelector : DataTemplateSelector
{
   public override DataTemplate SelectTemplate(
               object item, DependencyObject container)
   {
      FrameworkElement element = container as FrameworkElement;
      if (element != null && item != null && item is Task)
      {
         Task taskitem = item as Task;
         if (taskitem.Priority == 1)
         {
            return element.FindResource("importantTaskTemplate")
                                        as DataTemplate;
         }
         else
         {
            return element.FindResource("myTaskTemplate„)
                                        as DataTemplate;
         }
      }
      return null;
   }
}
```

http://msdn.microsoft.com/en-us/library/ms742521.aspx

TEMPLATE SELECTORS CONT'D

```
<Window.Resources>
   ...
   <local:TaskListDataTemplateSelector
                x:Key="myDataTemplateSelector"/>
   ...
</Window.Resources>

<ListBox Width="400" Margin="10”
   ItemsSource=
      "{Binding Source={StaticResource myTodoList}}”
   ItemTemplateSelector=
      "{StaticResource myDataTemplateSelector}”
   HorizontalContentAlignment="Stretch"/>
```

http://msdn.microsoft.com/en-us/library/ms742521.aspx

VALIDATION

DATA VALIDATION

- **ValidationRule**:
  - **ExceptionValidationRule**
    - equivalent to setting ValidatesOnExceptions="true"
  - **DataErrorValidationRule**
    - equivalent to setting ValidatesOnDataErrors=„true"
  - custom classes

```
<TextBox Name="StartPriceEntryForm" Grid.Row="2" Grid.Column="1"
     Style="{StaticResource textStyleTextBox}" Margin="8,5,0,5">
  <TextBox.Text>
    <Binding Path="StartPrice" UpdateSourceTrigger="PropertyChanged„>
      <Binding.ValidationRules>
        <ExceptionValidationRule />
      </Binding.ValidationRules>
    </Binding>
  </TextBox.Text>
</TextBox>
```

http://msdn.microsoft.com/en-us/library/ms752347.aspx

DataErrorValidationRule

- Equivalent to setting ValidatesOnDataErrors=„true”
  - True by default
- The most popular way to handle validation
- Expects implementation of one of two interfaces:
  - **IDataErrorInfo**
  - **INotifyDataErrorInfo**

Custom ValidationRule

```
class FutureDateRule : ValidationRule
{
  public override ValidationResult Validate(object value,
                              CultureInfo cultureInfo)
  {
    DateTime date;
    if (!DateTime.TryParse(value.ToString(), out date))
    {
      return new ValidationResult(false,
                        "Value is not a valid date.");
    }
    if (DateTime.Now.Date > date)
    {
      return new ValidationResult(false,
                    "Please enter a date in the future.");
    }
    else { return ValidationResult.ValidResult;}
  }
}
```

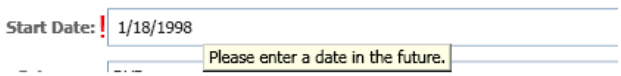
CUSTOM VALIDATIONRULE

```
<TextBox Name="StartDateEntryForm”
         Grid.Row="3" Grid.Column="1”
         Validation.ErrorTemplate=
                    "{StaticResource validationTemplate}”
         Style="{StaticResource textStyleTextBox}”
         Margin="8,5,0,5„>
  <TextBox.Text>
    <Binding Path="StartDate”
           UpdateSourceTrigger="PropertyChanged”
           Converter="{StaticResource dateConverter}" >
      <Binding.ValidationRules>
        <src:FutureDateRule />
      </Binding.ValidationRules>
    </Binding>
  </TextBox.Text>
</TextBox>
```

http://msdn.microsoft.com/en-us/library/ms752347.aspx

VALIDATION VISUAL FEEDBACK

- Validation.ErrorTemplate

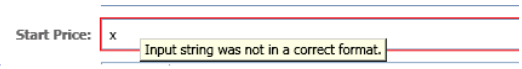



```
<ControlTemplate x:Key="validationTemplate„>
  <DockPanel>
    <TextBlock Foreground="Red" FontSize="20">
      !
    </TextBlock>
    <AdornedElementPlaceholder/>
  </DockPanel>
</ControlTemplate>
```

http://msdn.microsoft.com/en-us/library/ms752347.aspx

VALIDATION VISUAL FEEDBACK CONT'D

- Validation.HasErrors




```
<Style x:Key="textStyleTextBox" TargetType="TextBox„>
  <Setter Property="Foreground" Value="#333333" />
  <Setter Property="MaxLength" Value="40" />
  <Setter Property="Width" Value="392" />
  <Style.Triggers>
    <Trigger Property="Validation.HasErrors" Value="true„>
      <Setter Property="ToolTip”
      Value="{Binding RelativeSource=
                  {RelativeSource Self},
            Path=(Validation.Errors)[0]
                  .ErrorContent}"/>
    </Trigger>
  </Style.Triggers>
</Style>
```

http://msdn.microsoft.com/e .aspx

VALIDATION PROCESS

1. Run ValidationRule objects with ValidationStep of **RawProposedValue**
2. Call the converter (if it exists)
3. Run ValidationRule objects with ValidationStep of **ConvertedProposedValue**
4. Set the source property
5. Run ValidationRule objects with ValidationStep of **UpdatedValue**
   - default point at which DataErrorValidationRules fire
6. Run ValidationRule objects with ValidationStep of **CommittedValue**

Animations

WPF Property Animation System

- Property requirements:
  - dependency property
  - belonging to class inheriting from DependencyObject and implementing IAnimatable
  - availability of a compatible animation type
    - custom one can be created when necessary
- Animations can be applied to non-visual properties as well

Animation Types

- **<Type>Animation** (e.g. DoubleAnimation or ColorAnimation)
  - animates between a starting (**From**) and destination value (**To**), or by adding an offset value (**By**) to its starting value
- **<Type>AnimationUsingKeyFrames**
  - any number of target values and their interpolation method can be specified
- **<Type>AnimationUsingPath**
  - makes use of geometric path to produce animated values
- **<Type>AnimationBase**
  - abstract class which can be used to create custom animations

Timeline

- Base type for all animations
- Defines a segment of time
- Most important properties:
  - **Duration** (default is 1 second)
  - **AutoReverse** (default is false)
  - **RepeatBehavior** (the default is 1.0)
  - **FillBehavior**
    - {HoldEnd, Stop}
- **SpeedRatio**, **AccelerationRatio**, **DecelarationRatio**

STORYBOARD

- A type of container timeline
- Provides targeting information for the timelines it contains
- 2 ways to start:
  - XAML: **BeginStoryboard** object
    - typically within **EventTrigger**, **Trigger**, or **DataTrigger**
  - in code: Begin() method
- Controlled using code or XAML
- Available operations: pause, resume, seek, skipToFill, stop, remove

DOUBLEANIMATION

```
<Rectangle Name="MyRectangle" Width="100"
               Height="100" Fill="Blue">
  <Rectangle.Triggers>
    <EventTrigger RoutedEvent="Rectangle.Loaded">
      <BeginStoryboard>
        <Storyboard>
          <DoubleAnimation
                      Storyboard.TargetName="MyRectangle"
                      Storyboard.TargetProperty="Opacity"
                      From="1.0" To="0.0"
                      Duration="0:0:5" AutoReverse="True"
                      RepeatBehavior="Forever" />
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Rectangle.Triggers>
</Rectangle>
```

Key-Frame Animations

- Interpolation methods:
  - Linear
  - Discrete
  - Splined (Bezier curve based)
    - fixed start and end points: (0.0, 0.0), (1.0, 1.0); the other two set via the KeySpline property
  - can be mixed within a single animation
- **KeyTime**
  - specifies when that key frame ends
  - can be specified as percentage
  - special values: *Paced* and *Uniform* (default)

Key-frame Animations cont'd

```
<Storyboard>
  <DoubleAnimationUsingKeyFrames
       Storyboard.TargetName="MyAnimatedTranslateTransform"
       Storyboard.TargetProperty="X"
       Duration="0:0:10">
    <LinearDoubleKeyFrame Value="0" KeyTime="0:0:0" />
    <LinearDoubleKeyFrame Value="350" KeyTime="0:0:2" />
    <LinearDoubleKeyFrame Value="50" KeyTime="0:0:7" />
    <LinearDoubleKeyFrame Value="200" KeyTime="0:0:8" />
  </DoubleAnimationUsingKeyFrames>
</Storyboard>
```

```
<Storyboard>
  <DoubleAnimationUsingKeyFrames
       Storyboard.TargetName="ComboAnimatedTranslateTransform"
       Storyboard.TargetProperty="X"
       Duration="0:0:15"
       RepeatBehavior="Forever">
    <DiscreteDoubleKeyFrame Value="500" KeyTime="0:0:7" />
    <LinearDoubleKeyFrame Value="200" KeyTime="0:0:10" />
    <SplineDoubleKeyFrame Value="350" KeyTime="0:0:15"
       KeySpline="0.25,0.5 0.75,1" />
  </DoubleAnimationUsingKeyFrames>
</Storyboard>
```

Path Animation

- **PathGeometry**
  - generates its output based on the x, y, and angle information in the path
- 3 types:
  - **MatrixAnimationUsingPath**
    - generates Matrix values
    - one way to move an object along a path is to use a **MatrixTransform** and a **MatrixAnimationUsingPath** to transform an object along a complex path
    - **DoesRotateWithTangent**
      - forces the object to rotate along the tangent of the path
- **PointAnimationUsingPath**
  - generates Point values from the x- and y-coordinates of the PathGeometry
- **DoubleAnimationUsingPath**
  - generates Double values from its PathGeometry

Path Animation cont'd

```
<Button MinWidth="100" Content="A Button">
  <Button.RenderTransform>
    <MatrixTransform x:Name="ButtonMatrixTransform"/>
  </Button.RenderTransform>
  <Button.Triggers>
    <EventTrigger RoutedEvent="Button.Loaded">
      <BeginStoryboard>
        <Storyboard>
          <MatrixAnimationUsingPath
                Storyboard.TargetName="ButtonMatrixTransform"
                Storyboard.TargetProperty="Matrix"
                DoesRotateWithTangent="True"
                Duration="0:0:5"
                RepeatBehavior="Forever" >
            <MatrixAnimationUsingPath.PathGeometry>
              <PathGeometry
                Figures="M 10,100 C 35,0 135,0 160,
                                   100 180,190 285,200 310,100"
                PresentationOptions:Freeze="True" />
            </MatrixAnimationUsingPath.PathGeometry>
          </MatrixAnimationUsingPath>
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Button.Triggers>
</Button>
```

CRAZY BUTTON TIME!

WPF PROPERTY SYSTEM

WPF Property System

- Typically:
  - simple properties exposed as regular CLR properties

  but:
  - backed up by dependency properties
- Dependency properties:
  - may be computed based on other inputs (themes, styles, templates, data binding, animations)
  - may provide validation, default values, change notifications, value coercion based on runtime state etc.
  - may have metadata attached
  - get better support from WPF designers, support property value inheritance in the element tree

Dependency Properties

- Backed by **DependencyProperty** objects instead of private fields
  - require the wrapper property and DependencyProperty to follow the naming convention (<*PropertyName*> and <*PropertyName*>Property)
- Registered and owned by **DependencyObject** implementations

```
public static readonly DependencyProperty IsSpinningProperty =
     DependencyProperty.Register(
        "IsSpinning",   typeof(Boolean),... );

public bool IsSpinning
{
   get { return (bool)GetValue(IsSpinningProperty); }
   set { SetValue(IsSpinningProperty, value); }
}
```

Dependency Properties Metadata

- **PropertyMetadata**
  - DefaultValue
  - PropertyChangedCallback
  - …
- **FrameworkPropertyMetadata** (for FrameworkElement properties only)
  - AffectsArrange, AffectsMeasure, AffectsParentArrange, AffectsParentMeasure, AffectsRender
  - BindsTwoWayByDefault, IsDataBindingAllowed, DefaultUpdateSourceTrigger
  - Inherits

Dependency Properties Metadata Overrides

- Each class potentially holds individual metadata for all properties that are inherited from base classes, on a per-type basis

```
public class SpinnerControl : ItemsControl
{
  static SpinnerControl()
  {
    DefaultStyleKeyProperty.OverrideMetadata(
      typeof(SpinnerControl),
      new FrameworkPropertyMetadata(typeof(SpinnerControl))
      );
  }
}
```

Time to try it out

Attached Properties

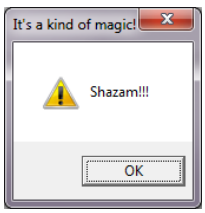
- Allow dynamically attaching properties owned by other objects
- XAML (not WPF) concept
- Implemented as dependency properties in WPF
- Typical usage:
  - parent element defines attached property whose values will be set by child elements
  - property owner will be used as child element for various parent elements
  - property is defined by a service class

```
<DockPanel>
   <CheckBox DockPanel.Dock="Top">Hello</CheckBox>
</DockPanel>
```

ATTACHED PROPERTIES CONT'D

```
public static readonly DependencyProperty IsBubbleSourceProperty =
   DependencyProperty.RegisterAttached(
           "IsBubbleSource", typeof(Boolean),
           typeof(AquariumObject),
           new FrameworkPropertyMetadata(false,
         FrameworkPropertyMetadataOptions.AffectsRender)
                              );

public static void SetIsBubbleSource(
                        UIElement element, Boolean value)
{
   element.SetValue(IsBubbleSourceProperty, value);
}

public static Boolean GetIsBubbleSource(UIElement element)
{
   return (Boolean)element.GetValue(IsBubbleSourceProperty);
}
```


TIME TO TRY IT OUT (AGAIN)



EVENTS & COMMANDS

EVENTS

```
<Image Name="FionaImage"
       Margin="5"
       Source="Images/Fiona_67x77.gif"
       MouseEnter="FionaImage_MouseEnter"
       MouseLeave="FionaImage_MouseLeave"/>
```

```
FionaImage.MouseEnter +=
              new MouseEventHandler(FionaImage_MouseEnter);
FionaImage.MouseLeave += FionaImage_MouseLeave;
```

Event Routing

- 'A routed event is a type of event that can invoke handlers on multiple listeners in an element tree, rather than just on the object that raised the event.'
- Routing strategies:
  - Direct
    - event originates in one element and is not passed to any other (as in Windows Forms)
    - e.g.: MouseEnter, MouseLeave
  - Bubbling
    - event travels up the hierarchy
    - most input or state change events e.g. MouseDown
  - Tunneling
    - event travels down the tree towards its source
    - preview events for most input event, e.g.: PreviewKeyDown
      - preview events can be suppressed or replaced by custom events by controls (e.g.: Click)
      - if preview event is marked as handled, normal event will not fire

RoutedEventArgs

- Base class for event arguments of all WPF events
- Properties:
  - **Source**
    - object that raised the event
    - may be different from sender – i.e. the object that is currently handling the event
  - **OriginalSource**
    - object that originally raised the event
    - read-only
    - may be different from Source if source adjustment has been performed
      - e.g. ListBox will report ListItem as source, OriginalSource will point to the element inside ListItem

RoutedEvent

- implementation of routed events conceptually similar to dependency properties
  - typically consists of 2 element:
    - static readonly RoutedEvent field
    - regular CLR event wrapper

```
public static readonly RoutedEvent TapEvent =
   EventManager.RegisterRoutedEvent("Tap",
   RoutingStrategy.Bubble, typeof(RoutedEventHandler),
   typeof(MyButtonSimple));

// Provide CLR accessors for the event
public event RoutedEventHandler Tap
{
   add { AddHandler(TapEvent, value); }
   remove { RemoveHandler(TapEvent, value); }
}
```


RoutedEvent cont'd

- Just like in case of dependency properties, it is possible to:
  - reference events bypassing the helper wrappers
  - create attached events

```
void MakeButton()
{
   Button b2 = new Button();
   b2.AddHandler(Button.ClickEvent,
               new RoutedEventHandler(Onb2Click));
}

void Onb2Click(object sender, RoutedEventArgs e)
{
   //logic to handle the Click event
}
```

TIME TO TRY IT OUT (YET AGAIN)

KEYBOARD INPUT

- Common keyboard events:
  - PreviewKeyDown, KeyDown, PreviewTextInput, TextInput, PreviewKeyUp, KeyUp
- Focus
  - **Focusable**
  - **TabIndex**
- **KeyboardEventArgs**
  - base class for all keyboard events args
  - not used directly by any event
- **Keyboard**
  - static class
  - current state of the keyboard
  - attached keyboard events definitions

Mouse Input

- Common mouse events (available for any UIElement):
  - MouseEnter, MouseLeave
    - direct routing strategy
  - MouseLeftButtonDown, MouseLeftButtonUp, MouseRightButtonDown, MouseRightButtonUp, and their corresponding Preview… tunneling events
- **Control** class events:
  - MouseDoubleClick, PreviewMouseDoubleClick
- **Mouse**
  - static class
  - allows retrieval of current mouse state
  - defines the mouse attached events
- **DragDrop**
  - static class facilitating drag & drop operations

Commands

- Encapsulate common tasks
  - usually available in different contexts and/or invoked by different UI elements
- **ICommand**
  - Execute()
  - CanExecute()

RoutedCommand

- 4 main concepts:
  - command
    - action to be executed
  - command source
    - object which invokes the command
  - command target
    - object that command is executed on
  - command binding
    - maps the command logic to the command
- Common routed commands available as properties of:
  - **MediaCommands**, **ApplicationCommands**, **NavigationCommands**, **ComponentCommands**, and **EditingCommands**

Command Sources and Targets

- **ICommandSource**
  - Command
  - CommandTarget
    - applicable only to RoutedCommand
  - CommandParameter
    - passed to the ICommand.Execute and the ICommand.CanExecute() methods
  - implemented by:
    - ButtonBase, MenuItem, Hyperlink, InputBinding etc.
- Command target
  - can be set explicitly by the command source
  - if not defined, the element with keyboard focus will be used

COMMAND SOURCES CONT'D

```
<StackPanel>
  <Menu>
    <MenuItem Command="ApplicationCommands.Paste" />
  </Menu>
  <TextBox />
</StackPanel>
```

```
<Window.InputBindings>
   <KeyBinding Key="B" Modifiers="Control"
                       Command="ApplicationCommands.Open" />
</Window.InputBindings>
```


COMMAND BINDINGS

- **CommandBinding**
  - Command
  - PreviewExecuted, Executed
  - PreviewCanExecute, Execute
- Associates the command with event handlers
  - effectively, transforms the command into a routed event
  - binding scope is defined by the object to which the command binding is attached
    - often it is useful to attach it to an ancestor of the command target

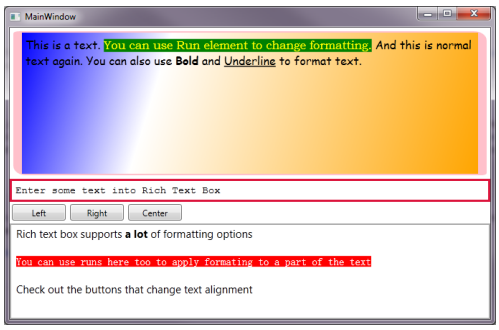
COMMAND BINDINGS CONT'D

```
<Window.CommandBindings>
  <CommandBinding
            Command="ApplicationCommands.Open“
            Executed="OpenCmdExecuted”
            CanExecute="OpenCmdCanExecute"/>
</Window.CommandBindings>
```

```
void OpenCmdCanExecute(object sender, CanExecuteRoutedEventArgs e)
{
    e.CanExecute = true;
}
```

```
void OpenCmdExecuted(object target, ExecutedRoutedEventArgs e)
{
    string command, targetobj;
    command = ((RoutedCommand)e.Command).Name;
    targetobj = ((FrameworkElement)target).Name;
    MessageBox.Show("The " + command +
                " command has been invoked on target object " + targetobj);
}
```

DEMO TIME: RICHTEXTBOX COMMANDS



MARKUP EXTENSIONS

MARKUP EXTENSIONS

- Extension point for XAML
- Allow introduction into the elements tree values which are not newly created objects (as typical elements do)
- Provide values for attributes
- 2 kinds of syntax
  - traditional property element syntax - visually no different than element value

```
<x:Array Type="sys:String“ xmlns:sys="clr-namespace:System;assembly=mscorlib">
   <sys:String>Hello</sys:String>
   <sys:String>World</sys:String>
</x:Array>
```

  - attribute value - distinguished by presence of curly braces

```
<TextBox Style="{StaticResource textStyleTextBox}" >
```

Markup Extensions

- XAML-defined markup extensions:
  - x:Type
  - x:Static
  - x:Null
  - x:Array

Markup Extensions

- WPF-specific built-in markup extensions:
  - StaticResource
  - DynamicResource
  - Binding
  - RelativeSource
  - TemplateBinding
  - ColorConvertedBitmap
  - ComponentResourceKey
  - ThemeDictionary

Custom Markup Extensions

```
[MarkupExtensionReturnType(typeof(Style))]
public class CompoundStyleExtension : MarkupExtension
{
    public Style Base { get; set; }
    public Style Extra1 { get; set; }
    public Style Extra2 { get; set; }


    public override object ProvideValue(IServiceProvider
                                        serviceProvider)
    {
        Style baseStyle = new Style(Base.TargetType, Base);
        MergeStyles(baseStyle, this.Extra1);
        MergeStyles(baseStyle, this.Extra2);
        return baseStyle;
    }

…
```


Custom Markup Extensions cont'd

```
…

    private void MergeStyles(Style baseStyle, Style extraStyle)
    {
        if (extraStyle != null)
        {
            if (extraStyle.BasedOn != null)
            {
                MergeStyles(baseStyle, extraStyle.BasedOn);
            }
            foreach (var setter in extraStyle.Setters)
                baseStyle.Setters.Add(setter);
            foreach (var trigger in extraStyle.Triggers)
                baseStyle.Triggers.Add(trigger);
        }
    }
}
```

Putting it all together, AKA demo time: DataGrid

PUTTING IT ALL TOGETHER

DataGrid Control

- Control for displaying and manipulating tabular data
- Similar to **`GridView`** in ASP.NET or **`DataGridView`** in WinForms
- Not included in the initial release of WPF
  - Microsoft heavily criticized
- CTP released in August 2008 (at the same time as .NET 3.5 SP1)

Defining Columns

- **`ItemsSource`**
  - any IEnumerable as data source for binding
- Columns generation:
  - automatic
  - manual
    - DataGridTextColumn
    - DataGridCheckBoxColumn
    - DataGridComboBoxColumn
    - DataGridHyperlinkColumn
    - DataGridTemplateColumn

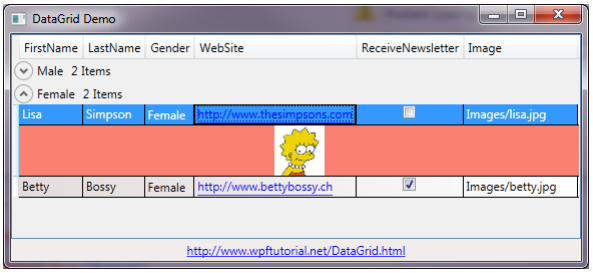
Advanced Capabilities

- Grouping data
  - **`CollectionView.GroupDescription`**
  - **`DataGrid.GroupStyle`**
- Row details
  - **`DataGrid.RowDetailsTemplate`**
  - alternatively:
    - use **`DataTemplateSelector`**
    - set **`DataGrid.RowDetailsTemplateSelector`**

ADVANCED CAPABILITIES CONT'D

- Selection
  - **`SelectionUnit`**
    - individual cell / whole row
  - **`SelectionMode`**
    - single / multiple
- **`FrozenColumnsCount`**
- Editing data:
  - CanUserAddRows, CanUserDeleteRows, CanUserReorderColumns, CanUserResizeColumns, CanUserSortColumns
  - Built-in commands
    - Begin edit (F2), Cancel edit (Esc), Commit edit(Enter), Delete(Del)

HOW ABOUT SOME PRACTICE? (FINALLY)



http://www.wpftutorial.net/DataGrid.html

GRAPHICS

GRAPHICS IN WPF

- Resolution-independent and device-independent
  - device independent pixel: 1/96th of an inch
    - automatically scaled to match dpi settings
- Double-precision floating-point number coordinates precision
- Support for advanced graphic rendering scenarios
  - retained rendering system
  - animations support
  - hit-testing support
  - full alpha-compositing support
- Hardware acceleration

The Visual Layer

- The most light-weight but least interactive way to render 2D Graphics in WPF
- Can only be accessed through code (not from XAML)
- Used to display massive amounts of graphical data or static images over very large surfaces
- **`System.Windows.Media.Visual`** - abstract class
  - minimum set of graphics services (rendering, hit-testing, transformations)
  - no support for non-visual services (e.g. responding to input events)

DrawingVisual

- **`DrawingVisual`**
  - inherits from Visual
  - used to display shapes, images and text
  - no support for layout or event handling
    - thus: requires a container
    - common scenario: creating a custom layout manager that displays its content using the visual layer

DrawingContext

- Offers drawing operations API
- Obtained from a DrawingVisual object

```
DrawingVisual dv = new DrawingVisual();
using (DrawingContext dc = dv.RenderOpen())
{
// Create a rectangle and draw it in the DrawingContext.
Rect rect =
  new Rect(new System.Windows.Point(160, 100),
         new System.Windows.Size(320, 80));
dc.DrawRectangle(
         System.Windows.Media.Brushes.LightBlue,
         (System.Windows.Media.Pen)null, rect);
}
```


DrawingVisual cont'd

```
public class MyVisualHost : FrameworkElement
{
  // Create a collection of child visual objects.
  private VisualCollection _children;

  public MyVisualHost()
  {
    _children = new VisualCollection(this);
    _children.Add(CreateDrawingVisualRectangle());
    _children.Add(CreateDrawingVisualText());
    _children.Add(CreateDrawingVisualEllipses());

    // Add the event handler for MouseLeftButtonUp.
    this.MouseLeftButtonUp +=
         new MouseButtonEventHandler(
                MyVisualHost_MouseLeftButtonUp);
  }
```

http://msdn.microsoft.com/en-us/library/ms742254.aspx

DRAWINGVISUAL CONT'D

```
// Create a DrawingVisual that contains a rectangle.
private DrawingVisual CreateDrawingVisualRectangle()
{
   DrawingVisual drawingVisual = new DrawingVisual();

   // Retrieve the DrawingContext in order
   // to create new drawing content.
   DrawingContext drawingContext = drawingVisual.RenderOpen();

   // Create a rectangle and draw it in the DrawingContext.
   Rect rect = new Rect(new System.Windows.Point(160, 100),
                        new System.Windows.Size(320, 80));
   drawingContext.DrawRectangle(
                  System.Windows.Media.Brushes.LightBlue,
                  (System.Windows.Media.Pen)null, rect);

   // Persist the drawing content.
   drawingContext.Close();
   return drawingVisual;
}
```

http://msdn.microsoft.com/en-us/library/ms742254.aspx

DRAWINGVISUAL CONT'D

```
// Provide a required override for
// the VisualChildrenCount property.
protected override int VisualChildrenCount
{
  get { return _children.Count; }
}

// Provide a required override for the GetVisualChild method.
protected override Visual GetVisualChild(int index)
{
  if (index < 0 || index >= _children.Count)
  {
    throw new ArgumentOutOfRangeException();
  }
  return _children[index];
}
```

http://msdn.microsoft.com/en-us/library/ms742254.aspx

Hit Testing

```
void MyVisualHost_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{
   // Retreive the coordinates of the mouse button event.
   System.Windows.Point pt = e.GetPosition((UIElement)sender);
   // Initiate the hit test by setting up a hit test result callback method.
   VisualTreeHelper.HitTest(this, null, new HitTestResultCallback(myCallback),
                  new PointHitTestParameters(pt));
}

// If a child visual object is hit, toggle its opacity to visually indicate a hit.
public HitTestResultBehavior myCallback(HitTestResult result)
{
   if (result.VisualHit.GetType() == typeof(DrawingVisual))
   {
      if (((DrawingVisual)result.VisualHit).Opacity == 1.0)
      {
         ((DrawingVisual)result.VisualHit).Opacity = 0.4;
      }
      else
      {
         ((DrawingVisual)result.VisualHit).Opacity = 1.0;
      }
   }
   // Stop the hit test enumeration of objects in the visual tree.
   return HitTestResultBehavior.Stop;
}
```

http://msdn.microsoft.com/en-us/library/ms742254.aspx

Drawing

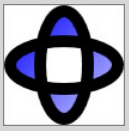
- **Drawing**
  - abstract class representing 2D drawing
  - Visual
    - thus: light-weight
      - no layout, input & focus support
  - base class for:
    - **DrawingGroup**
    - **GeometryDrawing**
    - **GlyphRunDrawing**
    - **ImageDrawing**
    - **VideoDrawing**

DrawingGroup

- **DrawingGroup**
  - a **Drawing**
  - combines multiple drawings into a single composite drawing
    - may also contain other DrawingGroups
  - offers additional possibilities:
    1. **OpacityMask**
    2. **Opacity**
    3. **BitmapEffect**
    4. **ClipGeometry**
    5. **GuidelineSet**
    6. **Transform**

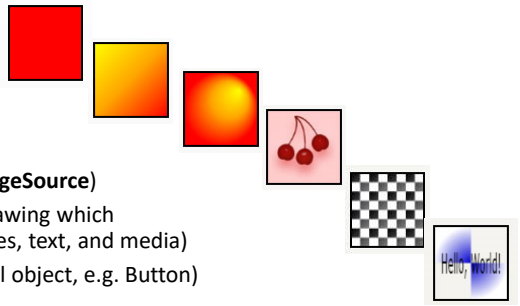
Drawing in practice

```
<Image Stretch="None" HorizontalAlignment="Left">
 <Image.Source>
  <DrawingImage>
   <DrawingImage.Drawing>
    <GeometryDrawing>
     <GeometryDrawing.Geometry>
      <!-- Create a composite shape. -->
      <GeometryGroup>
       <EllipseGeometry Center="50,50" RadiusX="45" RadiusY="20" />
       <EllipseGeometry Center="50,50" RadiusX="20" RadiusY="45" />
      </GeometryGroup>
     </GeometryDrawing.Geometry>
     <GeometryDrawing.Brush>
      <LinearGradientBrush> ... </LinearGradientBrush>
     </GeometryDrawing.Brush>
     <GeometryDrawing.Pen> ... </GeometryDrawing.Pen>
    </GeometryDrawing>
   </DrawingImage.Drawing>
  </DrawingImage>
 </Image.Source>
</Image>
```



http://msdn.microsoft.com/en-us/library/system.windows.media.geometrydrawing.aspx

BRUSHES

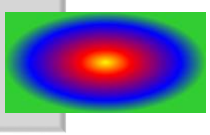


- Available brushes:
  - **SolidColorBrush**
  - **LinearGradientBrush**
  - **RadialGradientBrush**
  - **ImageBrush** (uses an **ImageSource**)
  - **DrawingBrush** (uses a Drawing which can contain shapes, images, text, and media)
  - **VisualBrush** (uses a Visual object, e.g. Button)
- Common brush features:
  - Opacity
  - Transform, RelativeTransform
  - can be animated
  - can be frozen (i.e. cannot be modified)

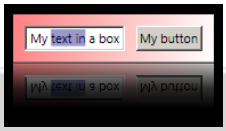
BRUSHES CONT'D

```
<Rectangle Width="200" Height="100">
  <Rectangle.Fill>
    <LinearGradientBrush StartPoint="0,0"
EndPoint="1,1">
      <GradientStop Color="Yellow" Offset="0.0" />
      <GradientStop Color="Red" Offset="0.25" />
      <GradientStop Color="Blue" Offset="0.75" />
      <GradientStop Color="LimeGreen" Offset="1.0" />
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

```
<Rectangle Width="200" Height="100">
  <Rectangle.Fill>
    <RadialGradientBrush GradientOrigin="0.5,0.5"
        Center="0.5,0.5" RadiusX="0.5" RadiusY="0.5">
      <GradientStop Color="Yellow" Offset="0" />
      <GradientStop Color="Red" Offset="0.25" />
      <GradientStop Color="Blue" Offset="0.75" />
      <GradientStop Color="LimeGreen" Offset="1" />
    </RadialGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```



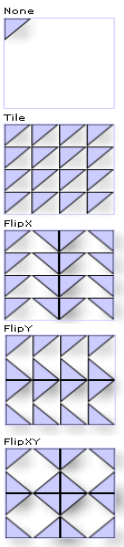
BRUSHES CONT'D



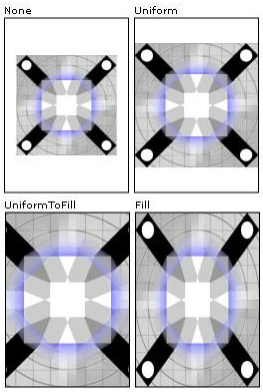

```
<StackPanel Canvas.Top="300" Canvas.Left="50">
  <Border Name="ReflectedVisual">[...]</Border>

  <Rectangle Height="1" Fill="Gray" HorizontalAlignment="Stretch" />
  <Rectangle  Height="{Binding Path=ActualHeight, ElementName=ReflectedVisual}"
              Width="{Binding Path=ActualWidth, ElementName=ReflectedVisual}">
    <Rectangle.Fill>
      <VisualBrush Opacity="0.75" Stretch="None"
                   Visual="{Binding ElementName=ReflectedVisual}">
        <VisualBrush.RelativeTransform>
          <TransformGroup>
            <ScaleTransform ScaleX="1" ScaleY="-1"/> <TranslateTransform Y="1"/>
          </TransformGroup>
        </VisualBrush.RelativeTransform>
      </VisualBrush>
    </Rectangle.Fill>
    <Rectangle.OpacityMask>
      <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
        <GradientStop Color="#FF000000" Offset="0.0" />
        <GradientStop Color="#44000000" Offset="0.5" />
        <GradientStop Color="#00000000" Offset="0.9" />
      </LinearGradientBrush>
    </Rectangle.OpacityMask>
    <Rectangle.BitmapEffect>
        <BlurBitmapEffect Radius="1.5" />
    </Rectangle.BitmapEffect>
  </Rectangle>
</StackPanel>
```

TILEBRUSH

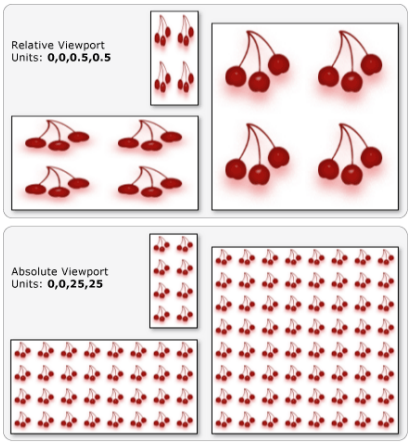
TileMode



Stretch

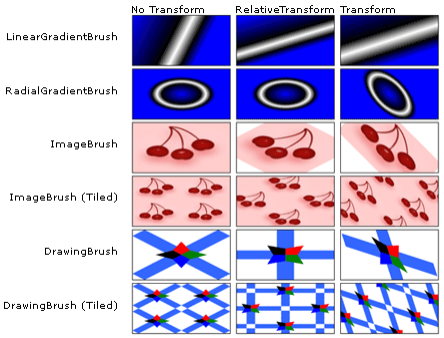


Viewport, ViewportUnits



Brush Transformations

- Brush processing algorithm:
  1. Process the brush's contents
  2. Project the brush's output onto the 1 x 1 transformation rectangle
  3. Apply the brush's **RelativeTransform**, if it has one
  4. Project the transformed output onto the area to paint
  5. Apply the brush's **Transform**, if it has one



Shapes

- **Shape**
  - base class for all available shape objects
    - **Ellipse**, **Line**, **Path**, **Polygon**, **Polyline**, and **Rectangle**
  - **Stroke, StrokeThickness**
  - **Fill, Transform, Stretch**

```
PointCollection myPointCollection =
    new PointCollection();
myPointCollection.Add(new Point(0, 0));
myPointCollection.Add(new Point(0, 1));
myPointCollection.Add(new Point(1, 1));


Polygon myPolygon = new Polygon();
myPolygon.Points = myPointCollection;
myPolygon.Fill = Brushes.Blue;
myPolygon.Width = 100;
myPolygon.Height = 100;
myPolygon.Stretch = Stretch.Fill;
myPolygon.Stroke = Brushes.Black;
myPolygon.StrokeThickness = 2;
```

```
<Polygon
    Points="0,0 0,1 1,1"
    Fill="Blue"
    Width="100"
    Height="100"
    Stretch="Fill"
    Stroke="Black"
    StrokeThickness="2" />
```

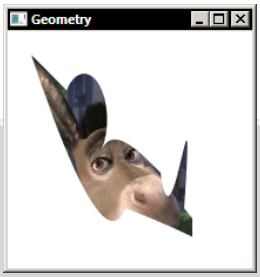
GEOMETRIES

- More versatile than **Shape** objects:
  - allow rendering more complex shapes
    - open or closed
    - curved
  - allow defining regions for clipping
  - allow defining regions for hit testing
- **Path**
  - makes use of a **Geometry** to describe its contents
  - can be used to render a **Geometry** (by setting the **Data**, **Fill**, and **Stroke** properties)

GEOMETRIES CONT'D

- Simple geometry types:
  - **LineGeometry**, **RectangleGeometry**, **EllipseGeometry**
- **PathGeometry**
  - contains a collection of **PathFigure**s – each consisting of segments:
    - LineSegment, PolyLineSegment, ArcSegment, BezierSegment, PolyBezierSegment, QuadraticBezierSegment, PolyQuadraticBezierSegment
- **StreamGeometry**
  - defines a complex geometric shape that may contain curves, arcs, and lines
  - does not support data binding, animation, or modification (unlike the PathGeometry) – light-weight alternative to PathGeometry
- **CombinedGeometry**
- **GeometryGroup**

GEOMETRIES CONT'D




```
<Image Source="/Images/Donkey.gif">
  <Image.Clip>

    <PathGeometry>
      <PathGeometry.Figures>
        <PathFigure StartPoint="20,20">
          <PathFigure.Segments>
            <LineSegment Point="50,50"/>
            <ArcSegment Point="80,100" Size="10,20"
                        SweepDirection="Clockwise"/>
            <BezierSegment Point1="90,80"
                           Point2="110,100" Point3="130,120"/>
            <PolyBezierSegment Points="140,200,160,0,150,150,
                                       190,200,180,180,100,150"/>
            <QuadraticBezierSegment Point1="80,200" Point2="20,20"/>
          </PathFigure.Segments>
        </PathFigure>
      </PathGeometry.Figures>
    </PathGeometry>

  </Image.Clip>
</Image>
```


IMAGING

- Codecs for BMP, JPEG, PNG, TIFF, Windows Media Photo, GIF, ICON
  - all except ICON support both decoding and encoding
- **BitmapSource**
  - **BitmapImage**
  - **CroppedBitmap**, **FormatConvertedBitmap**, **TransformedBitmap**
- **BitmapFrame**
- **Image** control

IMAGE CONTROL

```
<Image Width="200" Source="Water Lilies.jpg"/>


<Image Width="200">
  <Image.Source>
    <BitmapImage DecodePixelWidth="200"
                 UriSource="Water Lilies.jpg" />
  </Image.Source>
</Image>
```

http://msdn.microsoft.com/en-us/library/ms748873.aspx

IMAGE CONTROL

```
// Create Image Element
Image myImage = new Image();
myImage.Width = 200;

// Create source
BitmapImage myBitmapImage = new BitmapImage();

// BitmapImage.UriSource must be in a BeginInit/EndInit
block
myBitmapImage.BeginInit();
myBitmapImage.UriSource = new Uri(@"Water Lilies.jpg");
myBitmapImage.DecodePixelWidth = 200;
myBitmapImage.EndInit();

//set image source
myImage.Source = myBitmapImage;
```

http://msdn.microsoft.com/en-us/library/ms748873.aspx

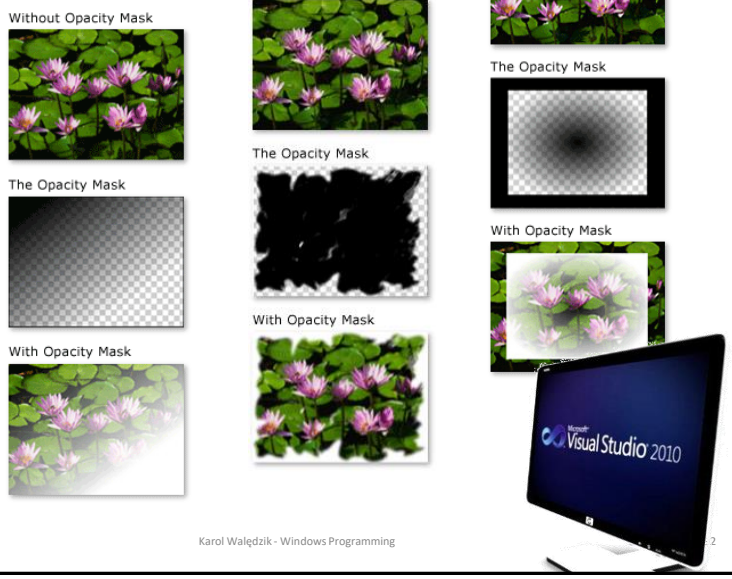
IMAGE CONTROL

```
<Image Width="150" Margin="5" Grid.Column="0" Grid.Row="1„>
  <Image.Source>
    <TransformedBitmap Source=„/Images/watermelon.jpg" >
      <TransformedBitmap.Transform>
        <RotateTransform Angle="90"/>
      </TransformedBitmap.Transform>
    </TransformedBitmap>
  </Image.Source>
</Image>
```

```
<!-- Grayscale XAML Image -->
<Image Width="200" Grid.Column="0" Grid.Row="1„>
    <Image.Source>
        <FormatConvertedBitmap Source="/sampleImages/rocks.jpg”
                                DestinationFormat="Gray4" />
    </Image.Source>
</Image>
```

http://msdn.microsoft.com/en-us/library/ms748873.aspx

OPACITY MASK



3D Graphics

- 2D vs 3D coordinates system

Figure 1
2D Coordinate System

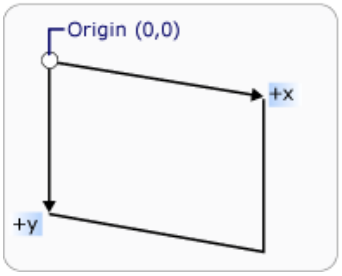
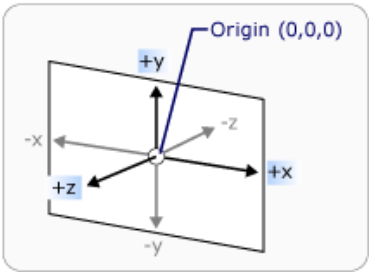


Figure 2
3D Coordinate System



- **`Viewport3D`** – container for displaying 3D graphics (surface to which the scene is projected)

Cameras

- Specify the point from which the scene is being observed
  - **`Position`** – location of the camera (Point3D)
  - **`Direction`** in which the camera is looking (Vector3D)
  - **`FieldOfView`** – horizontal view angle (in degrees)
- **`ProjectionCamera`** – specifies how the scene is being projected to 2D surface (defines the perspective)
  - **`OrtographicCamera`** and **`PerspectiveCamera`**

Orthographic Projection

Perspective Projection

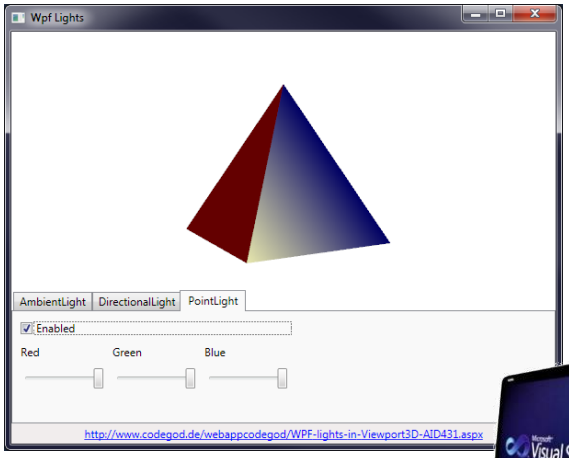
LIGHTS

- At least one light must be included in the scene to make it visible
- **`AmbientLight`** – illuminates all objects uniformly
- **`DirectionalLight`** – iluminates like a distant source of light; has direction, but no location
- **`PointLight`** – illuminates like a nearby source of light; has specified location, but no direction
- **`SpotLight`** – inherits from PointLight, but also has a direction; illuminates a cone-shaped area

3D OBJECTS

- **`Model3D`** – abstract class representing a 3D object
- **`GeometryModel3D`** – derived from Model3D
  - **`Geometry`** property – **`MeshGeometry3D`** object
    - mesh composed of triangles
    - **`Positions`** – collection of all triangle vertices that make up the mesh
    - **`TriangleIndices`** – defines the order in which points specified in Positions property construct triangles
- **`Material`** –characteristics of the object's surface
  - **`DiffuseMaterial`** – matt surface
  - **`SpecularMaterial`** – reflective surface
  - **`EmissiveMaterial`** – a glowing object

TIME TO TURN THE LIGHTS OFF?



http://www.codegod.de/webappcodegod/WPF-lights-in-Viewport3D-AID431.aspx

VIEWPORT2DVISUAL3D

- Enables placing interactive 2D content on a 3D object