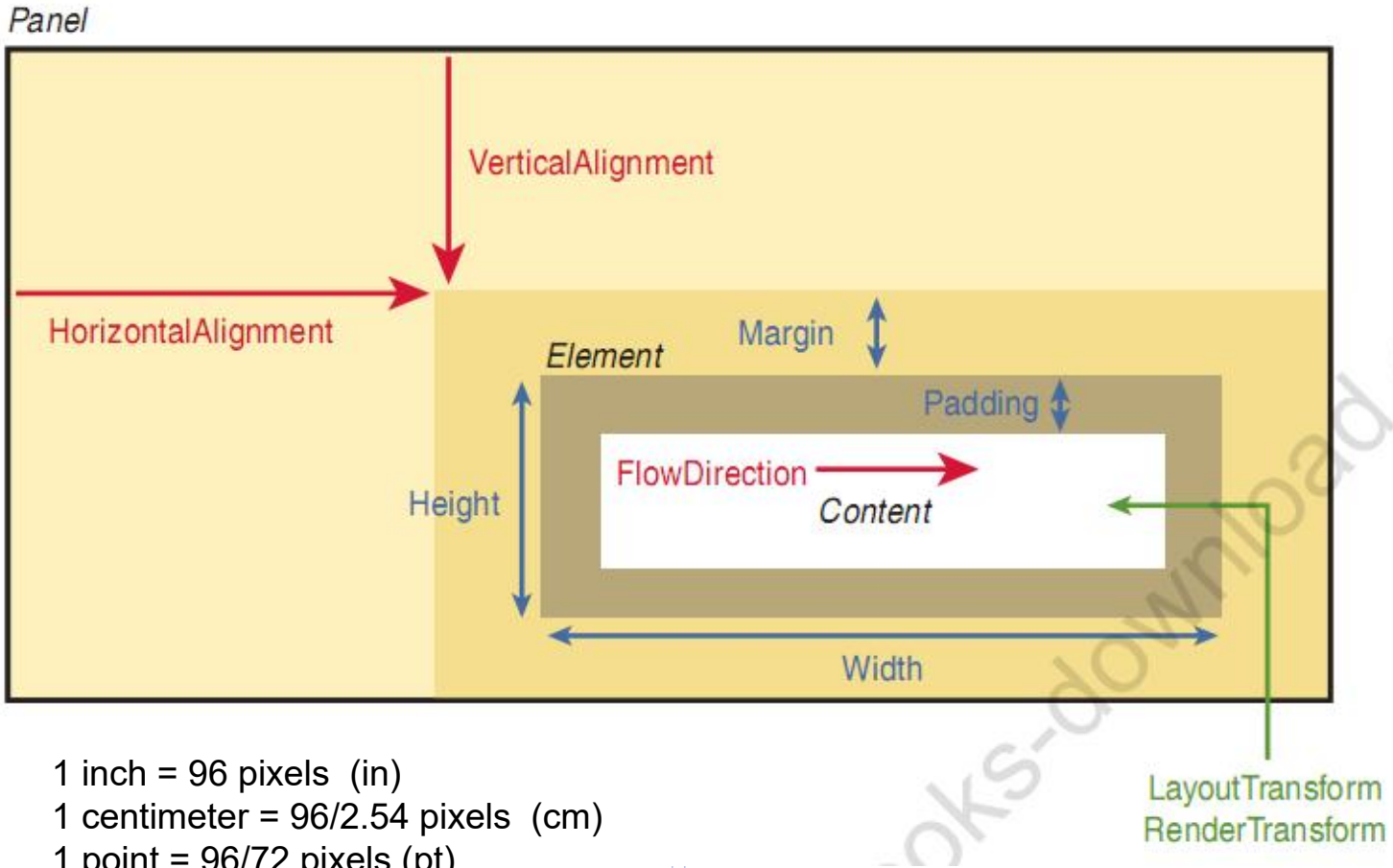
Windows Presentation Foundation

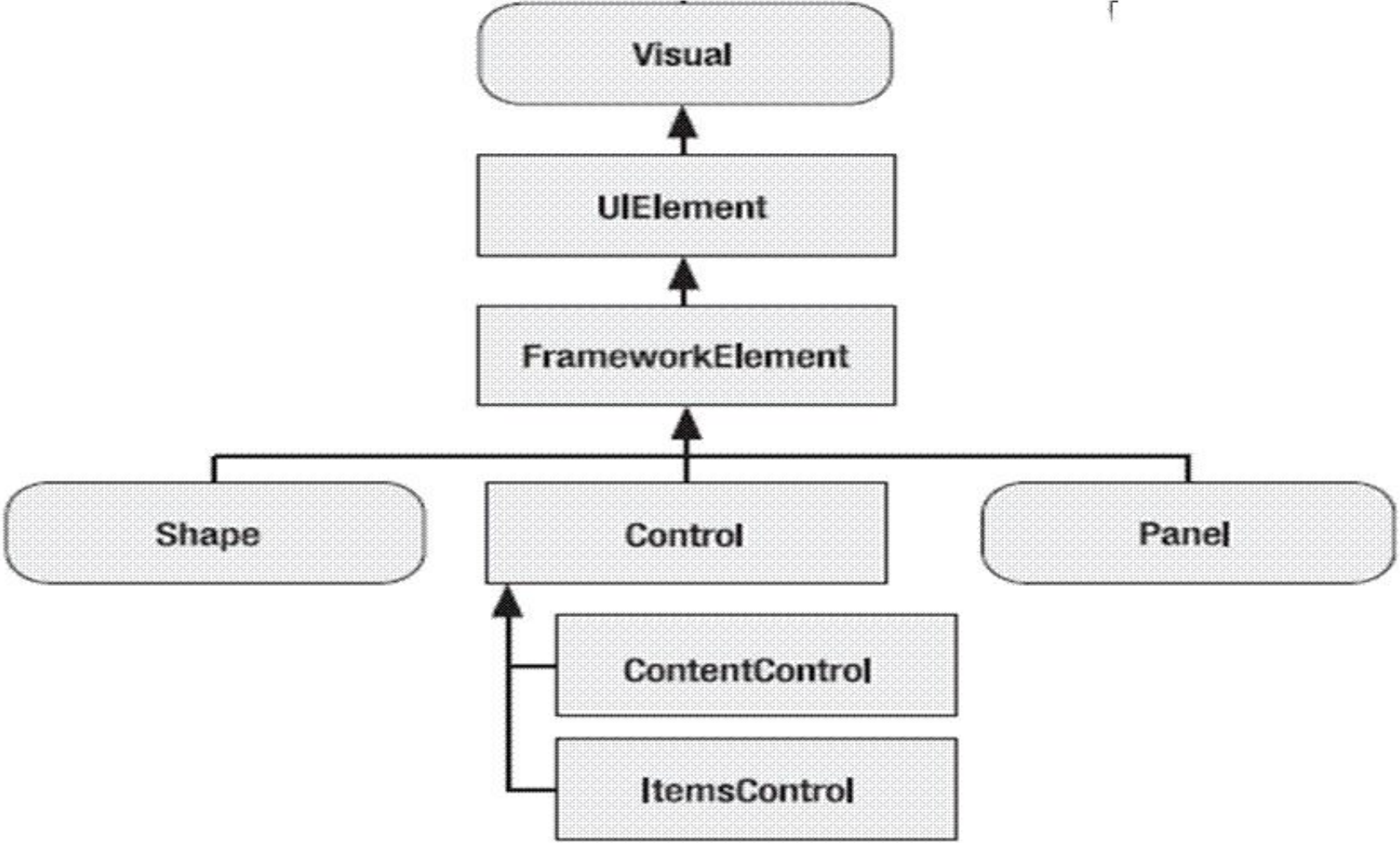
WPF

Agenda

- Brushes
- Resources
- Triggers
- Brushes
- Binding



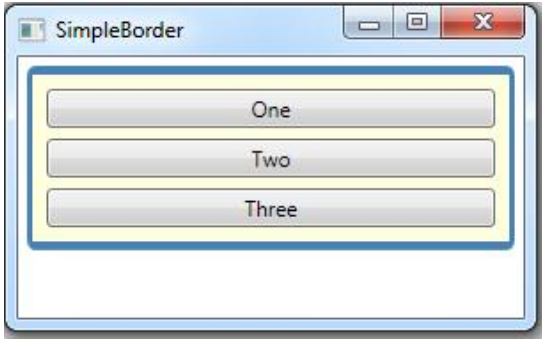
1 inch = 96 pixels (in)
1 centimeter = 96/2.54 pixels (cm)
1 point = 96/72 pixels (pt)


Visual
UIElement
FrameworkElement
Shape
Control
Panel
ContentControl
ItemsControl

Decorators

- The Border

```
<Border Margin="5" Padding="5" Background="LightYellow"
  BorderBrush="SteelBlue“
  BorderThickness="3,5,3,5"
  CornerRadius="3“ >
  <StackPanel>
        <Button Margin="3">One</Button>
        <Button Margin="3">Two</Button>
        <Button Margin="3">Three</Button>
  </StackPanel>
</Border>
```



Decorators (Con.)

- TheViewbox

  The basic principle behind the Viewbox any content you place inside the Viewbox is scaled up or down to fit the bounds of the Viewbox

Button

- When IsCancel is true

  This button is designated as the cancel button for a window. You press the Escape key while positioned anywhere on the current window, this button is triggered

- When IsDefault is true

  This button is designated as the default button(accept button)

- However, there should be only a single cancel button and a single default button in a window

ToggleButton

- ToggleButton
  - A button that has two states (pushed or unpushed). When you click a ToggleButton, it stays in its pushed state until you click it again to release it.
  - The ToggleButton is genuinely useful inside a ToolBar
  - Class derived from ButtonBase
  - RadioButton and Checkbox drived from ToggleButton Class

ToolTip

- The ToolTip property is defined in the FrameworkElement class, so it's available on anything you'll place in a WPF window

```
<Button ToolTip="This is my tooltip">
        I have a tooltip
</Button>
```

Text Controls

- WPF includes three text-entry controls:
  - TextBox
  - RichTextBox
  - PasswordBox
- The PasswordBox derives directly from Control.
- The TextBox and RichTextBox controls go through another level and derive from TextBoxBas

Text Controls & PasswordBox

- The **PasswordBox** looks like a TextBox, but it displays a string of circle symbols to mask the characters it shows
- You can choose a different mask character by setting the **PasswordChar** property
- **PasswordBox** does not support the clipboard, so you can’t copy the text inside
- It provides a **MaxLength** property

Brushes

- Brushes fill an area:
  - whether it's the background, foreground, or border of an element
  - or the fill or stroke of a shape
- The simplest type of brush is SolidColorBrush, which paints a solid, continuous color
- Brushes support partial transparency (Opacity property)
- SystemBrushes class provides access to brushes that use the colors defined in the Windows system preferences for the current computer.

- LinearGradientBrush

  Paints an area using a gradient fill, a gradually shaded fill that changes from one color to another

- ImageBrush (Viewbox- Viewport)

  Paints an area using an image that can be stretched, scaled, or tiled

- RadialGradientBrush

  Paints an area using a radial gradient fill, which is similar to a linear gradient except it radiates out in a circular pattern starting from a center point

- VisualBrush

LinearGradientBrush

- GradientStops collection:
  - To create this gradient, you need to add one GradientStop for each color
  - each color in your gradient using an Offset value from 0 to 1
- StartPoint Property: allow you to choose the point where the first color begins to change
- EndPoint Property: the point where the color change ends with the final color
- SpreadMethod property : (Pad – Reflect – Repeat)

RadialGradientBrush

- GradientOrigin Property:

  To identify the point where the first color in the gradient starts, property. By default, it’s (0.5, 0.5)

- RadiusX , RadiusY :

  determine the size of the limiting circle, and by default, they’re both set to **0.5**

- Used also RadialGradientBrush.GradientStops

- VisualBrush is an unusual brush that allows you to take the visual content of an element and use it to fill any surface.
- You could copy the appearance of a button in a window to a region somewhere else in that same window.
- However, the button copy won't be clickable or interactive in any way. It's simply a copy of how your element looks.

```
<VisualBrush Visual="{Binding
ElementName=cmd}“>
</VisualBrush>
```

Resources

W
P
F

Resources

- **Resources have a number of important benefits:**
  - **Efficiency:**

    Resources let you define an object once and use it in several places in your markup
  - **Maintainability:**

    Resources let you take low-level formatting details (such as font sizes) and move them to a central place where they're easy to change
  - **Adaptability**:

    Once certain information is separated from the rest of your application and placed in a resource section, it becomes possible to modify it dynamically.

- Every element includes a Resources property, which stores a dictionary collection of resources (It's an instance of the ResourceDictionary class.)

- The resources collection can hold any type of object, indexed by string (Key)

- The most common way to define resources is at the window-level, every element has access to the resources in its own resource collection and the resources in all of its parents' resource collections

```
<Window.Resources>
    <FontFamily x:Key="ButtonFontFamily">
        Tahoma
    </FontFamily>
    <sys:Double x:Key="ButtonFontSize">
        18
    </sys:Double>
    <FontWeight x:Key="ButtonFontWeight">
        Bold
    </FontWeight>

</Window.Resources>
```

- To use a resource in your XAML markup

```
<Button
    FontFamily= "{StaticResource ButtonFontFamily}“ >
</Button>
```

```
<Button
    FontFamily="{DynamicResource
ButtonFontFamily}“ ></Button>
```

```
<Button>
   <Button.FontFamily>
      <StaticResource    [DynamicResource]
            ResourceKey="ButtonFontFamily">
      </StaticResource>
   </Button.FontFamily>
</Button>
```

Many Layers of Styles

```
<Window.Resources>
<Style x:Key="BigFontButtonStyle">
……
</Style>
<Style x:Key="EmphasizedBigFontButtonStyle"
BasedOn="{StaticResource BigFontButtonStyle}">
<Setter Property="Control.Foreground" Value="White" />
<Setter Property="Control.Background" Value="DarkBlue"
    />
</Style>
</Window.Resources>
```

Triggers

- You can react when a property is changed and adjust a style automatically
- Triggers are linked to styles through the Style.Triggers collection.
- Every style can have an unlimited number of triggers, and each trigger is an instance of a class that derives from System.Windows.TriggerBase

Classes That Derive from TriggerBase

- Trigger

  This is the simplest form of trigger. It watches for a change in a property and then uses a setter to change the style.

- MultiTrigger

  This is similar to trigger but combines multiple conditions. All the conditions must be met before the trigger springs into action.

- EventTrigger

  This is the most sophisticated trigger. It applies an animation when an event occurs.

Binding

Data Binding

- Binding to elements

  Using Binding.ElementName property

- Binding to Objects That Aren’t Elements

It’s more common to create binding expressions that draw their data from a non-visual object.

Using Source property

- Binding to Database