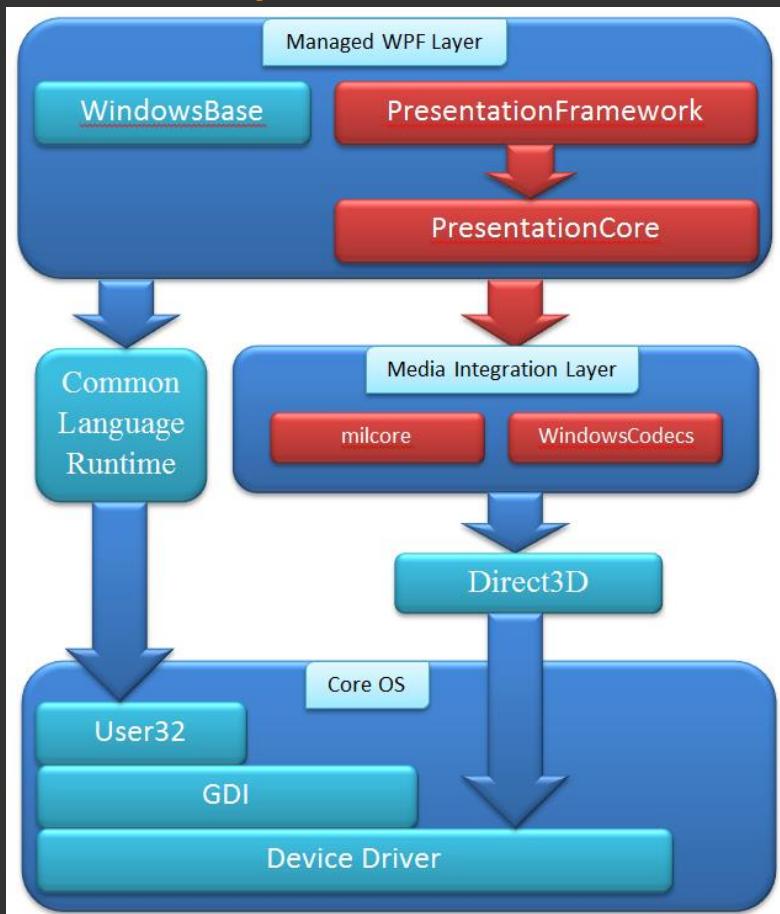


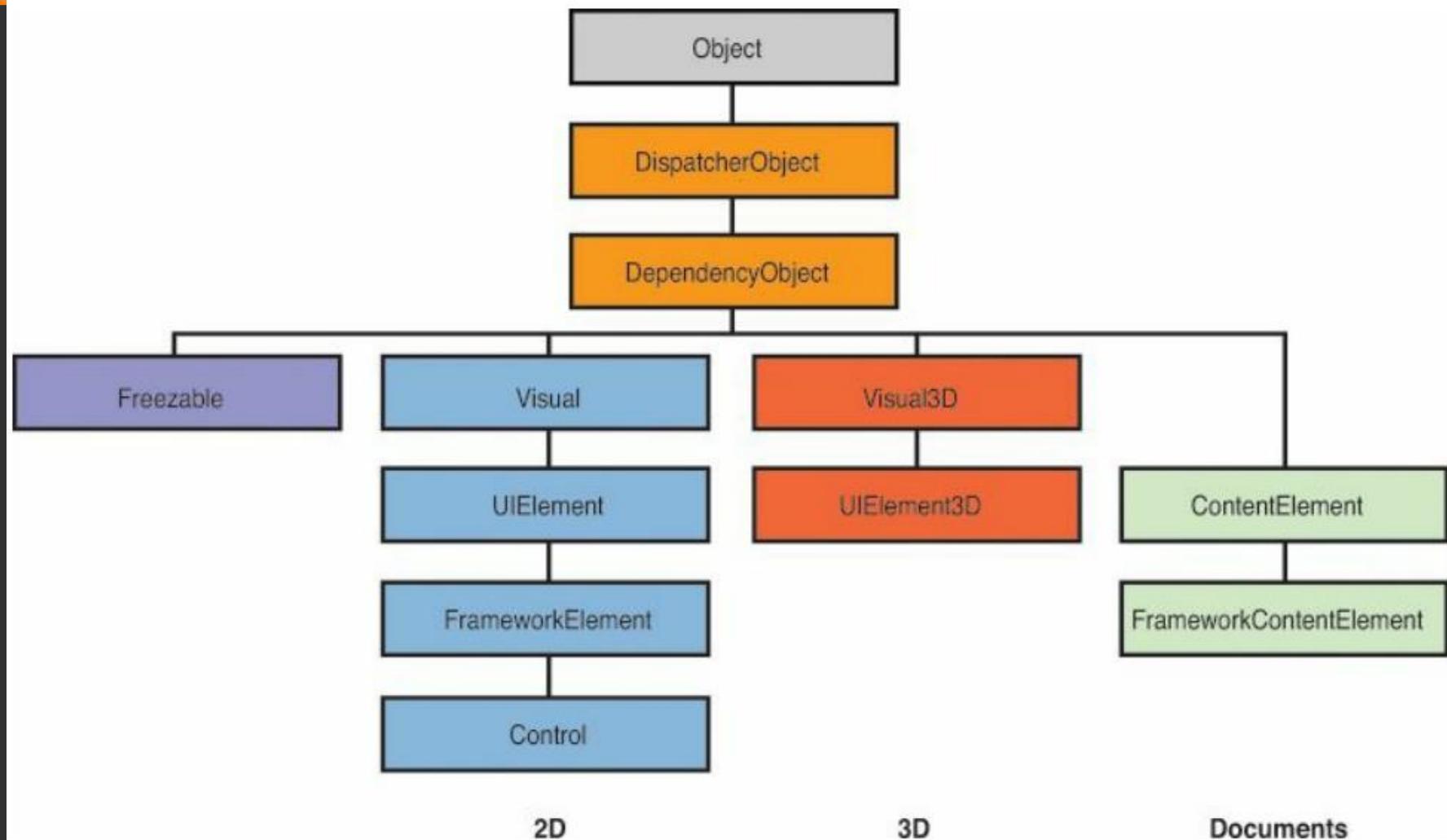
Технологія Windows Presentation Foundation

Питання 2.2.



- Presentation Framework – Holds top level WPF types includes Window, Controls, Styles, and Layout Panels etc. The code and controls written in WPF Application is mostly interacting with this layer.
- Presentation Core – Holds base types such as UI Element and Visual. Almost all the controls you are directly interacting with are derived from these types. Presentation Framework uses most of the types defined in this layer.
- MilCore – Media Integration Library is core rendering system. MIL is unmanaged code. This layer converts WPF elements into the format that Direct3D expects. Windows7 and Windows Vista uses this assembly to render its Desktop.
- WindowsCodecs – provides supports for imaging like image processing, image displaying and scaling etc.
- Direct3D – This layer is used to render graphics created using WPF Applications.

Базові класи WPF



Огляд ієрархії класів

- Фундаментальні класи технології WPF:
 - **Object** - базовий клас, від якого успадковуються решта .NET-класів.
 - **DispatcherObject** - базовый класс, предназначенный для объектов, к которым можно обращаться только из того потока, где они были созданы.
 - Большинство классов WPF наследуют DispatcherObject и, следовательно, принципиально небезопасны относительно потоков.
 - Слово Dispatcher в имени класса относится к реализованному в WPF варианту цикла обработки сообщений Win32.
 - **DependencyObject** - базовый класс, предназначенный для объектов, поддерживающих свойства зависимости.

Огляд ієрархії класів

- Есть несколько фундаментальных для работы WPF классов:
 - `Freezable` - базовый класс для объектов, которые можно «заморозить» в состоянии, разрешающем только чтение, - ради повышения производительности. К замороженным объектам можно безопасно обращаться из разных потоков, в отличие от объектов всех прочих классов, производных от `DispatcherObject`. Замороженный объект нельзя разморозить, однако можно клонировать, в результате чего получается незамороженная копия. По большей части объекты `Freezable` - это графические примитивы: кисти, перья, геометрические фигуры и

Огляд ієрархії класів

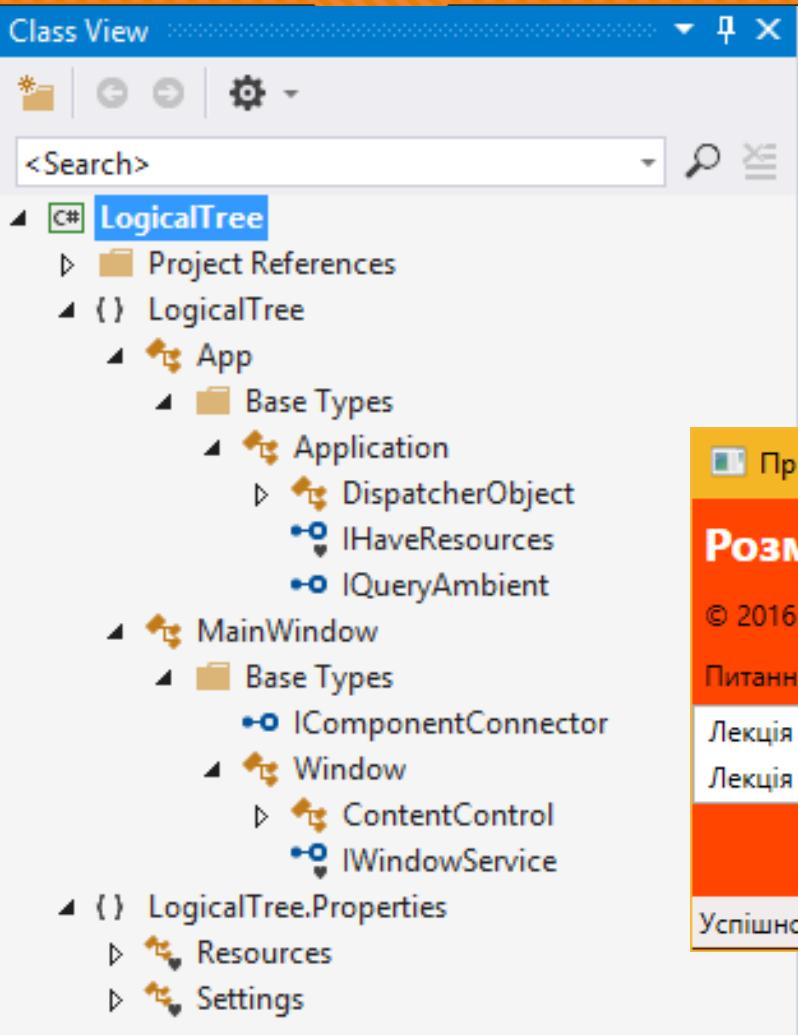
- Есть несколько фундаментальных для работы WPF классов:
 - Visual - базовый класс для объектов, имеющих двумерное визуальное представление. Визуальные объекты подробно рассматриваются в главе 15 «Двумерная графика».
 - UIElement - базовый класс для двумерных визуальных объектов с поддержкой маршрутизации событий, привязки команд, компоновки и захвата фокуса. Эти механизмы обсуждаются в главе 5 «Компоновка с помощью панелей» и в главе 6 «События ввода: клавиатура, мышь, стилус и мультисенсорные устройства».
 - Visual3D — базовый класс для объектов, имеющих трехмерное визуальное представление. Рассматривается в главе 16 «Трехмерная графика».
 - UIElement3D - базовый класс для трехмерных визуальных объектов с поддержкой маршрутизации событий, привязки команд и захвата фокуса. Также рассматривается в главе 16.

Огляд ієрархії класів

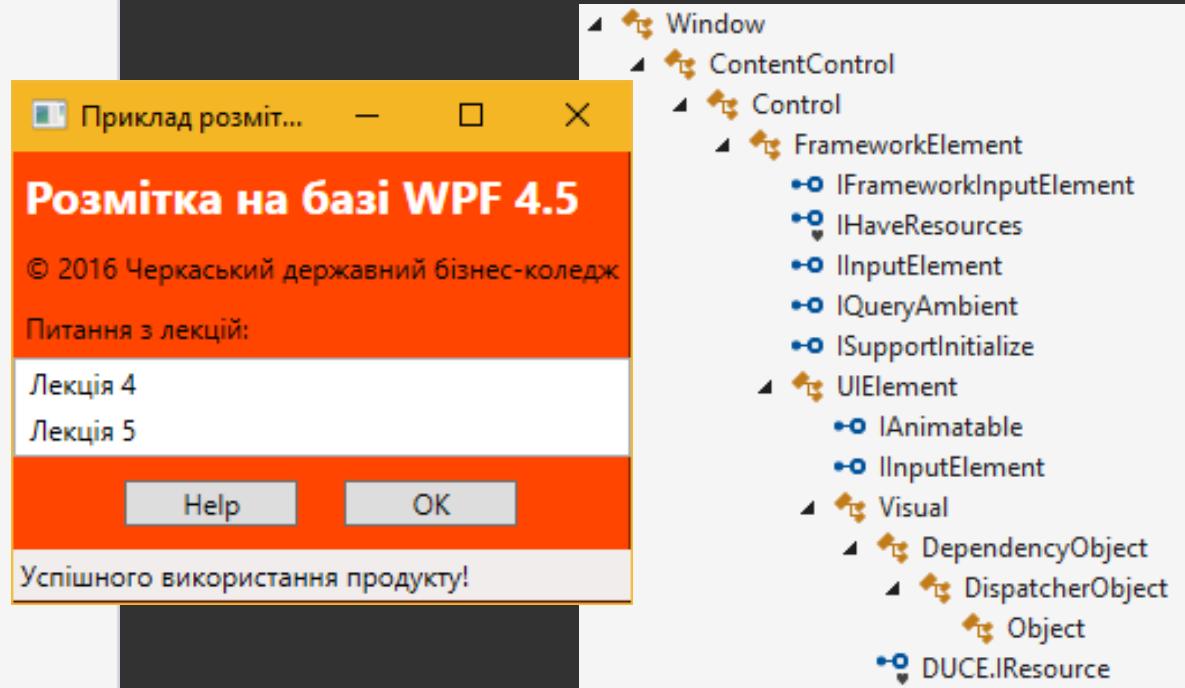
- ContentElement - базовый класс, аналогичный UIElement, но предназначенный для тех частей содержимого, которые относятся к документам и потому не имеют собственного механизма визуализации.
 - Чтобы объект типа ContentElement появился на экране, им должен владеть объект класса, производного от Visual.
 - Часто для правильной визуализации объект ContentElement нуждается в нескольких объектах Visual (охватывающих несколько строк, столбцов и страниц).
- FrameworkElement - базовый класс, добавляющий поддержку стилей, привязки к данным, ресурсов и нескольких механизмов, общих для всех элементов управления в Windows, в частности всплывающих подсказок и контекстных меню.
- FrameworkContentElement - аналог FrameworkElement для содержимого.
- Control - базовый класс для таких хорошо знакомых элементов управления, как Button, ListBox и StatusBar. Класс Control добавляет к своему базовому классу FrameworkElement множество свойств, например Foreground, Background и FontSize, а также возможность тотального изменения стиля.

```
<Window x:Class="LogicalTree.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:LogicalTree"
    mc:Ignorable="d"
    Title="Приклад розмітки на базі WPF 4.5" SizeToContent="WidthAndHeight"
    Background="OrangeRed" d:DesignWidth="317.674" d:DesignHeight="252.63">
<Grid>
    <StackPanel>
        <Label FontWeight="Bold" FontSize="20" Foreground="White">
            Розмітка на базі WPF 4.5
        </Label>
        <Label>© 2016 Черкаський державний бізнес-коледж</Label>
        <Label>Питання з лекцій:</Label>
        <ListBox>
            <ListBoxItem>Лекція 4</ListBoxItem>
            <ListBoxItem>Лекція 5</ListBoxItem>
        </ListBox>
        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
            <Button MinWidth="75" Margin="10">Help</Button>
            <Button MinWidth="75" Margin="10">OK</Button>
        </StackPanel>
        <StatusBar>Успішного використання продукту!</StatusBar>
    </StackPanel>
</Grid>
</Window>
```

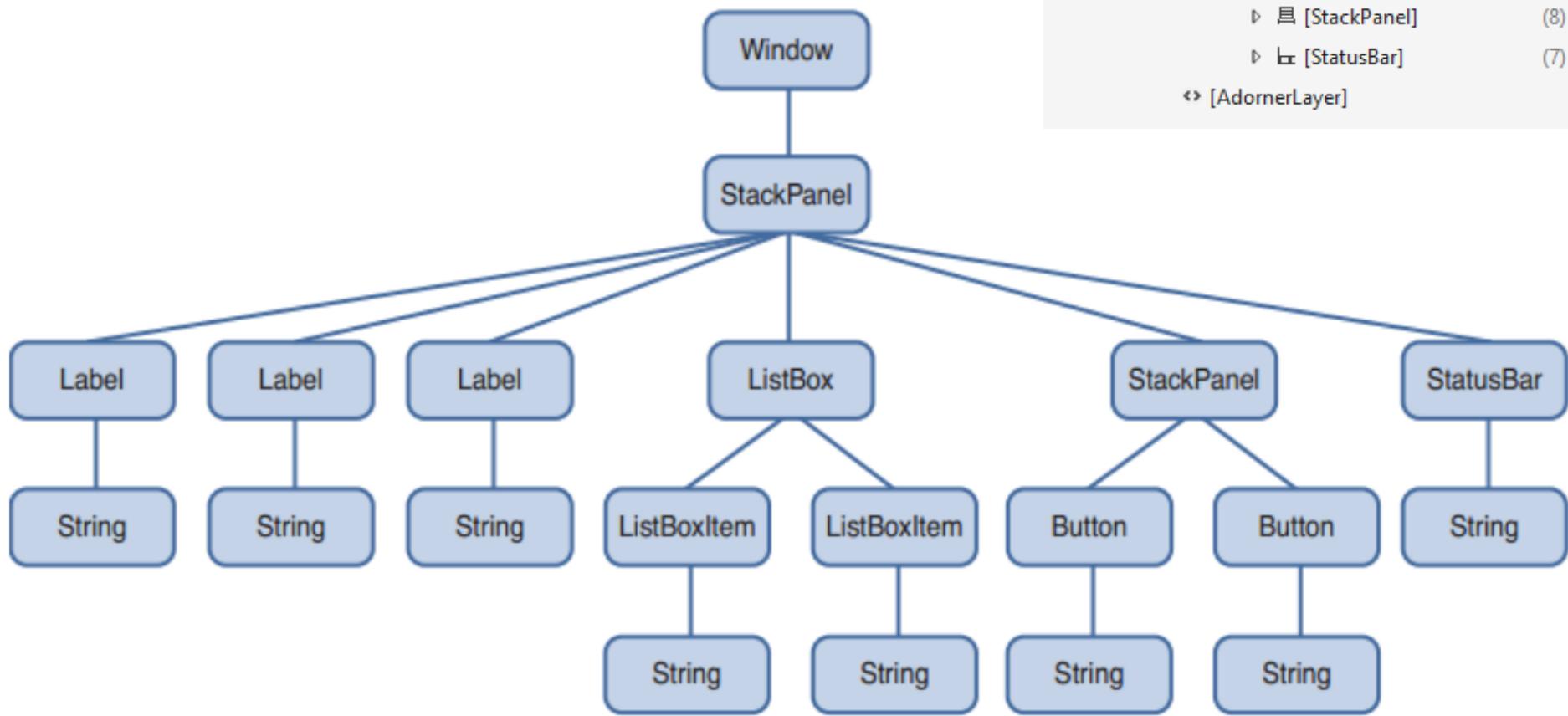
Логічні та візуальні дерева



- В WPF пользовательский интерфейс представляет собой дерево объектов, которое называется **логическим деревом**.
- Тут корнем логичного дерева є об'єкт Window.
 - Дочірній елемент StackPanel містить кілька простих елементів управління та ще один StackPanel з двома кнопками Button.



Логическое дерево, соответствующее коду в листе



↳ [MainWindow]	(54)
↳ [Border]	(53)
↳ [AdornerDecorator]	(52)
↳ [ContentPresenter]	(50)
↳ [Grid]	(49)
↳ [StackPanel]	(48)
↳ [Label]	(3)
↳ [Label]	(3)
↳ [Label]	(3)
↳ [ListBox]	(18)
↳ [StackPanel]	(8)
↳ [StatusBar]	(7)
↳ [AdornerLayer]	

Візуальне дерево

- Логическое дерево в WPF фактически дает упрощенную картину того, что в действительности происходит при визуализации элементов.
- Полное дерево содержащее все визуализированные элементы, называется визуальным деревом.
 - раскрытое логическое дерево, в котором каждый узел является вершиной поддерева, содержащего его визуальные компоненты.
 - Например, ListBox - логически единый элемент управления, однако по умолчанию его визуальное представление состоит из более простых WPF-элементов: рамки Border, двух полос прокрутки ScrollBar и др.
- В визуальном дереве представлены не все узлы логического дерева, а лишь элементы, производные от классов System.Windows.Media.Visual или System.Windows.Media.Visual3D.
 - Прочие элементы не включаются, потому что не обладают собственник поведением визуализации.

Візуальне дерево



несмотря на то, что визуальное дерево является существенной частью инфраструктуры WPF, задумываться о нем имеет смысл, только если вы собираетесь радикально изменять стили элементов управления или выполнять низкоуровневое рисование

Властивості залежності

- У WPF з'явились **властивості залежності**; використовуються для призначення стилів, автоматичної прив'язки до даних, анімації та ін.
 - Свойство зависимости зависит от нескольких поставщиков, которые определяют его значение во время выполнения.
 - Поставщиком может быть анимация, постоянно изменяющая значение свойства, родительский элемент, распространяющий значение своего свойства на потомков, и т. д.
 - Пожалуй, наиболее существенной особенностью свойства зависимости является встроенная возможность генерировать уведомления об изменениях своего значения!
- Причина— стремление описать развитую функциональность на уровне декларативной разметки.
 - Свойства можно задавать в XAML-коде (непосредственно или с помощью инструментов конструирования) без написания процедурного кода.
 - Но в отсутствие механизмов, предлагаемых свойствами зависимости, было бы весьма трудно получить желаемый результат без дополнительного кода.

Властивості залежності

- свойства зависимости расширяют функциональность обычных свойств .NET в следующих направлениях:
 - Уведомление об изменениях
 - Наследование значений свойств
 - Поддержка нескольких поставщиков
- свойство зависимости - это обычное свойство .NET, которое включено в состав дополнительной инфраструктуры, предоставляемой WPF.
 - Для этого в WPF предусмотрены специальные API;
 - ни один .NET-совместимый язык программирования (кроме XAML) ничего не знает о свойствах зависимости.

Реализация свойства зависимости

```
public class Button : ButtonBase
{
    // Свойство зависимости
    public static readonly DependencyProperty IsDefaultProperty;

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

    // Обертка в виде обычного свойства .NET (необязательно)
    public bool IsDefault
    {
        get { return (bool)GetValue(Button.IsDefaultProperty); }
        set { SetValue(Button.IsDefaultProperty, value); }
    }
    // Метод, вызываемый при изменении свойства (необязательно)
    private static void OnIsDefaultChanged(
        DependencyObject o, DependencyPropertyChangedEventArgs e) { ... }
    ...
}
```

По принятому соглашению все поля типа `DependencyProperty` открыты, статические и имеют имя, оканчивающееся словом `Property`.

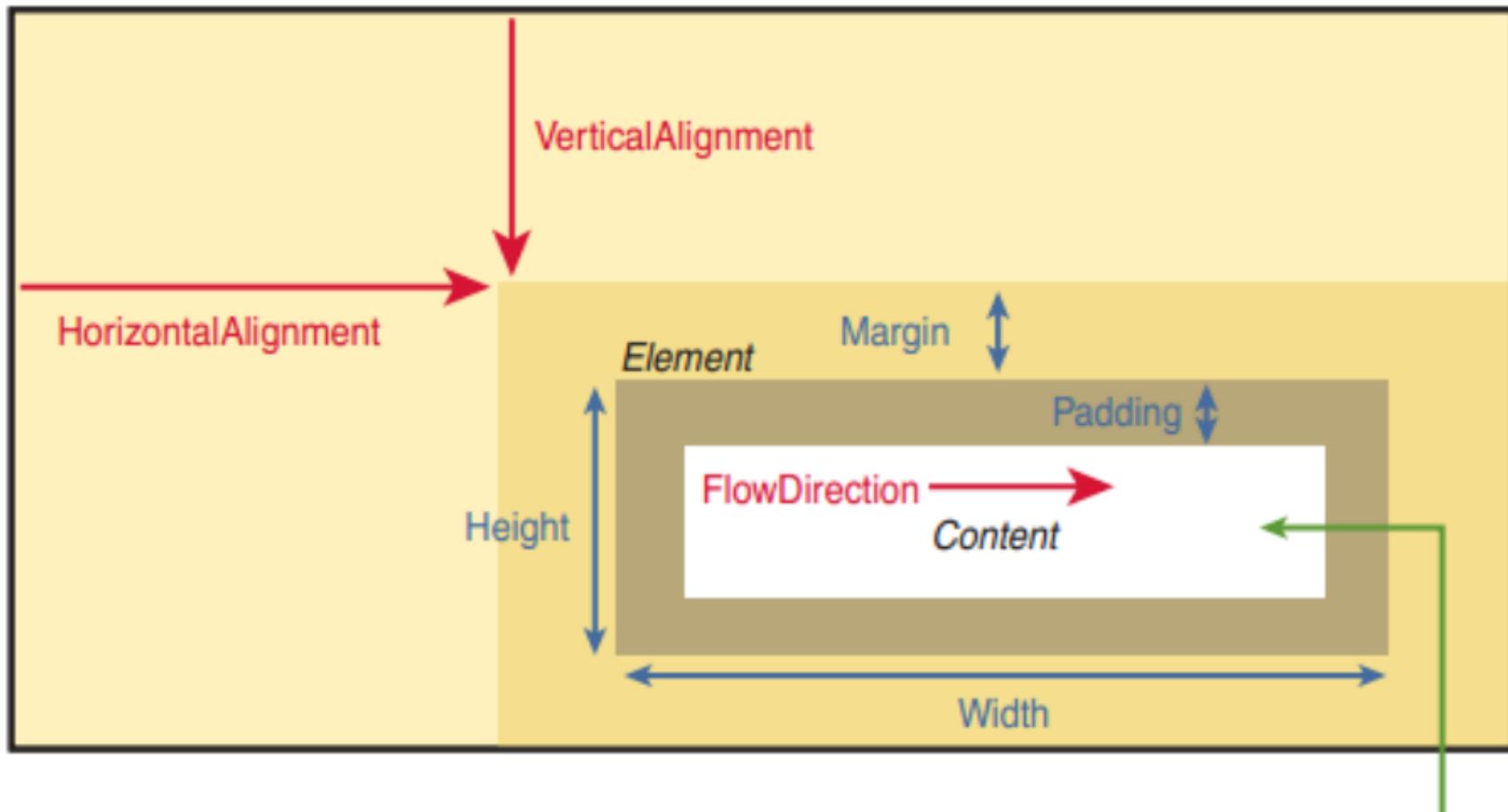
Некоторые компоненты инфраструктуры требуют обязательного соблюдения этого соглашения, например средства локализации, загрузчики XAML и пр.

Елементи управління та їх основні властивості

- Процедура задания размеров и положений элементов управления (и других элементов) называется компоновкой, или версткой макета.
 - В ее основе лежит механизм взаимодействия между элементами-родителями и их потомками.
 - Совместно они договариваются об окончательных размерах и положении.
 - Родительские элементы, поддерживающие компоновку нескольких детей, называются панелями, они наследуют абстрактному классу `System.Windows.Controls.Panel`.
 - Все элементы, участвующие в процессе компоновки (как родители, так и потомки), наследуют классу `System.Windows.UIElement`.

Основные свойства, управляющие компоновкой дочерних элементов (в данном питанні)

Panel



LayoutTransform
RenderTransform

Управление размером. Свойства Height и Width

- Во всех классах, производных от FrameworkElement, есть свойства Height (высота) и Width (ширина) (типа double), а также MinHeight, MaxHeight, MinWidth и MaxWidth, которыми можно пользоваться для задания допустимых диапазоне» значений.
 - Все эти свойства можно задавать в любой комбинации как в процедурном коде, так и в XAML.
 - Обычно элемент стремится принять минимально возможный размер, поэтому если задано свойство MinHeight или MinWidth, то при визуализации выбирается именно такая высота или ширина при условии, что содержимое не вынуждает увеличить размер.
 - Но увеличение можно ограничить с помощью свойств MaxHeight и MaxWidth (при условии, что эти значения больше соответствующих минимальных).
 - Если одновременно с минимальными и максимальными значениями заданы свойства Height и Width, то последние имеют приоритет при условии, что попадают внутрь диапазона между Min и Max.
 - По умолчанию MinHeight и MinWidth равны 0, а MaxHeight и MaxWidth - величине

ПРЕДУПРЕЖДЕНИЕ

Избегайте явного задания размеров!

Если явно задавать размеры элементов управления, особенно производных от класса ContentControl, например Button и Label, то возникает риск отсечения текста в случае, когда пользователь изменяет системный шрифт или текст переводится на другие языки. Поэтому лучше не задавать размеры явно, если без этого можно обойтись. К счастью, благодаря наличию панелей необходимость явно задавать размеры возникает редко.

КОПНЕМ ГЛУБЖЕ

Специальное значение длины "Auto"

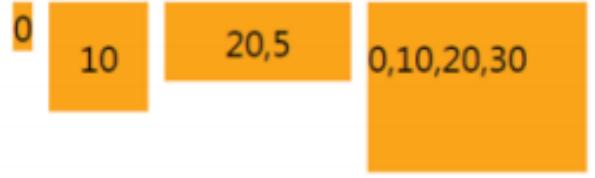
Свойства Height и Width класса FrameworkElement по умолчанию принимают значение Double.NaN (NaN означает *not a number* — «не число»), которое означает, что размер элемента подстраивается под размер содержимого. Это значение можно и явно задать в XAML-коде в виде строки "NaN" (чувствительной к регистру) или более предпочтительной строки "Auto" (нечувствительной к регистру), что обеспечивает конвертер типа LengthConverter, ассоциированный с этими свойствами. Чтобы проверить, выбирается ли размер элемента автоматически, можно воспользоваться статическим методом Double.IsNaN.

- Ситуация осложняется тем, что в классе FrameworkElement есть еще несколько свойств, относящихся к размеру:
 - DesiredSize (наследуется от UIElement)
 - RenderSize (наследуется от UIElement)
 - ActualHeight и ActualWidth
- В отличие от остальных шести свойств, они являются не входными данными для процедуры компоновки, а выходными — представляющими результат компоновки, и потому доступны только для чтения.
- Свойство элемента DesiredSize вычисляется в процессе компоновки на основе значений других свойств (в том числе вышеупомянутых Width, Height, MinXXX и MaxXXX) и места, родитель готов выделить.
- Оно используется панелями для внутренних целей.
- Свойство RenderSize представляет окончательный размер элемента по завершении компоновки, а ActualHeight и ActualWidth в точности то же самое, RenderSize. Height и RenderSize. Width.

- Запомните: каким бы способом ни был задан размер элемента - явно, с помощью допустимых диапазонов значений или не задан вовсе, - родитель вправе изменить окончательный размер элемента на экране.
- Поэтому три упомянутых свойства бывают полезны в тех редких случаях, когда поведение программы зависит от размера элемента.
- С другой стороны, значения остальных свойств, относящихся к размеру очень интересны для формулирования алгоритма.
- Например, если свойств Height и Width не заданы явно, то они будут иметь значение Double.NaN, каким бы ни оказался истинный размер элемента.

Свойства Margin и Padding

- Свойство Margin определено для всех объектов, производных от FrameworkElement, свойство Padding во всех элементах управления, производных от класса Control (а также в классе Border).
 - Различие в том, что Margin задает внешнее поле вокруг элемента, а Padding – внутренний отступ между содержимым элемента и его границами.
 - Оба свойства имеют тип System.Windows.Thickness; это любопытный класс, который может представлять одно, два или четыре значения типа double.



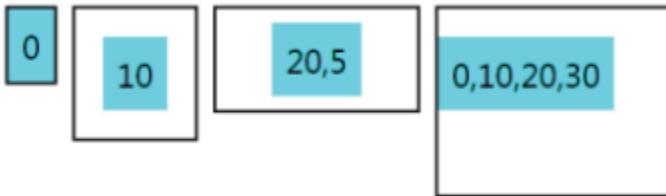
```
<!-- Отступы: -->

<!-- 1 значение: один и тот же отступ со всех четырех сторон: -->
<Label Padding="0" Background="Orange">0</Label>
<Label Padding="10" Background="Orange">10</Label>

<!-- 2 значения: первое значение относится к левому и правому
      отступам, второе - к верхнему и нижнему:-->
<Label Padding="20,5" Background="Orange">20,5</Label>

<!-- 4 значения: левый, верхний, правый, нижний: -->
<Label Padding="0,10,20,30" Background="Orange">0,10,20,30</Label>
```

Елемент управління Button



- Второй набор меток ключей в рамки Border, поскольку иначе поля были бы не видны.
- Хотя на рисунке это и не показано, свойство Margin (но не Padding) может принимать отрицательные значения.

```
<!-- Поля: -->

<Border BorderBrush="Black" BorderThickness="1">
    <!-- Поле отсутствует: -->
    <Label Background="Aqua">0</Label>
</Border>

<Border BorderBrush="Black" BorderThickness="1">
    <!-- 1 значение: одно и то же поле со всех четырех сторон: -->
    <Label Margin="10" Background="Aqua">10</Label>
</Border>

<Border BorderBrush="Black" BorderThickness="1">
    <!-- 2 значение: первое значение относится к левому и правому,
        полям, второе – к верхнему и нижнему : -->
    <Label Margin="20,5" Background="Aqua">20,5</Label>
</Border>

<Border BorderBrush="Black" BorderThickness="1">
    <!-- 4 значение: левое,верхнее,правое,нижнее: -->
    <Label Margin="0,10,20,30" Background="Aqua">0,10,20,30</Label>
</Border>
```

КОПНЕМ ГЛУБЖЕ

Синтаксис задания значений типа Thickness

Синтаксис задания значений через запятую, поддерживаемый свойствами Margin и Padding, обеспечивает - догадайтесь, кто - конвертер типа System.Windows.ThicknessConverter, который конструирует объект типа Thickness из строки. В классе Thickness определены два конструктора: первый принимает одно значение типа double, второй - четыре. Следовательно, в программе на языке C# допустимы следующие формы:

```
myLabel.Margin = new Thickness(10);      // То же, что Margin="10" в XAML  
myLabel.Margin = new Thickness(20, 5, 20, 5); // То же, что Margin="20,5" в XAML  
myLabel.Margin = new Thickness(0,10,20,30); // То же, что Margin="0,10,20,30" в XAML
```

Обратите внимание, что удобный синтаксис с двумя числами доступен только через конвертер типа!

Какие единицы измерения применяются в WPF?

- Конвертер LengthConverter, ассоциируемый с различными свойствами, касающимися длин, поддерживает явное задание единиц измерения см, pt, in и px (по умолчанию).
 - По умолчанию все абсолютные измерения выражаются в независимых от устройства пикселях.
 - Такие «логические пиксели» представляют 1/96 дюйма независимо от разрешения экрана, выраженного в точках на дюйм (DPI).
 - Отметим, что количество независимых от устройства пикселов всегда задается в виде значения с двойной точностью, то есть может быть дробным.
- Точная величина - 1/96 дюйма - не так существенна, а выбрана она была потому, что на типичном экране с разрешением 96 DPI один независимый от устройства пикsel в точности совпадает с одним физическим пикселом.
 - Разумеется, понятие «истинного» дюйма зависит от физического устройства отображения. Если приложение нарисует отрезок длиной 1 дюйм на экране моего ноутбука, то при выводе изображения на проектор длина этого отрезка, конечно, окажется больше!

Свойство Visibility

- Тип свойства элемента Visibility - не Boolean, а перечисление System.Windows.Visibility с тремя состояниями, то есть оно может принимать три значения:
 - Visible – элемент виден и участвует в компоновке.
 - Collapsed – элемент не виден и не участвует в компоновке.
 - Hidden – элемент не виден, но тем не менее участвует в компоновке.
- Свернутый (Collapsed) элемент, по существу, имеет нулевой размер, тогда как скрытый (Hidden) элемент сохраняет свой первоначальный размер



Властивості FlatStyle та FlatAppearance

- панель StackPanel со свернутой кнопкой:

```
<StackPanel Height="100" Background="Aqua">
<Button Visibility="Collapsed">Collapsed Button</Button>
<Button>Below a Collapsed Button</Button>
</StackPanel>
```

- сравнивается с панелью StackPanel со скрытой кнопкой:

```
<StackPanel Height="100" Background="Aqua">
<Button Visibility="Hidden">Hidden Button</Button>
<Button>Below a Hidden Button</Button>
</StackPanel>
```

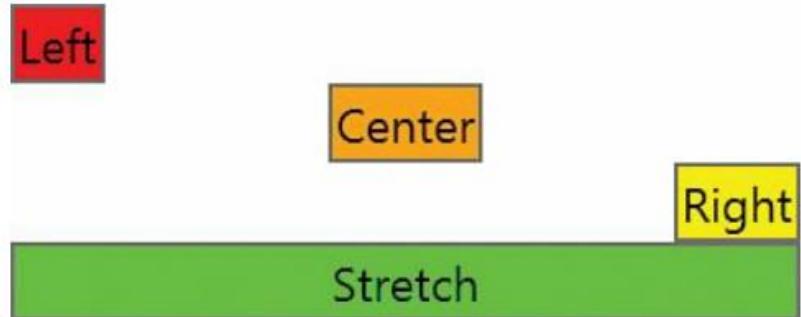
- Скрытая кнопка занимает место на экране, а свернутая - нет

Управление положением

- С помощью свойств `HorizontalAlignment` и `VerticalAlignment` элемент может управлять распределением избыточного пространства, выделенного ему родителем.
 - Значениями свойств являются одноименные перечисления, которые определены в пространстве имен `System.Windows`:
 - `HorizontalAlignment - Left, Center, Right, Stretch`
 - `VerticalAlignment - Top, Center, Bottom, Stretch`

```
<StackPanel>
<Button HorizontalAlignment="Left" Background="Red">Left</Button>
<Button HorizontalAlignment="Center" Background="Orange">Center</Button>
<Button HorizontalAlignment="Right" Background="Yellow">Right</Button>
<Button HorizontalAlignment="Stretch" Background="Lime">Stretch</Button>
</StackPanel>
```

Эти два свойства полезны ТОЛЬКО в случае, когда родительская панель выделяет дочернему элементу больше места, чем тому необходимо.



КОПНЕМ ГЛУБЖЕ

Взаимодействие между типом выравнивания Stretch и явным заданием размера элемента

Даже если для элемента в качестве выравнивания задано растяжение (Stretch) по горизонтали или по вертикали, приоритет все равно отдается явно заданной высоте Height или ширине Width. Свойства MaxHeight и MaxWidth также более приоритетны, но только в том случае, когда их значения меньше размера, получившегося после растяжения. Аналогично свойствам MinHeight и MinWidth приоритет отдается лишь тогда, когда их значения больше размера, получившегося после растяжения. Если свойство Stretch используется в контексте, где на размер элемента налагаются ограничения, то оно действует как тип выравнивания Center (или Left, если элемент слишком велик и не может быть отцентрирован внутри своего родителя).

Выравнивание содержимого

- Помимо свойств HorizontalAlignment и VerticalAlignment, в классе Control имеются свойства HorizontalContentAlignment и VerticalContentAlignment.
 - Они определяют порядок размещения содержимого внутри элемента управления.

```
<StackPanel>
    <Button HorizontalContentAlignment="Left" Background="Red">Left</Button>
    <Button HorizontalContentAlignment="Center" Background="Orange">Center</Button>
    <Button HorizontalContentAlignment="Right" Background="Yellow">Right</Button>
    <Button HorizontalContentAlignment="Stretch" Background="Lime">Stretch</Button>
</StackPanel>
```

Свойства, управляющие выравниванием содержимого, принадлежат тем же типам перечисления, что и соответствующие свойства выравнивания.

Однако по умолчанию свойство HorizontalContentAlignment равно Left, а VerticalContentAlignment равно Top.



Свойство FlowDirection

- Свойство `FlowDirection`, определенное в классе `FrameworkElement` (и еще нескольких), позволяет изменить направление визуализации внутреннего содержимого элемента.
- Оно применимо к некоторым панелям, где влияет на размещение дочерних элементов, а также модифицирует способ выравнивания содержимого внутри дочерних элементов.
- Тип этого свойства - перечисление `System.Windows.FlowDirection`, принимающее два значения: `LeftToRight` (по умолчанию в классе `FrameworkElement`) и `RightToLeft`.
- Идея `FlowDirection` заключается в том, что для языка, записываемого справа налево, должно быть задано направление `RightToLeft`.

Свойство FlowDirection

```
<StackPanel>
    <Button FlowDirection="LeftToRight"
        HorizontalContentAlignment="Left" VerticalContentAlignment="Top"
        Height="40" Background="Red">LeftToRight</Button>
    <Button FlowDirection="RightToLeft"
        HorizontalContentAlignment="Left" VerticalContentAlignment="Top"
        Height="40" Background="Orange">RightToLeft</Button>
</StackPanel>
```

Отметим, что свойство FlowDirection не оказывает влияния на направление записи букв в надписи внутри кнопок.

Английские буквы всегда записываются слева направо, арабские - справа налево.

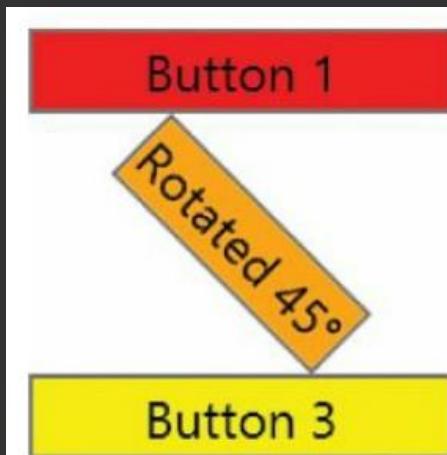
Но понятия левого и правого во всех остальных частях интерфейса, которые обычно должны соответствовать направлению записи букв, меняются местами.



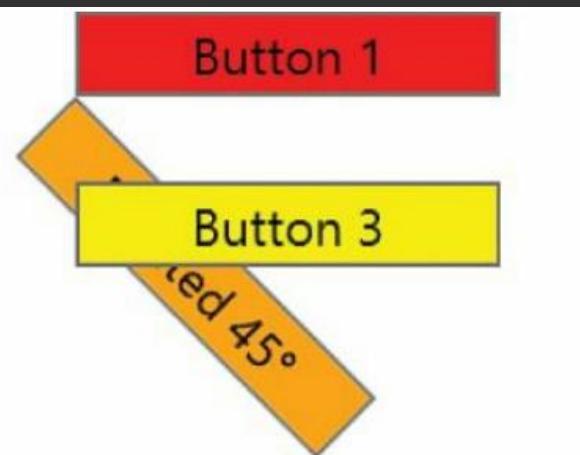
Свойство FlowDirection необходимо задавать явно в соответствии с текущей культурой

Применение преобразований

- Во всех подклассах FrameworkElement имеется два свойства типа Transform, позволяющих применять преобразования:
 - LayoutTransform - применяется до компоновки элемента
 - RenderTransform (унаследовано от UIElement) - применяется после завершения компоновки (непосредственно перед визуализацией элемента)

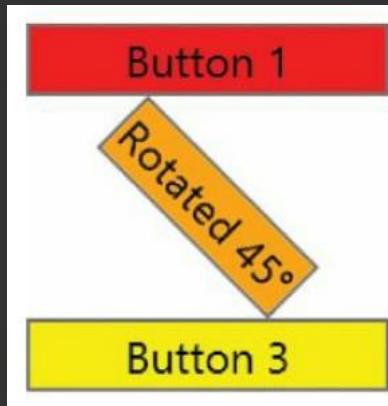


Rotation as a LayoutTransform

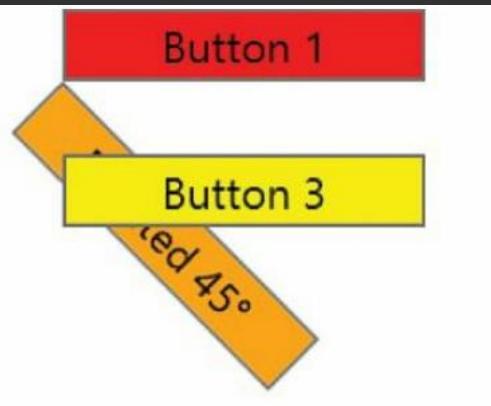


Rotation as a RenderTransform

- В обоих случаях преобразование применяется ко второй из трех кнопок.
 - Но если оно применено как LayoutTransform, то третья кнопка сдвигается вниз, а если как RenderTransform, то третья кнопка размещается так, будто вторая не поворачивалась вовсе.
- В классе UIElement имеется также полезное свойство RenderTransformOrigin, представляющее начальную точку преобразования (которая остается неподвижной).
 - Для преобразования RotateTransform началом является левый верхний угол кнопки, вокруг которого кнопка поворачивается.
 - Для преобразований, применяемых в режиме LayoutTransform, понятие начальной точки не определено, потому что положение преобразованного элемента диктуется правилами компоновки, применяемыми родителем.



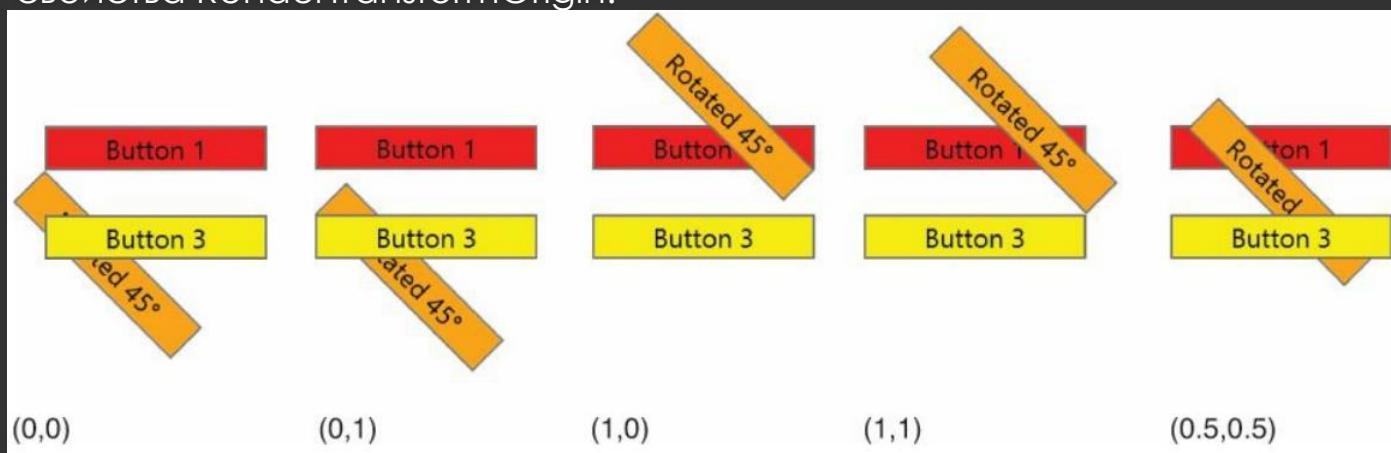
Rotation as a LayoutTransform



Rotation as a RenderTransform

Свойство RenderTransformOrigin

- Свойство RenderTransformOrigin имеет тип System.Windows.Point и по умолчанию равно (0,0). Этой точке соответствует левый верхний угол элемента.
 - Точка (0,1) представляет левый нижний угол, (1,0) - правый верхний угол, а (1,1) - правый нижний угол.
 - Можно задавать и числа, большие 1, - тогда начальная точка окажется вне границ элемента.
 - Дробные значения также допустимы. В частности, точка (0.5,0.5) представляет середину объекта.
 - На рис. показано пять начальных точек, обычно задаваемых в качестве значения свойства RenderTransformOrigin.



- Благодаря конвертеру System.Windows.PointConverter значение RenderTransform.Origin можно задавать в XAML в виде двух чисел, разделенных запятой (без скобок).

```
<Button RenderTransformOrigin="0.5,0.5" Background="Orange">
<Button.RenderTransform>
<RotateTransform Angle="45"/>
</Button.RenderTransform>
    Rotated 45°
</Button/>
```

для стандартных элементов управления со стилями по умолчанию подобное преобразование выглядит нелепо.
Часто это имеет смысл в приложениях с сильно перегруженными темами, но даже для стилизованных по умолчанию элементов преобразования в сочетании с анимацией могут создать интересные эффекты.

- рассмотрим все пять встроенных двумерных преобразований, определенных в пространстве имен System.Windows.Media:
 - RotateTransform
 - ScaleTransform
 - SkewTransform
 - TranslateTransform
 - MatrixTransform

Преобразование RotateTransform

- поворачивает элемент в соответствии со следующими тремя свойствами типа double:
 - Angle - угол поворота в градусах (по умолчанию 0)
 - CenterX - абсцисса центра поворота (по умолчанию 0)
 - CenterY - ордината центра поворота (по умолчанию 0)
- Точка (CenterX,CenterY), равная по умолчанию (0,0), соответствует левому верхнему углу.
- Свойства CenterX и CenterY принимаются во внимание, только если преобразование применяется в режиме RenderTransform, потому что для преобразования в режиме LayoutTransform положение центра поворота определяется родительской панелью.

В чем разница между использованием свойств CenterX и CenterY для преобразований вида RotateTransform и свойством RenderTransformOrigin элемента типа UIElement?

Складывается впечатление, что, когда преобразование применяется к элементу типа UIElement, свойства CenterX и CenterY избыточны, так как уже имеется свойство RenderTransformOrigin. Ведь оба механизма определяют начальную точку преобразования и оба работают, только если преобразование применяется в режиме RenderTransform.

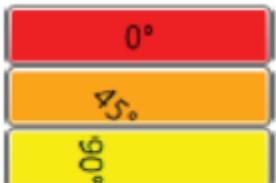
Однако же CenterX и CenterY задают абсолютное положение начальной точки, тогда как RenderTransformOrigin - относительное. Значения задаются в независимых от устройства пикселях, так что для правого верхнего угла элемента с шириной Width, равной 20, свойство CenterX будет равно 20, а CenterY - 0, а не (1,0), как для RenderTransformOrigin. Кроме того, при комбинировании нескольких преобразований RenderTransform(см. следующую главу) задание CenterX и CenterY для отдельных преобразований обеспечивает более точный контроль. Наконец, раздельные значения CenterX и CenterY типа double проще использовать для привязки к данным, чем одно значение RenderTransformOrigin типа Point.

Тем не менее, свойство RenderTransformOrigin, вообще говоря, более полезно, чем CenterX и CenterY. В типичном случае, когда элемент поворачивается относительно своей середины, относительные координаты (0.5,0.5), задаваемые с помощью RenderTransformOrigin, проще записать в XAML. Для достижения того же эффекта с помощью CenterX и CenterY пришлось бы писать процедурный код, вычисляющий абсолютные смещения.

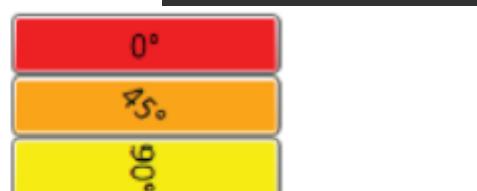
Отметим, что задавать RenderTransformOrigin для элемента можно одновременно с установкой значений CenterX и CenterY для его преобразования. В этом случае пары значений X и Y комбинируются для вычисления окончательной начальной точки.

- демонстрируется, что происходит, когда преобразование RotateTransform применяете в режиме RenderTransform к внутреннему содержимому кнопок с двумя различными значениями RenderTransformOrigin.
- Может показаться, что элементы TextBlock внутри Button в левой части повернуты не вокруг левого верхнего угла кнопки, но это потому, что сам элемент TextBlock чуть больше содержащегося внутри него текста.

```
<Button Background="Orange">
  <TextBlock RenderTransformOrigin="0.5,0.5">
    <TextBlock.RenderTransform>
      <RotateTransform Angle="45"/>
    </TextBlock.RenderTransform>
    45°
  </TextBlock>
</Button>
```



RotateTransform имеет параметризованные конструкторы, которые принимают значения угла или угла и центра, для удобства выполнения преобразования из процедурного кода.



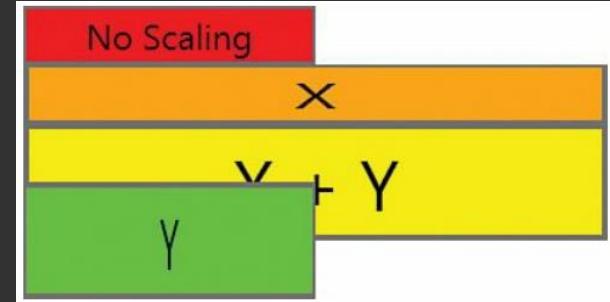
Преобразование ScaleTransform

- Преобразование ScaleTransform увеличивает или уменьшает элемент по горизонтали, по вертикали или в обоих направлениях.
- У него есть четыре свойства типа double:
 - ScaleX - коэффициент изменения ширины элемента (по умолчанию 1)
 - ScaleY - коэффициент изменения высоты элемента (по умолчанию 1)
 - CenterX - начальная точка для масштабирования по горизонтали (по умолчанию 0)
 - CenterY - начальная точка для масштабирования по вертикали (по умолчанию 0)
- Если ScaleX равно 0.5, то ширина рисуемого элемента уменьшается вдвое, а если ScaleX равно 2, то вдвое увеличивается.
- Смысл свойств CenterX и CenterY такой же, как для преобразования RotateTransform.

Преобразование ScaleTransform

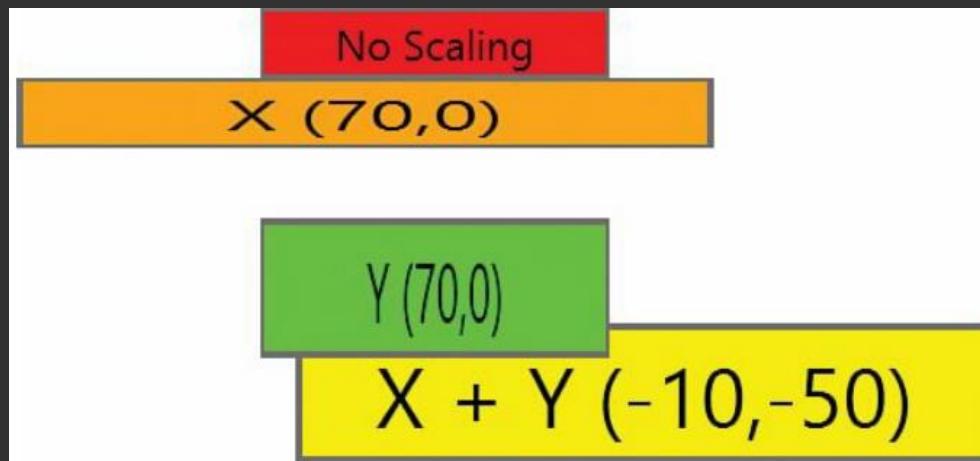
```
<StackPanel Width="100">
  <Button Background="Red">No Scaling</Button>
  <Button Background="Orange">
    <Button.RenderTransform>
      <ScaleTransform ScaleX="2"/>
    </Button.RenderTransform>
    X</Button>
  <Button Background="Yellow">
    <Button.RenderTransform>
      <ScaleTransform ScaleX="2" ScaleY="2"/>
    </Button.RenderTransform>
    X + Y</Button>
  <Button Background="Lime">
    <Button.RenderTransform>
      <ScaleTransform ScaleY="2"/>
    </Button.RenderTransform>
    Y</Button>
  </StackPanel>
```

преобразование ScaleTransform применяется к трем кнопкам Button на панели StackPanel, демонстрируя возможность независимого изменения высоты и ширины.



Преобразование ScaleTransform

- изображены кнопки из листинга, для которых дополнительно явно заданы свойства Center X и Center Y.
 - В качестве текста каждой кнопки указаны координаты начальной точки.
 - Обратите внимание, что зеленая кнопка не сдвинута влево, как оранжевая, хотя CenterX в обоих случаях равно 70. Center X принимается во внимание, только если значение отлично от 1, а CenterY - если ScaleY не равно 1.
 - Как и во всех остальных классах преобразований, в ScaleTransform определено несколько конструкторов для удобства создания объектов в процедурном коде.



FAQ

Как преобразования, подобные ScaleTransform, влияют на свойства ActualHeight и ActualWidth элемента типа FrameworkElement и на свойство RenderSize элемента типа UIElement?

Применение преобразования к элементу типа FrameworkElement никогда не изменяет значений этих свойств. Это справедливо вне зависимости от того применяется преобразование в режиме RenderTransform или LayoutTransform. Поэтому из-за преобразований эти свойства могут сообщать «ложный» размер элемента на экране. Например, у всех кнопок на рис. 4.11 и 4.12 величины ActualHeight, ActualWidth и RenderSize одинаковы.

Быть может, вас удивляет такая «ложь», но это правильное решение. Во-первых, не вполне понятно, как следует выражать эти значения для некоторых преобразований. Важнее, однако, то, что цель преобразования - изменить внешний вид элемента так, чтобы сам элемент об этом ничего не знал. Создавая у элемента иллюзию того, что он визуализируется нормально, мы можем единообразно преобразовывать произвольные элементы.

FAQ

Как преобразование ScaleTransform влияет на свойства Margin и Padding?

Свойство Padding масштабируется вместе со всем содержимым (поскольку отступ находится внутри элемента), а свойство Margin не масштабируется вовсе. Как и в случае ActualHeight и ActualWidth, числовое значение свойства Padding не изменяется, несмотря на визуальный эффект масштабирования.

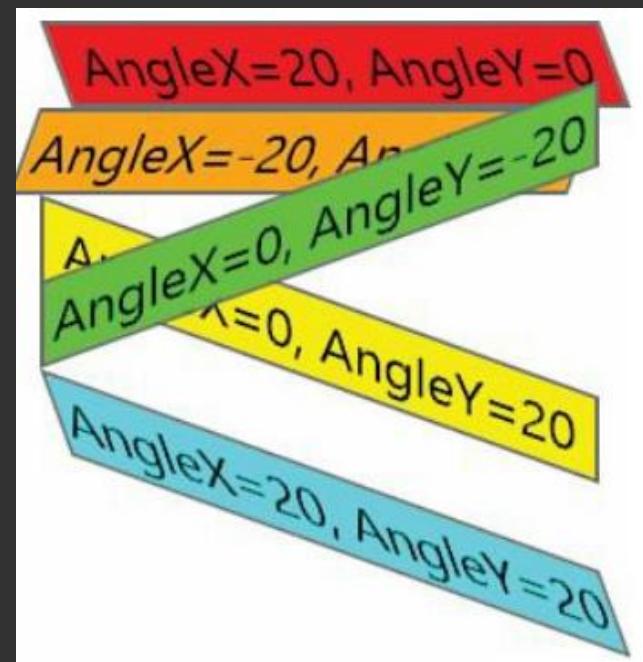
КОПНЕМ ГЛУБЖЕ

Взаимодействие между ScaleTransform и выравниванием типа Stretch

Если преобразование ScaleTransform применяется в режиме LayoutTransform к элементу, который уже растянут в направлении масштабирования, то оно принимается во внимание только в случае, когда размер после масштабирования больше размера, получившегося в результате растяжения.

Преобразование SkewTransform

- Преобразование `SkewTransform` наклоняет элемент в соответствии со значениями четырех свойств типа `double`:
 - `AngleX` – угол наклона по горизонтали (по умолчанию 0)
 - `AngleY` – угол наклона по вертикали (по умолчанию 0)
 - `CenterX` – начальная точка для наклона по горизонтали (по умолчанию 0)
 - `CenterY` – начальная точка для наклона по вертикали (по умолчанию 0)



Преобразование TranslateTransform

- Преобразование TranslateTransform просто параллельно переносит элемент в соответствии со значениями двух свойств типа double:
 - X - величина смещения по горизонтали (по умолчанию 0)
 - Y- величина смещения по вертикали (по умолчанию 0)
- TranslateTransform не дает никакого эффекта, когда применяется в режиме LayoutTransform, но применение его в режиме RenderTransform удобный способ «подвинуть» элементы.
 - Чаще всего это делается динамически в ответ на действия пользователя (и, быть может, в составе анимации).
 - Маловероятно, что при работе с панелями вы захотите использовать это преобразование для компоновки статического пользовательского интерфейса.

Преобразование MatrixTransform

- Преобразование MatrixTransform представляет собой низкоуровневый механизм описания произвольного двумерного преобразования.
- У него есть единственное свойство Matrix(типа System.Windows.Media.Matrix), представляющее матрицу аффинного преобразования размером 3x3.
- все рассмотренные выше преобразования (и их комбинации) также можно осуществить с помощью MatrixTransform.

M11	M12	0
M21	M22	0
OffsetX	OffsetY	1

Конвертер типа MatrixTransform

MatrixTransform единственное преобразование, конвертер типа которого позволяет описывать его в XAML с помощью простой строки. (Класс конвертера называется `TransformConverter` и, хотя он ассоциирован с абстрактным классом `Transform`, в реальности поддерживает только тип `MatrixTransform`.) Например, чтобы переместить кнопку на 10 единиц вправо и на 20 единиц вниз, нужно написать такой код:

```
<Button RenderTransform="1,0,0,1,10,20" />
```

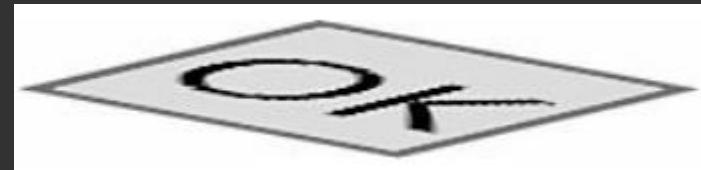
Через запятую указаны элементы матрицы в следующем порядке: M11, M12, M21, M22, OffsetX, OffsetY. Последовательность 1, 0, 0, 1, 0, 0 соответствует тождественной матрице (то есть отсутствию преобразования), поэтому задание преобразования `TranslateTransform` в виде `MatrixTransform`, по существу, означает, что мы начинаем с тождественной матрицы, а потом используем величины `OffsetX` и `OffsetY` в качестве свойств `X` и `Y` преобразования `TranslateTransform`. Масштабирование можно выразить, интерпретируя первое и четвертое значения как свойства `ScaleX` и `ScaleY` соответственно, а остальные, оставив такими, как в тождественной матрице. Поворот и наклон записываются несколько сложнее, потому что необходимо привлекать функции `sin` и `cos` и углы, выраженные в радианах.

Если вы уверенно владеете матричной нотацией, то сможете сэкономить время при написании XAML-кода вручную, поскольку такая запись короче (но и менее понятна).

Комбинирование преобразований

- Класс TransformGroup также наследует классу Transform (и потому может использоваться всюду, где разрешено применять описанные выше классы), а его задача -скомбинировать несколько дочерних объектов типа Transform.
 - В процедурном коде объекты отдельных преобразований добавляются в коллекцию Children, в XAML это делается следующим образом:

```
<Button>
<Button.RenderTransform>
<TransformGroup>
    <RotateTransform Angle="45"/>
    <ScaleTransform ScaleX="5" ScaleY="1"/>
    <SkewTransform AngleX="30"/>
</TransformGroup>
</Button.RenderTransform>
OK
</Button>
```



Комбинирование преобразований

- Для повышения производительности WPF сначала вычисляет комбинированное преобразование на основе потомков объекта TransformGroup, а затем применяет одно результирующее преобразование (как если бы с самого начала применялось преобразование MatrixTransform).
 - Отметим, что в составе группы TransformGroup может несколько раз встречаться одно и то же преобразование.
 - Например, два поворота MatrixTransform на 45° , эквивалентные одному повороту на 90°

ПРЕДУПРЕЖДЕНИЕ

Не все элементы типа FrameworkElement поддерживают преобразования!

Элементы, содержимое которых не является «родным» для WPF, не поддерживают преобразования, хотя и наследуют свойства LayoutTransform и RenderTransform. Например, к их числу относится элемент HwndHost, выступающий в роли владельца GDI-содержимого (обсуждается в главе 19 «Интероперабельность с другими технологиями»). Элемент управления Frame, который, в принципе, может содержать HTML-разметку (описывается в главе 9 «Однодетные элементы управления»), поддерживает преобразования в полном объеме, только если в нем нет HTML. В противном случае преобразование ScaleTransform можно применять для изменения размера, но содержимое при этом не масштабируется.

На рис. 4.15 показано, что происходит, когда на панели StackPanel размещено несколько кнопок и фрейм Frame, содержащий веб-страницу (с ограничением размера 100x100). Когда вся панель поворачивается и масштабируется, фрейм честно пытается выполнить масштабирование, но не поворачивается. В результате большая часть повернутых кнопок оказывается перекрыта.



Панель StackPanel до преобразования



StackPanel после масштабирования и поворота