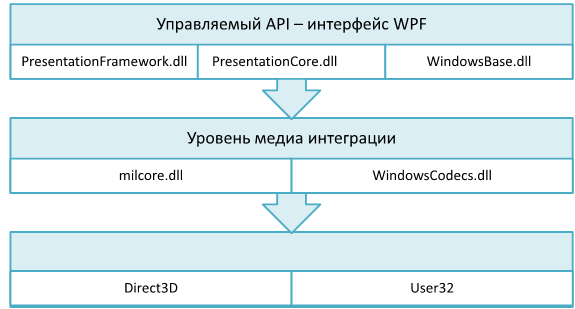
# Intro

* Windows Forms –для визуализации используется User32.dll и GDI/GDI+
* •WPF для визуализации используется DirectX (аппаратная поддержка). В случае со старыми видеокартами используется программное вычисление эффектов.
* Интерфейс в стандартных Windows приложениях не допускает масштабирования. На мониторе с высоким разрешением,размер окна уменьшается и становится менее удобным для работы. (Библиотека User32 не поддерживает полного масштабирования)
* •WPF визуализирует все элементы пользовательского интерфейса самостоятельно, и если кнопка при проектировании в ширину была пять сантиметров, то при выполнении приложения на каком либо разрешении кнопка останется шириной в пять сантиметров. В основу масштабирования ставится системный параметр DPI (DotsPerInch)
* Аппаратно-независимые единицы
* Аппаратно-независимая единица (device-independentunits) составляет 1/96 дюймаи используется для масштабирования окна и его элементов.
* ***[Размер одной единицы] = 1 / 96 дюйма \* [DPI системы]***
* Если создать кнопку с размером 96\*96 то при стандартных настройках системных DPI (96 dotsperinch) кнопка в ширину и в высоту будет 1 дюйм (количество пикселей в высоту и ширину ***96***).
* Если DPI системы 120 (дисплей с высоким разрешением)
* ***[Размер одной единицы] = 1 / 96 дюйма \* 120 = 1.24***
* Кнопка в ширину и в высоту будет 1 дюйм (количество пикселей в высоту и ширину ***119***)



System.Threading.DispatcherObject

WPF приложение использует модель STA (Single-ThreadAffinity–однопоточная родственность). Это означает что владельцем всего пользовательского интерфейса является один поток, и любое обращение к элементам управления, созданных в этом потоке, считается не безопасным. DispatcherObject используется для того,что бы координировать обработку сообщений и реализовывать межпоточноевзаимодействие объектов.

System.Windows.DependencyObject

В WPF главный способ взаимодействия с элементами на экране осуществляется черещсвойства. В WPF используется более мощная модель свойств, которая обладает средствами для уведомления об изменениях, наследовании значений по умолчанию, более экономичное хранение свойств. Конечный результат –это свойство зависимостей (dependencyproperty). Классы WPF получают поддержку свойств зависимости за счет порождения от DependencyObject

System.Windows.Media.Visual

Каждый элемент, который отображается в WPF, является производным от класса Visual. Этот класс описывает основную логику для визуализации объекта на окне. Также класс устанавливает связь между управляемыми библиотеками WPF и milcore.dll, которая осуществляет рендеринг.

System.Windows.UIElement

Добавляет поддержку компоновки, ввода, фокуса, событий и команд для объектов.

System.Windows.FrameworkElement

Класс FrameworkElement реализует некоторые члены, которые просто определены в UIElement. Например, UIElementустанавливает фундамент для системы компоновки WPF, но FrameworkElement включает некоторые свойства (такие как HorizontallAlignment и Margin),которые поддерживают его.

System.Windows.Shapes.Shape

От этого класса наследуются базовые фигуры, такие как Rectangle, Polygon, Ellipse, Line, Path.

System.Windows.Controls.Control

Элемент управления –это компонент, который может взаимодействовать с пользователем. Данный класс вводит дополнительные свойства для управлением внешним видом контролов(изменение шрифта, фона и т.д.) Добавляет возможность поддержки шаблонов для быстрого задания стилей элементам управления.

System.Windows.Controls.ContentControl

Это базовый класс для всех элементов управления, которые содержат единственную порцию содержимого. Примеры некоторых производных классов: Label, Button, Window и т.д.

System.Windows.Controls.ItemControl

Класс описывает все элементы управления, которые содержат в себе коллекцию элементов (TreeView, ListBox, ComboBox)

System.Windows.Controls.Panel

Это базовый класс для всех элементов компоновки –объекты способные хранит один и более дочерних элементов.

**XAML**(англ.*e****X****tensible****A****pplication****M****arkup****L****anguage*—расширяемыйязык разметкиприложений произносится (*зэмл)*)—основанный наXMLязык разметки для декларативного программирования приложений, разработанныйMicrosoft.

* + •Каждый элемент в документе ХAML отображается как экземпляр класса .NET. Имя элемента в точности соответствует имени класса.Например элемент <Button> сообщает WPFчто должен быть создан объект Button.
  + •Как и любой XML-документ код XAML допускает вложение одного элемента внутрь другого. Если вы видите элемент Button внутри элемента Grid , то пользовательский интерфейс включает Grid, содержащий внутри себя Button.
  + •Свойства каждого класса можно устанавливать через атрибуты. Тем не менее, в некоторых ситуациях атрибуты не достаточно мощны, чтобы справится с этой работой. В этих случаях понадобятся вложенные дескрипторы со специальным синтаксисом.

Загрузка и компиляция XAML

В общем случае существует три разных стиля кодирования которые могут применятся при создании приложения WPF:

* + - •Только код. Это традиционный подход ,используемы в Visual Studio для приложений Windows Forms. Пользовательский интерфейс в нем генерируется операторами кода.
    - •Код и компилированная разметка (BAML).Это предпочтительный подход для WPF, поддерживаемый Visual Studio.Для каждого окна создается шаблон XAML, и этот код XAML компилируется в BAML,после чего встраивается в конечную сборку. Во время выполнения скомпилированный BAML извлекается и используется для регенерации пользовательского интерфейса.

# Компоновка

* •Окно в WPF может содержать только один элемент.
* •Размеры элементов не должны быть заданы явно.
* •Элементы не отражают свое положение с помощью экранных координат.
* •Контейнеры компоновки разделяют доступное пространство между своими дочерними элементами.
* •Контейнеры компоновки допускают вложения.

Процесс компоновки делится на две стадии:

* + •Измерение -контейнер компоновки просматривает свои дочерние элементы и запрашивает у них предпочтительный размер.
  + •Упорядочивание –контейнер компоновки помещает элементы управления в соответствующие позиции.
* **Grid** – выстраивает элементы в строки и колонки невидимой таблицы. Это один из наиболее гибких и широко используемых контейнеров компоновки.
*  **StackPanel** –размещает элементы в горизонтальном или вертикальном стеке. Этот контейнер компоновки обычно используется в небольших разделах крупного и более сложного окна.
*  **DockPanel** – выстраивает элементы по краю контейнера.
*  **WrapPanel** – размещает элементы в последовательностях строк с переносом. В горизонтальной ориентации WrapPanel располагает элементы в строке слева направо, затем переходит к следующей строке. В вертикальной ориентации WrapPanel располагает элементы сверху вниз, используя дополнительные колонки для дополнения оставшихся элементов .
*  **UniformGrid** – помещает элементы в невидимую таблицу, устанавливая одинаковый размер для всех ячеек. Данный контейнер компоновки используется нечасто.
*  **Canvas** – позволяет элементам позиционироватся по фиксированным координатам. Это неподходящий выбор для окон переменного размера, если только вы не собираететсь взвалить на свои плечи значительный объем работы.
* **InkCanvas** –в WPF также имеется элемент InkCanvas, главное предназначение которого заключается в обеспечении перьевого ввода, а также считывания жестов пользователя.
*  **Z-Порядок** – при наличии более одного перекрывающегося элемента с помощью присоединенного свойства Canvas.Zindex можно управлять их расположением.Обычно все добавляемые элементы имеют одинаковый Zindex-0. Элементы с одинаковым Zindex отображаются в том порядке, в каком они представлены в коллекции Canvas.Children. Который основан на порядке их определения в разметке XAML**.**
*  Вложение контейнеров компоновки. Панели StackPanel, WrapPanel и DockPanel редко используются сами по себе. Вместо этого они применяются для формирования частей интерфейса. Например панель DockPanel можно использовать для размещения разных контейнеров StackPanel и WrapPanel в соответствующих областях окна.

# Биндинг

    Binding binding = new Binding();

    binding.ElementName = "myTextBox"; // элемент-источник

    binding.Path = new PropertyPath("Text"); // свойство элемента-источника

    myTextBlock.SetBinding(TextBlock.TextProperty, binding); // установка привязки для элемента-приемника

# Application

Во время выполнения каждое приложение WPFпредставлено экземпляром класса System.Windows.Application. Этот класс отслеживает все открытые окна в приложении, решает, когда приложение должно быть остановлено, и инициирует события приложения, которые можно обрабатывать для выполнения инициализации и очистки.

В WPF приложение проходит через простой жизненный цикл.

Вскоре после запуска приложения создается объект Application.Во время его выполнения возникают различные события приложения, которые можно отслеживать. И, когда объект приложения освобождается, приложение завершается.

События класса Application

* •Start
* •Exit
* •SessionEnding
* •Activated
* •Deactivated
* •DispatcherUnhandledException

Single Thread Apartment

Однопоточные апартаменты –модель исполнения которая поддерживается в WPF.

* •Элементы WPF обладают *потоковым родством (thread affinity)*. Поток, который создал элементы, владеет ими, а другие потоки не могут с ними взаимодействовать непосредственно.
* •Объекты WPF,которые обладают потоковым родством, порождены от DispatcherObject
* Dispatcher
* Диспетчерэто экземпляр класса System.Windows.Threading.Dispatcher он
* управляет работами, которые возникают во время выполнения WPF приложения. Диспетчер владеет потоком приложения и управляет очередью задач.
*  Какие события класса Application Вы знаете? Опишите их.
*  Где находится метод Main в WPF приложении сгенерированном VisualStudio?
*  Как создать многопоточное приложение в WPF?
*  С помощью чего можно реализовать межпоточное взаимодействие в приложении?

# Content

*  Класс ContentControl добавляет свойство Content, которое поддерживает любой тип объекта в качестве содержимого. Все типы, которые можно присвоить свойству Content можно разбить на две группы:
  + o Типы не производные от UIElement. Объект, установленный в качестве содержимого, будет отображаться как строка, которую возвращает метод ToString();
  + o Типы производные от UIElement**.** Объект, установленный в качестве содержимого, будет визуализироваться посредством метода UIElement. OnRender() ;
*  **Label**. Класс Label является простейшим элементом управления содержимым. Как и любой другой элемент управления содержимым он принимает одиночную порцию содержимого, которая размещается внутри него. Отличительной чертой элемента Label является его поддержка мнемонических команд―нажатий клавиш, которые передают фокус соответствующему элементу управления.
*  **ButtonBase**. Класс ButtonBase содержит лишь несколько членов. Он определяет событие Click и добавляет поддержку команд, которые позволяют подключать кнопки к высокоуровневым задачам приложений.
*  **ToolTip**. Представляет элемент управления, который создает всплывающее окно, отображающее сведения для элемента в интерфейсе. Например, можно использовать объект ToolTip для предоставления имени Button или ToolBar в объекте ToolBarTray. Содержимое элемента управления ToolTip может меняться от простой текстовой строки до более сложного содержимого, такого как объект StackPanel, который содержит встроенный текст и изображения. Содержимое объекта ToolTip не может получать фокус.
*  **ScrollViewer**. Класс ScrollViewer нужен для того чтобы добавить полосу прокрутки в окно, которая необходима при размещении большого объема содержимого в ограниченную область.
*  **UserControl**. Если необходимо создать новый элемент управления, то самый простой способ создать класс, производный от UserControl.
*  **Window**. Предоставляет возможность создания, настройки отображения и управления временем существования окон и диалоговых окон.
*  **HeaderContentControl**. HeaderContentControl это просто контейнер с содержимым и заголовком. От класса HeaderContentControl порождены классы GroupBox, TabItem и Expander.
*  **GroupBox**. Класс GroupBox используется для группировки небольшого количества взаимосвязанных элементов управления, таких как кнопки переключателя. В классе GroupBox нет никаких встроенных функций, поэтому его можно применять где угодно.
*  **TabItem**. Объекты TabItem представляют собой страницы в элементе TabControl. Свойство IsSelected указывает, видима ли данная вкладка в TabControl.
*  **Expander**. Класс Expander содержит область содержимого, которую пользователь может показать или скрыть, щелкнув на кнопочке со стрелкой.
*  **Свойство Margin** описывает расстояние от границы дочернего элемента до границы его родительского элемента.
*  **Свойство Padding** описывает расстояние от границы элемента до контента находящегося в нем.
*  **Border** рисует границу, фон или и то и другое вокруг другого элемента.

**Закрепление материала**

*  Какие элементы управления наследуются от класса ContentControl? Опишите их.
*  Какие типы ContentControl может поддерживать в качестве содержимого?
*  Для чего нужен декоратор Border?
*  С помощью чего нужно выравнивать содержимое?

# **WPF Interview questions with answers**

## **What is the need of WPF when we had windows forms?**

A - Anywhere execution ( Windows or Web)

B - Bindings ( less coding)

C - Common look and feel ( resource and styles)

D - Declarative programming (XAML)

E - Expression blend animation ( Animation ease)

F - Fast execution ( Hardware acceleration)

G - Graphic hardware independent ( resolution independent)

## **What is XAML in WPF and why do we need it?**

XAML is a XML file which represents your WPF UI. The whole point of creating the UI representation in XML was write once and run it anywhere. So the same XAML UI can be rendered as windows application with WPF and the same UI can be displayed on the browser using WPF browser or Silverlight application.

## **What is xmlns in XAML file?**

“xmlns” stands for XML namespaces. It helps us to avoid name conflicts and confusion in XML documents. For example consider the below two XML which have table elements, one table is a HTML table and the other represents a restaurant table. Now if both these elements come in a single XML document there would name conflicts and confusion.

## **What is the difference between xmlns and xmlns:x in WPF ?**

Bothe namespaces helps to define / resolved XAML UI elements.

The first namespace is the default namespace and helps to resolve overall WPF elements.

Hide   Copy Code

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

The second namespace is prefixed by “x:” and helps to resolve XAML language definition.

Hide   Copy Code

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

For instance for the below XAML snippet , we have two things one is the “StackPanel” and the other is “x:name”. “StackPanel” is resolved by the default namespace and the “x:name” is resolved by using “xmlns:x” namespace.

Hide   Copy Code

<StackPanel x:Name="myStack" />

## **Provide some instances where you have “xmlns:x” namespace in XAML ?**

There are two common scenarios where we use “xmlns:x” namespace :-

To define behind code for the XAML file using “x:class” attribute.

Hide   Copy Code

<Page

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

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

x:Class="MyNamespace.MyCanvasCodeInline"

>

Second to provide name to an element.

Hide   Copy Code

<StackPanel x:Name="myStack" />

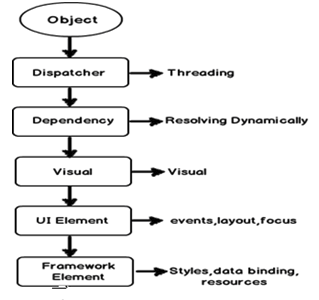
## **What are the different kinds of controls in WPF?**

WPF controls can be categorized in to four categories:-

* **Control: -**This is the basic control with which you will work most of time. For example textbox, buttons etc. Now controls which are standalone control like button , text box , labels etc are termed as content control. Now there are other controls which can hold other controls, for instance itemscontrols. Itemscontrol can have multiple textbox controls, label controls etc.
* **Shape: -** These controls help us to create simple graphic controls like Ellipse, line, rectangle etc.
* **Panel: -** These controls help to align and position the controls. For instance grid helps us to align in a table manner, stack panel helps for horizontal and vertical alignment.
* **Content presenter: -** This control helps to place any XAML content inside it. Used when we want to add dynamic controls on a WPF screen.

All the above four types of WPF controls finally inherit from the frameworkelement class of WPF.

## **Can you explain the complete WPF object hierarchy?**



**Object: -** As WPF is created using .NET so the first class from which WPF UI classes inherits is the .NET object class.

**Dispatcher: -** This class ensures that all WPF UI objects can be accessed directly only by the thread who own him. Other threads who do not own him have to go via the dispatcher object.

**Dependency: -**WPF UI elements are represented by using XAML which is XML format. At any given moment of time a WPF element is surrounded by other WPF elements and the surrounded elements can influence this element and this is possible because of this dependency class. For example if a textbox surrounded by a panel, its very much possible that the panel background color can be inherited by the textbox.

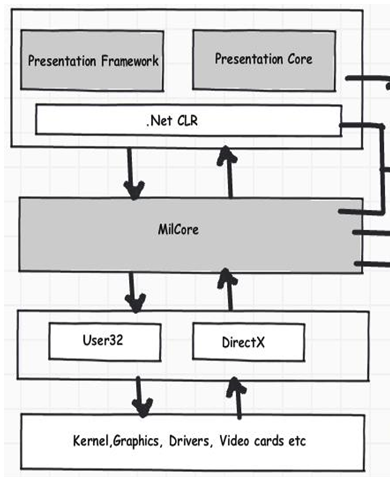
**Visual: -** This is the class which helps WPF UI to have their visual representation.

**UI Element: -** This class helps to implement features like events, input, layouting etc.

**Framework element: -** This class supports for templating , styles , binding , resources etc.

And finally all WPF controls textbox , button , grids and whatever you can think about from the WPF tool box inherits from the framework element class.

## **Can you explain the overall architecture of WPF?**



## **What is App.xaml in WPF project?**

App.xaml is the start up file or a boot strapper file which triggers your first XAML page from your WPF project.

## **What are resources in WPF?**

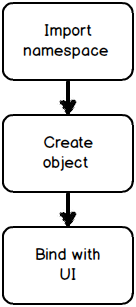
Resources are objects referred in WPF XAML. In C# code when we create an object we do the following three steps :-

Hide   Copy Code

using CustomerNameSpace; // import the namespace.

Customer obj = new Customer(); // Create object of the class

Textbox1.text = obj.CustomerCode; // Bind the object with UI elements



So even in WPF XAML to define resources which are nothing but objects we need to the above 3 steps :-

* Import namespace where the class resides: - To define namespace we need to use the “xmlns” attribute as shown in the below XAML code.

Hide   Copy Code

<Window x:Class="LearnWpfResources.MainWindow"

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

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

xmlns:custns="clr-namespace:LearnWpfResources"

Title="MainWindow" Height="350" Width="525">

* Create object of the class :- To create an object of the class in XAML we need to create a resource by using the resource tag as the below code. You can the object name is ‘custobj”.

Hide   Copy Code

<Window.Resources>

<custns:Customer x:Key="custobj"/>

</Window.Resources>

The above code you can map to something like this in C#

Hide   Copy Code

Customer custobj = new Customer();

* Bind the object with UI objects :- Once the object is created we can then bind them using bindings like one way , two way as explained in “Explain one way, two way, one time and one way to source?” question explained above.

Hide   Copy Code

<TextBox Text="{Binding CustomerCode, Mode=TwoWay, Source={StaticResource custobj}}" />

## **Explain the difference between static and dynamicresource?**

Resources can be referred statically or dynamically. Static referred resources evaluate the resource only once and after that if the resources change those changes are not reflected in the binding. While dynamic referred resources are evaluated every time the resource is needed.

Consider the below “SolidColorBrush” resource which is set to “LightBlue” color.

Hide   Copy Code

<Window.Resources>

<SolidColorBrush Color="LightBlue" x:Key="buttonBackground" />

</Window.Resources>

The above resource is binded with the below two textboxes statically and dynamically respectively.

Hide   Copy Code

<TextBox Background="{DynamicResource buttonBackground}" />

<textbox background="{StaticResource buttonBackground}">

Now if we modify the resource , you see the first text box changes the background and the other textbox color stays as it is.

Hide   Copy Code

private void Button\_Click(object sender, RoutedEventArgs e)

{

Resources["buttonBackground"] = Brushes.Black;

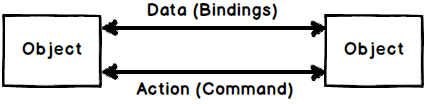
}

## **When should we use static resource over dynamic resource ?**

Dynamic resources reduce application performance because they are evaluated every time the resource is needed. So the best practice is use Static resource until there is a specific reason to use dynamic resource. If you want resource to be evaluated again and again then only use dynamic resource.

## **Explain the need of binding and commands?**

WPF Binding’s helps to send / receive data between WPF objects while command helps to send and receive actions.The object that emits data or action is termed as **source** and the object who wants to receive data or action is termed as **target**.



## **Explain one way, two way, one time and one way to source?**

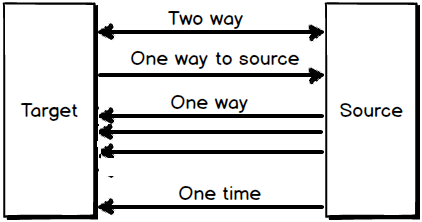
All the above 4 things define how data will flow between target and source objects when WPF binding is applied.

**Two way: -** Data can flow from both source to target and from target to source.

**One way: -** Data flows only from source to target.

**One way to source: -** Data flows only from target to source.

**One time: -** Data flows only for the first time from source to target and after that no communication happens.



## **Can you explain WPF command with an example?**