1. Перечислите преимущества и недостатки WPF?

Технология WPF (Windows Presentation Foundation) является часть экосистемы платформы .NET и представляет собой подсистему для построения графических интерфейсов.

Если при создании традиционных приложений на основе WinForms за отрисовку элементов управления и графики отвечали такие части ОС Windows, как User32 и GDI+, то приложения WPF основаны на DirectX. В этом состоит ключевая особенность рендеринга графики в WPF: используя WPF, значительная часть работы по отрисовке графики, как простейших кнопочек, так и сложных 3D-моделей, ложиться на графический процессор на видеокарте, что также позволяет воспользоваться аппаратным ускорением графики.

Одной из важных особенностей является использование языка декларативной разметки интерфейса XAML, основанного на XML: вы можете создавать насыщенный графический интерфейс, используя или декларативное объявление интерфейса, или код на управляемых языках C#, VB.NET и F#, либо совмещать и то, и другое.

Первая версия - WPF 3.0 вышла вместе с .NET Framework 3.0 и операционной системой Windows Vista в 2006 году. И с тех пор платформа WPF является частью экосистемы .NET и развивается вместе с фреймворком .NET. Например, на сегодняшний день последней версией фреймворка .NET является .NET 7, и WPF полностью поддерживается этой версией фреймворка.

Что вам, как разработчику, предлагает WPF?

Использование традиционных языков .NET-платформы - C#, F# и VB.NET для создания логики приложения

Возможность декларативного определения графического интерфейса с помощью специального языка разметки XAML, основанном на xml и представляющем альтернативу программному созданию графики и элементов управления, а также возможность комбинировать XAML и C#/VB.NET

Независимость от разрешения экрана: поскольку в WPF все элементы измеряются в независимых от устройства единицах, приложения на WPF легко масштабируются под разные экраны с разным разрешением.

Новые возможности, которых сложно было достичь в WinForms, например, создание трехмерных моделей, привязка данных, использование таких элементов, как стили, шаблоны, темы и др.

Хорошее взаимодействие с WinForms, благодаря чему, например, в приложениях WPF можно использовать традиционные элементы управления из WinForms.

Богатые возможности по созданию различных приложений: это и мультимедиа, и двухмерная и трехмерная графика, и богатый набор встроенных элементов управления, а также возможность самим создавать новые элементы, создание анимаций, привязка данных, стили, шаблоны, темы и многое другое

Аппаратное ускорение графики - вне зависимости от того, работаете ли вы с 2D или 3D, графикой или текстом, все компоненты приложения транслируются в объекты, понятные Direct3D, и затем визуализируются с помощью процессора на видеокарте, что повышает производительность, делает графику более плавной.

Создание приложений под множество ОС семейства Windows

В тоже время WPF имеет определенные ограничения. Несмотря на поддержку трехмерной визуализации, для создания приложений с большим количеством трехмерных изображений, прежде всего игр, лучше использовать другие средства - DirectX или специальные фреймворки, такие как Monogame или Unity.

Также стоит учитывать, что по сравнению с приложениями на Windows Forms объем программ на WPF и потребление ими памяти в процессе работы в среднем несколько выше. Но это с лихвой компенсируется более широкими графическими возможностями и провышенной производительностью при отрисовке графики.

Кроме того, несмотря на то, что WPF работает поверх кроссплатформенной среды .NET 5/6/7, но в силу природы WPF и зависимости от компонентов Windows, на данный момент создавать приложения на WPF можно только под ОС Windows.

1. Зачем нужен язык XAML? Каким образом он используется в WPF?

XAML (eXtensible Application Markup Language) - язык разметки, используемый для инициализации объектов в технологиях на платформе .NET. Применительно к WPF (а также к Silverlight) данный язык используется прежде всего для создания пользовательского интерфейса декларативным путем. Хотя функциональность XAML только графическими интерфейсами не ограничивается: данный язык также используется в технологиях WCF и WF, где он никак не связан с графическим интерфейсом. То есть его область шире. Применительно к WPF мы будем говорить о нем чаще всего именно как о языке разметки, который позволяет создавать декларативным путем интерфейс, наподобие HTML в веб-программировании. Однако опять же повторюсь, сводить XAML к одному интерфейсу было бы неправильно, и далее на примерах мы это увидим.

XAML - не является обязательной частью приложения, мы вобще можем обходиться без него, создавая все элементы в файле связанного с ним кода на языке C#. Однако использование XAML все-таки несет некоторые преимущества:

Возможность отделить графический интерфейс от логики приложения, благодаря чему над разными частями приложения могут относительно автономно работать разные специалисты: над интерфейсом - дизайнеры, над кодом логики - программисты.

Компактность, понятность, код на XAML относительно легко поддерживать.

При компиляции приложения в Visual Studio код в xaml-файлах также компилируется в бинарное представление кода xaml, которое называется BAML (Binary Application Markup Language). И затем код baml встраивается в финальную сборку приложения - exe или dll-файл.

1. Какие бывают контейнеры компоновки?

Компоновка (layout) представляет собой процесс размещения элементов внутри контейнера. Возможно, вы обращали внимание, что одни программы и веб-сайты на разных экранах с разным разрешением выглядят по-разному: где-то лучше, где-то хуже. В большинстве своем такие программы используют жестко закодированные в коде размеры элементов управления. WPF уходит от такого подхода в пользу так называемого "резинового дизайна", где весь процесс позиционирования элементов осуществляется с помощью компоновки.

В WPF компоновка осуществляется при помощи специальных контейнеров. Фреймворк предоставляет нам следующие контейнеры: ***Grid, UniformGrid, StackPanel, WrapPanel, DockPanel и Canvas.***

В WPF при компоновке и расположении элементов внутри окна нам надо придерживаться следующих принципов:

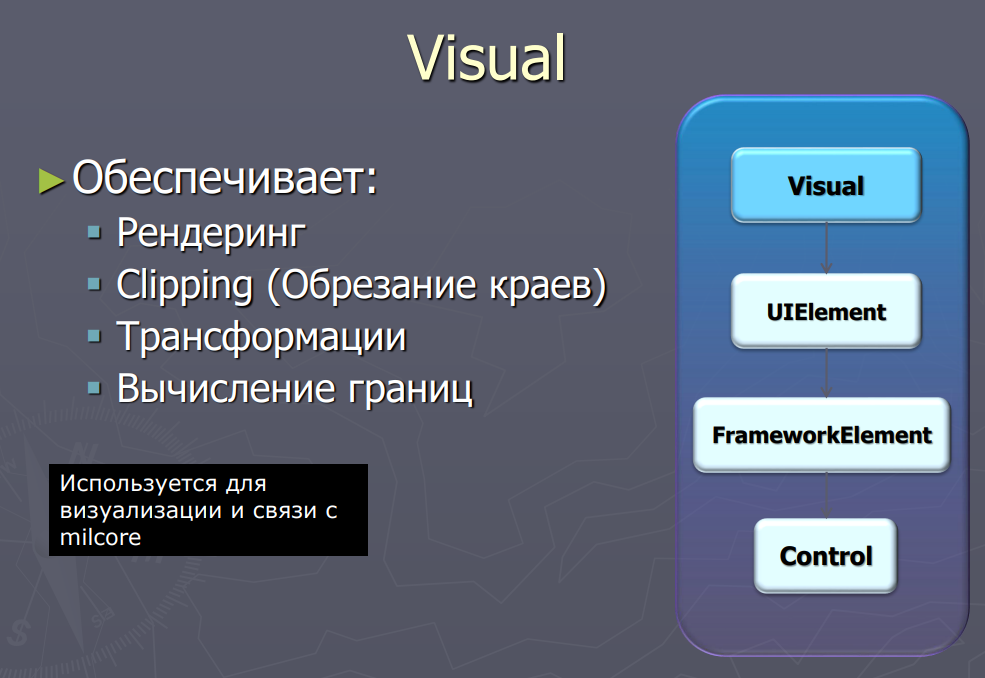
Нежелательно указывать явные размеры элементов (за исключеним минимальных и максимальных размеров). Размеры должны определяться контейнерами.

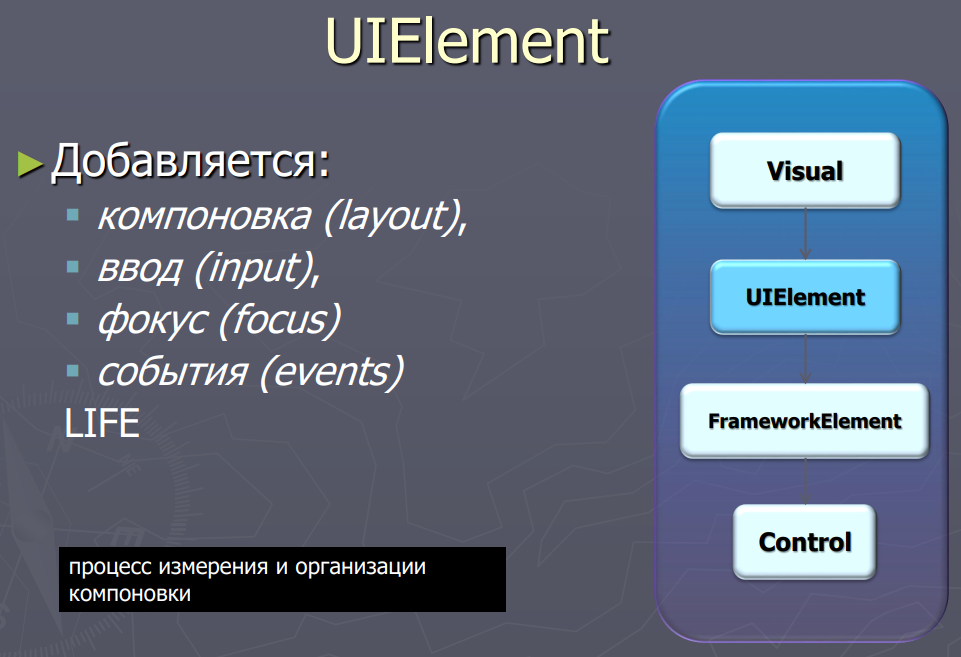
Нежелательно указывать явные позицию и координаты элементов внутри окна. Позиционирование элементов всецело должно быть прерогативой контейнеров. И контейнер сам должен определять, как элемент будет располагаться. Если нам надо создать сложную систему компоновки, то мы можем вкладывать один контейнер в другой, чтобы добиться максимально удобного расположения элементов управления.

1. Перечислите основные группы элементов управления. Расскажите про объектную модель WPF.

Все элементы управления могут быть условно разделены на несколько подгрупп:

* Элементы управления содержимым, например кнопки (Button), метки (Label)
* Специальные контейнеры, которые содержат другие элементы, но в отличие от элементов Grid или Canvas не являются контейнерами компоновки - ScrollViewer,GroupBox
* Декораторы, чье предназначение создание определенного фона вокруг вложенных элементов, например, Border или Viewbox.
* Элементы управления списками, например, ListBox, ComboBox.
* Текстовые элементы управления, например, TextBox, RichTextBox.
* Элементы, основанные на диапазонах значений, например, ProgressBar, Slider.
* Элементы для работ с датами, например, DatePicker и Calendar.
* Остальные элементы управления, которые не вошли в предыдущие подгруппы, например, Image.



1. Объясните назначение класса DependencyProperty. Зачем нужны свойства зависимостей? Как создать новой свойство зависимости и в каких случаях это необходимо?

Класс DependencyProperty в WPF используется для создания свойств зависимостей, которые позволяют связывать значения свойств объектов с другими свойствами или данными. Свойства зависимостей используются, когда требуется уведомлять объекты об изменениях в связанных свойствах, а также при реализации анимации, шаблонов и стилей. Чтобы создать новое свойство зависимости, нужно определить класс, который будет хранить это свойство, задать имя свойства, тип значения и реализовать методы доступа к нему. Это может быть необходимо, например, когда требуется создать новый элемент управления с уникальными свойствами.

1. Каким образом осуществляется обработка событий в WPF? Что означает концепция маршрутизированных событий? Перечислите основные группы событий.

В WPF обработка событий осуществляется с помощью механизма маршрутизации событий. Это означает, что событие, возникающее в элементе управления, может быть обработано не только самим элементом, но и его родительскими элементами вверх по дереву визуального дерева или даже другими элементами вниз по дереву. Маршрутизация событий позволяет создавать более гибкие и расширяемые приложения. Основные группы событий в WPF: ***маршрутизированные события, туннелируемые события и всплывающие события.***

1. Что такое Resource Dictionary?

Resource Dictionary в WPF - это коллекция ресурсов, которые могут быть использованы в различных частях приложения. Ресурсы могут включать в себя стили, шаблоны, изображения, объекты, строковые ресурсы и т.д. Resource Dictionary может быть создан как отдельный файл XAML или встроен в другой файл XAML.

1. Что такое стиль и как его создать? В чем преимущество использования стиля?

Стиль в WPF - это набор свойств, которые могут быть применены к элементу управления или группе элементов управления, чтобы изменить их внешний вид или поведение. Стиль может содержать свойства для изменения цвета, шрифта, размера и расположения элементов управления. Для создания стиля нужно определить его в ресурсах приложения, задать имя стиля и определить набор свойств для изменения внешнего вида элементов управления. Преимущества использования стиля в WPF включают повторное использование кода и легкость изменения внешнего вида элементов управления.

1. Что такое Command? Расскажите паттерн Command. Как в WPF используется Command и для чего?

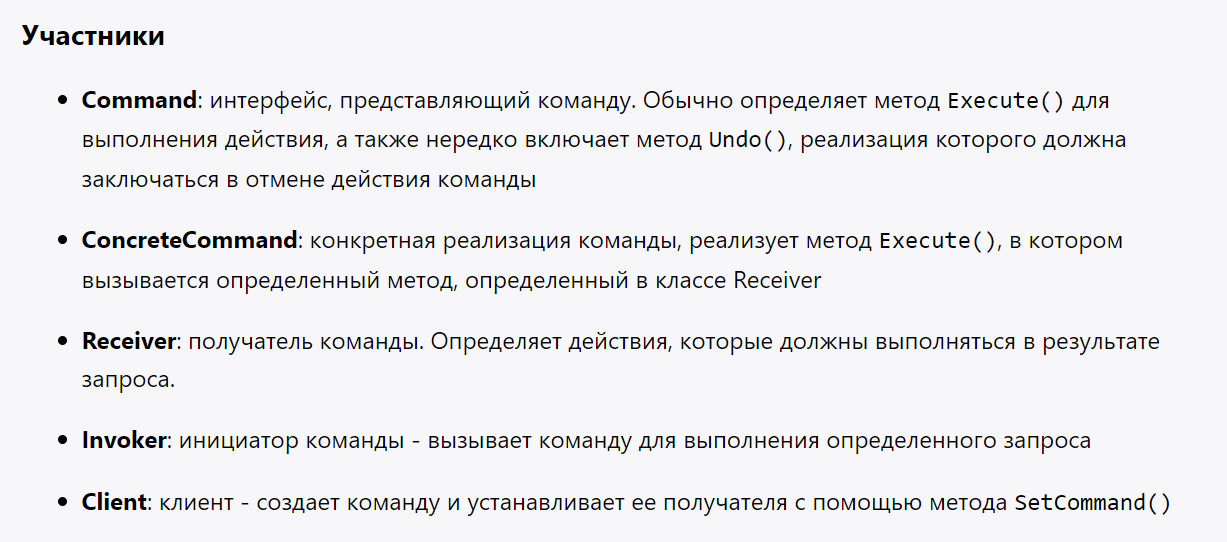
Команды представляют механизм выполнения какой-нибудь задачи, например, копирования текста - когда мы нажимаем Ctrl+C, то мы копируем текст в буффер. В процессе копирования выполняется ряд действий, и все вместе эти действия объединяются в одну команду. Использование команд помогает нам сократить объем кода и использовать одну и ту же команду для нескольких элементов управления в различных местах программы. Таким образом, команды позволяют абстрагировать набор действий от конкретных событий конкретных элементов.

Модель команд в WPF состоит из четырех аспектов:

* Сама команда, которая представляем выполняемую задачу
* Привязка команд, которая связывает команду с определенной логикой приложения
* Источник команды - элемент пользовательского интерфейса, который запускает команду (например, кнопка, по нажатию который выполняется команда)
* Цель команды - элемент интерфейса, на котором выполняется команда

Используется паттерн command:

* Когда надо передавать в качестве параметров определенные действия, вызываемые в ответ на другие действия. То есть когда необходимы функции обратного действия в ответ на определенные действия.
* Когда необходимо обеспечить выполнение очереди запросов, а также их возможную отмену.
* Когда надо поддерживать логгирование изменений в результате запросов. Использование логов может помочь восстановить состояние системы - для этого необходимо будет использовать последовательность запротоколированных команд.



В WPF есть своя реализация команд!