1. Перечислите преимущества и недостатки WPF?
2. **огатая модель рисования**
3. Вместо рисования пикселей в WPF вы имеете дело с примитивами — базовыми фигурами, блоками текста и прочими графическими ингредиентами. Кроме того, доступны такие новые средства, как действительно прозрачные элементы управления, возможность укладывания друг на друга множества уровней с разной степенью прозрачности, а также встроенная поддержка трехмерной графики.
4. **Развитая текстовая модель**
5. После многих лет нестандартной обработки текстов WPF наконец-то предоставляет Windows-приложениям возможность отображения расширенного стилизованного текста в любом месте пользовательского интерфейса. И если нужно отображать значительные объемы текста, для повышения читабельности можно воспользоваться развитыми средствами отображения документов, такими как переносы, разбиение на колонки и выравнивание.
6. **Анимация**
7. В WPF нет необходимости использовать таймер для того, чтобы заставить форму перерисовать себя. Вместо этого доступна анимация — неотъемлемая часть платформы. Анимация определяется декларативными дескрипторами, и WPF запускает ее в действие автоматически.
8. **Поддержка аудио и видео**
9. Прежние инструментальные наборы для построения пользовательских интерфейсов, такие как Windows Forms, были весьма ограничены в работе с мультимедиа. Однако WPF включает поддержку воспроизведения любого аудио или видеофайла, поддерживаемого проигрывателем Windows Media, позволяя воспроизводить более одного медиафайла одновременно. Что еще больше впечатляет — WPF предоставляет в ваше распоряжение инструменты для интеграции видеосодержимого в остальную часть пользовательского интерфейса, позволяя выполнять такие экзотические трюки, как размещение видеоокна на поверхности вращающегося трехмерного куба.
10. **Стили и шаблоны**
11. Стили позволяют стандартизировать форматирование и многократно использовать его по всему приложению. Шаблоны дают возможность изменить способ отображения элементов, даже таких основополагающих, как кнопки. Построение интерфейса с обложками еще никогда не было таким простым.
12. **Команды**
13. Большинству пользователей известно, что не имеет значения, откуда они инициируют команду открытия (Open) — через меню или панель инструментов; конечный результат один и тот же. Теперь эта абстракция доступна коду — можно определять команды приложения в одном месте и привязывать их к множеству элементов управления.
14. **Декларативный пользовательский интерфейс**

Минусы

1. Более слабая поддержка в дизайнере окон. Много приходится вводить руками в XAML-код.  
2. Сглаженный шрифт.  
3. Отсутствуют некоторые полезные графические возможности, которые были в GDI/GDI+. В частности, в WPF нет операций ROP2, не поддерживаются наконечники линий и нет составных линий (например нельзя нарисовать двойную линию).  
4. Трудно нарисовать собственный элемент управления со стандартным оформлением какого-либо системного контрола (в WinForms для этого были специальные классы с методами рендеринга).  
5. Нет автоматического затенения рисунков на отключенных (disabled) кнопках, приходится это делать вручную.  
6. Неудачные названия многих классов, из-за чего полная каша и путаница с названиями.  
7. Плохая документированность. Многие тривиальные вещи приходится узнавать, ища ответ в интернете.  
8. Отсутствуют такие элементы управления, как NumericUpDown и PropertyGrid.  
9. Отсутствует HelpProvider.  
10. Чувствительность к драйверам видеокарты (на старых версиях может глючить).  
11. Нормально работает только под Windows XP/Vista/7.  
12. Ну и наконец, идеология WPF в корне отличается от всего, что было раньше. Придется учиться всему заново. Причем, "немножко подучиться" не получится. Нужно изучать всё - с самого начала и до конца.

2. 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**.

3. Какие бывают контейнеры компоновки?

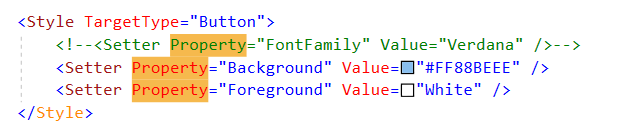
Grid, UniformGrid, StackPanel, WrapPanel, DockPanel и Canvas.

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



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

Присоединенное свойство — это свойство, которое позволяет любому объекту сообщать сведения о типе, определяющем присоединенное свойство. В WPF любой тип, наследующий от [DependencyObject](https://learn.microsoft.com/ru-ru/dotnet/api/system.windows.dependencyobject?view=windowsdesktop-7.0) него, может использовать присоединенное свойство независимо от того, наследуется ли тип от типа, определяющего свойство. Присоединенное свойство — это функция языка XAML. Чтобы задать присоединенное свойство в XAML, используйте ownerType. синтаксис propertyName .



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

В прошлой главе рассматривались различные элементы управления. Но чтобы с ними взаимодействовать, нам надо использовать модель событий. WPF в отличие от других технологий, например, от Windows Forms, предлагает новую концепцию событий - маршрутизированные события (routed events).

Для элементов управления в WPF определено большое количество событий, которые условно можно разделить на несколько групп:

События клавиатуры

События мыши

События стилуса

События сенсорного экрана/мультитач

События жизненного цикла

Определение маршрутизированных событий отличается от стандартного определения событий в языке C#. Для определения маршрутизированных событий в классе создвалось статическое поле по типу RoutedEvent:

public static RoutedEvent СобытиеEvent

Это поле, как правило, имеет суффикс Event. Затем это событие регистрируется в статическом конструкторе.

И также класс, в котором создается событие, как правило определяет объект-обертку над обычным событием. В этой обертке с помощью метода AddHandler происходит добавление обработчика для данного события, а с помощью метода RemoveHandler - удаление обработчика.

7. Что такое Resource Dictionary?

Словарь ресурсов

Мы можем определять ресурсы на уровне отдельных элементов окна, например, как ресурсы элементов Window, Grid и т.д. Однако есть еще один способ определения ресурсов, который предполагает использование словаря ресурсов.

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

Стили позволяют определить набор некоторых свойств и их значений, которые потом могут применяться к элементам в xaml. Стили хранятся в ресурсах и отделяют значения свойств элементов от пользовательского интерфейса. Также стили могут задавать некоторые аспекты поведения элементов с помощью триггеров. Аналогом стилей могут служить каскадные таблицы стилей (CSS), которые применяются в коде html на веб-страницах.

Зачем нужны стили? Стили помогают создать стилевое единообразие для определенных элементов.

Стиль создается как ресурс с помощью объекта Style, который представляет класс System.Windows.Style. И как любой другой ресурс, он обязательно должен иметь ключ. С помощью коллекции Setters определяется группа свойств, входящих в стиль. В нее входят объекты Setter, которые имеют следующие свойства:

1)Property: указывает на свойство, к которому будет применяться данный сеттер. Имеет следующий синтаксис: Property="Тип\_элемента.Свойство\_элемента". Выше в качестве типа элемента использовался Control, как общий для всех элементов. Поэтому данный стиль мы могли бы применить и к Button, и к TextBlock, и к другим элементам.

2) Value: устанавливает значение

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

Паттерн "Команда" (Command) позволяет инкапсулировать запрос на выполнение определенного действия в виде отдельного объекта. Этот объект запроса на действие и называется командой. При этом объекты, инициирующие запросы на выполнение действия, отделяются от объектов, которые выполняют это действие.

Команды могут использовать параметры, которые передают ассоциированную с командой информацию. Кроме того, команды могут ставиться в очередь и также могут быть отменены.

Паттерн "Команда" (Command) позволяет инкапсулировать запрос на выполнение определенного действия в виде отдельного объекта. Этот объект запроса на действие и называется командой. При этом объекты, инициирующие запросы на выполнение действия, отделяются от объектов, которые выполняют это действие.

Команды могут использовать параметры, которые передают ассоциированную с командой информацию. Кроме того, команды могут ставиться в очередь и также могут быть отменены.

В WPF кроме обработки событий приложение может взаимодействовать с пользователем с помощью команд. 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.

В некотором роде команды в WPF являются реализацией общераспространенного паттерна Команда.

Модель команд в WPF состоит из четырех аспектов:

1)Сама команда, которая представляем выполняемую задачу

2)Привязка команд, которая связывает команду с определенной логикой приложения

3)Источник команды - элемент пользовательского интерфейса, который запускает команду (например, кнопка, по нажатию который выполняется команда)

4)Цель команды - элемент интерфейса, на котором выполняется команда