1. **Для чего в WPF используются ресурсы? Каким образом можно определить новый ресурс и управлять им? Опишите назначение класса ResourceDictionary. Зачем каждый элемент имеет собственную коллекцию ресурсов?**

Чтобы разделить ресурсы между множеством проектов можно создать словарь ресурсов. Словарь ресурсов представляет собой просто XAML-документ, который всего лишь хранит необходимые ресурсы.

Создание словаря ресурсов

При добавлении словаря ресурсов в приложение удостоверьтесь, что свойство Build Action установлено в Page (страница); это принимается для всех XAML-файлов. В результате словарь ресурсов скомпилируется в формат BAML и обеспечит более высокую производительность. Однако вполне допустимо установить свойство Build Action словаря ресурсов в Resource (ресурс); в этом случае он будет встраиваться в сборку, но не компилироваться. Синтаксический анализ во время выполнения тогда будет проходить медленнее.

1. **Какая разница между статическими и динамическими ресурсами?**

Ресурсы могут быть статическими и динамическими. Статические ресурсы устанавливается только один раз. А динамические ресурсы могут меняться в течение работы программы.

Для установки ресурса в качестве статического используется выражение StaticResource

А для установки ресурса как динамического применяется выражение DynamicResource

1. **Что такое триггеры? Для чего в WPF используются триггеры? Назовите основные типы триггеров.**

Триггеры позволяют декларативно задать некоторые действия, которые выполняются при изменении свойств стиля. Существует три вида триггеров:

* **Триггеры свойств**: вызываются в ответ на изменения свойствами зависимостей своего значения
* **Триггеры данных**: вызываются в ответ на изменения значений любых свойств (они необязательно должны быть свойствами зависимостей)
* **Триггеры событий**: вызываются в ответ на генерацию событий
* **Мультитриггеры**: вызываются при выполнении ряда условий

**Триггеры свойств**

* Простые триггеры свойств задаются с помощью объекта **Trigger**. Они следят за значением свойств и в случае их изменения с помощью объекта **Setter** устанавливают значение других свойств.

**MultiTrigger**

* При необходимости отслеживания не одного, а сразу нескольких свойств используют объект **MultiTrigger**. Он содержит коллекцию элементов **Condition**, каждый из которых, как и обычный триггер, определяет отслеживаемое свойство и его значение

**EventTrigger**

* Если простой триггер наблюдает за изменением свойства, то EventTrigger реагирует на определенные события совсем как обработчики событий. Правда, триггеры событий более ограничены в своих возможностях.

**Триггер данных DataTrigger**

* DataTrigger отслеживает изменение свойств, которые необязательно должны представлять свойства зависимостей. Для соединения с отслеживаемыми свойства триггеры данных используют выражения привязки

1. **Что такое локализация и как ее обеспечить**.

Локализация – это перевод ресурсов приложения в локализованные версии для определенных культур, которые поддерживает приложение.

Когда вы разрабатываете свое приложение и ваше приложение доступно только на одном языке, вы ограничиваете количество своих клиентов и размер вашего бизнеса. Если вы хотите увеличить свою клиентскую базу, что также увеличит ваш бизнес, то ваш продукт должен быть доступен и доступен для глобальной аудитории. Экономичная локализация вашего продукта – один из лучших и наиболее экономичных способов охвата большего количества клиентов.

В WPF локализуемые приложения очень легко создавать с помощью файла resx, который является самым простым решением для локализации. Давайте рассмотрим простой пример, чтобы понять, как это работает.

Создайте новый проект WPF с именем WPFLocalization.

В обозревателе решений вы увидите файл Resources.resx в папке «Свойства».

1. **Что такое тема? Опишите процесс создания темы на основе ресурсов и стилей.**

Стили позволяют задать стилевые особенности для определенного элемента или элементов одного типа. Но иногда возникает необходимость применить ко всем элементам какое-то общее стилевое единообразие. И в этом случае мы можем объединять стили элементов в темы. Например, все элементы могут выполнены в светлом стиле, или, наоборот, к ним может применяться так называемая "ночная тема". Более того может возникнуть необходимость не просто определить общую тему для всех элементов, но и иметь возможность динамически выбирать понравившуюся тему из списка тем.

1. **Что такое шаблон и как его создать?**

Шаблон описывает общий вид и внешний вид элемента управления. Для каждого элемента управления есть шаблон по умолчанию, связанный с ним, который дает элементу вид его. В приложениях WPF вы можете легко создавать свои собственные шаблоны, когда вы хотите настроить визуальное поведение и внешний вид элемента управления.

Связь между логикой и шаблоном может быть достигнута путем привязки данных. Основные различия между **стилями** и **шаблонами** перечислены ниже.

* Стили могут изменять внешний вид вашего элемента управления только с помощью свойств этого элемента по умолчанию.
* С помощью шаблонов вы можете получить доступ к большему количеству частей элемента управления, чем к стилям. Вы также можете указать как существующее, так и новое поведение элемента управления.

Есть два типа шаблонов, которые наиболее часто используются –

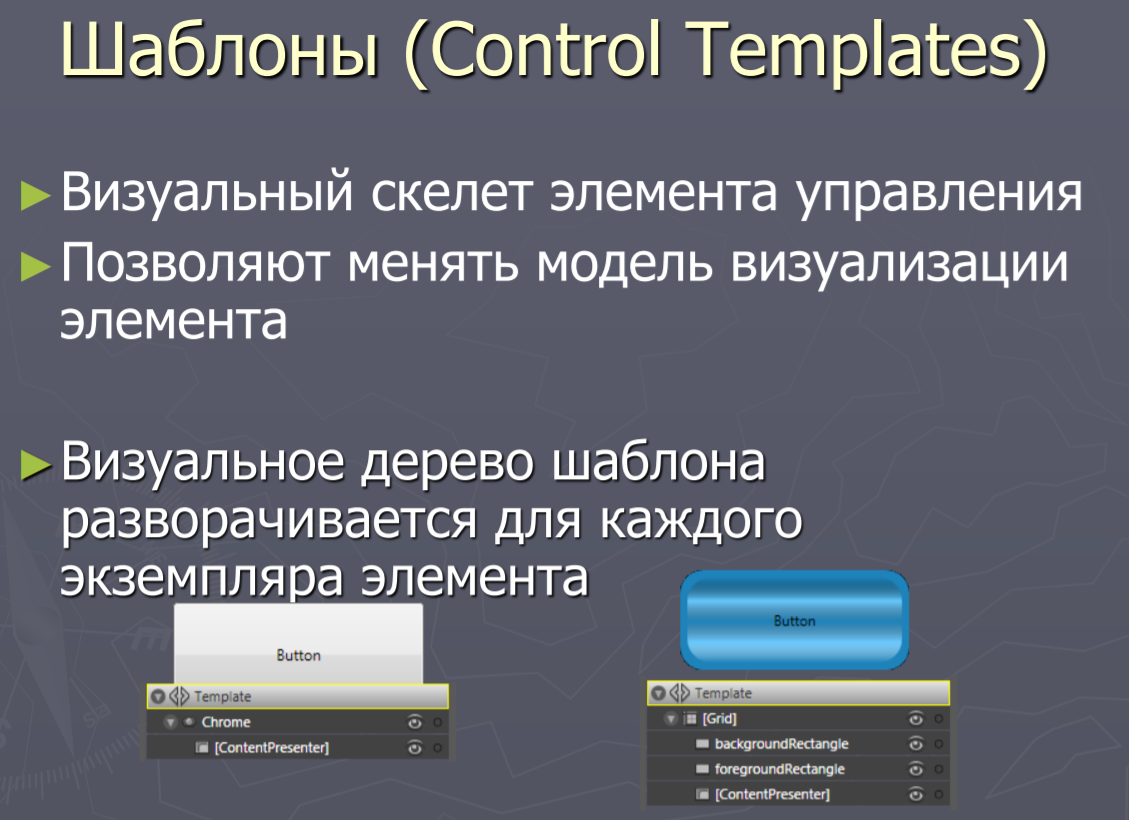
* Шаблон управления
* Шаблон данных

## Шаблон управления

Шаблон элемента управления определяет внешний вид элемента управления. Все элементы пользовательского интерфейса имеют внешний вид и поведение, например, кнопка имеет внешний вид и поведение. Событие щелчка или событие наведения мыши – это поведение, которое запускается в ответ на щелчок и наведение, а также имеется внешний вид кнопки по умолчанию, который можно изменить с помощью шаблона элемента управления.

## Шаблон данных

Шаблон данных определяет и определяет внешний вид и структуру коллекции данных. Это обеспечивает гибкость для форматирования и определения представления данных на любом элементе пользовательского интерфейса. В основном он используется в элементах управления, связанных с данными, таких как ComboBox, ListBox и т. Д.



1. **Зачем нужны пользовательские элементы управления? Как создать собственный элемент. Опишите члены класса UserControl.**

Хотя пользовательский элемент можно построить в любом проекте WPF, обычно такие элементы размещаются в специально выделенной сборке — библиотеке классов (DLL). Это позволяет разделять работу с множеством приложений WPF.

Чтобы гарантировать наличие всех необходимых ссылок на сборки и импорт всех нужных пространств имен, при создании приложения в Visual Studio в качестве типа проекта следует выбрать Custom Control Library (WPF) (Библиотека пользовательских элементов управления (WPF)). Внутри библиотеки классов можно создавать сколько угодно элементов управления.

Приложения WPF позволяют создавать пользовательские элементы управления, что позволяет легко создавать многофункциональные и настраиваемые элементы управления. Пользовательские элементы управления используются, когда все встроенные элементы управления, предоставляемые Microsoft, не соответствуют вашим критериям или вы не хотите платить за сторонние элементы управления.

## Пользовательский контроль

Пользовательские элементы управления предоставляют возможность собирать и объединять различные встроенные элементы управления и упаковывать их в повторно используемый XAML. Пользовательские элементы управления используются в следующих сценариях –

* Если элемент управления состоит из существующих элементов управления, т. Е. Вы можете создать один элемент управления из нескольких уже существующих элементов управления.
* Если элемент управления не нуждается в поддержке тем. Пользовательские элементы управления не поддерживают сложные настройки, шаблоны элементов управления и сложные стили.
* Если разработчик предпочитает писать элементы управления, используя модель с выделенным кодом, где представление, а затем прямой код для обработчиков событий.
* Вы не будете делиться своим контролем между приложениями.

UserControl это по сути своей контрол на котором вы группируете другие элементы и используете из все вместе, как пример контрол поиска состоящий из текстового поля, подписи и кнопки, что бы не копировать и не менять потом по всему проекту какую либо настройку вы создаете UserControl и уже его используете везде

1. **Что такое привязка данных? В чем разница между следующими режимами привязки: OneWay, TwoWay и OneTime**.

Привязка подразумевает взаимодействие двух объектов: источника и приемника. Объект-приемник создает привязку к определенному свойству объекта-источника. В случае модификации объекта-источника, объект-приемник также будет модифицирован.

Ключевым объектом при создании привязки является объект **System.Windows.Data.Binding**.

### Режимы привязки

Свойство **Mode** объекта Binding, которое представляет режим привязки, может принимать следующие значения:

* **OneWay**: свойство объекта-приемника изменяется после модификации свойства объекта-источника.
* **OneTime**: свойство объекта-приемника устанавливается по свойству объекта-источника только один раз. В дальнейшем изменения в источнике никак не влияют на объект-приемник.
* **TwoWay**: оба объекта - приемник и источник могут изменять привязанные свойства друг друга.

1. **Объясните назначение интерфейса INotifyPropertyChanged**

Интерфейс [INotifyPropertyChanged](https://docs.microsoft.com/ru-ru/dotnet/api/system.componentmodel.inotifypropertychanged?view=netframework-4.8) используется для уведомления клиентов, как правило, привязки клиентов, что значение свойства изменилось.