**Лабораторная №8 Ресурсы, стили, триггеры, шаблоны**

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?**

под ресурсами подразумеваются не дополнительные файлы (или **физические ресурсы**), как, например, аудиофайлы, файлы с изображениями, которые добавляются в проект. Здесь речь идет о **логических ресурсах**, которые могут представлять различные объекты - элементы управления, кисти, коллекции объектов и т.д. Логические ресурсы можно установить в коде XAML или в коде C# с помощью свойства Resources. Данное свойство опредлено в базовом классе **FrameworkElement**, поэтому его имеют большинство классов WPF.

В чем смысл использования ресурсов? Они повышают эффективность: мы можем определить один раз какой-либо ресурс и затем многократно использовать его в различных местах приложения. В связи с этим улучшается поддержка - если возникнет необходимость изменить ресурс, достаточно это сделать в одном месте, и изменения произойдут глобально в приложении.

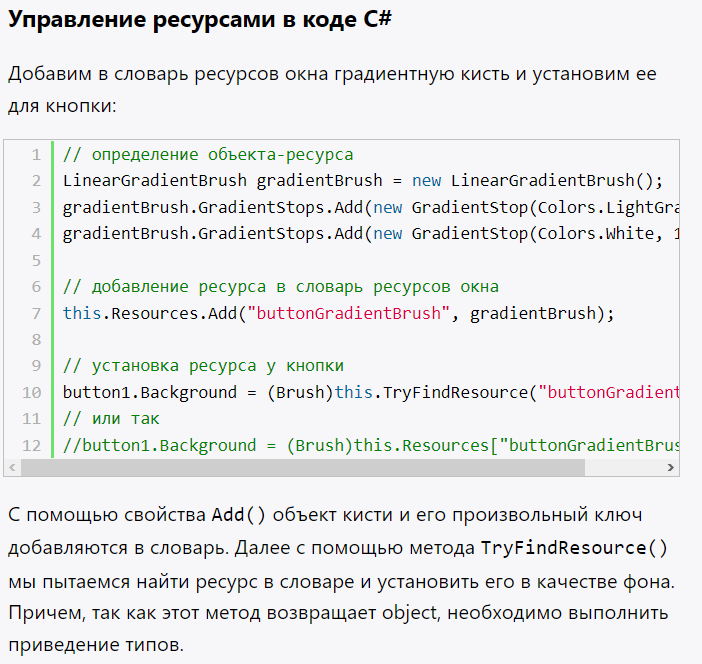
Свойство **Resources** представляет объект **ResourceDictionary** или словарь ресурсов, где каждый хранящийся ресурс имеет определенный ключ.

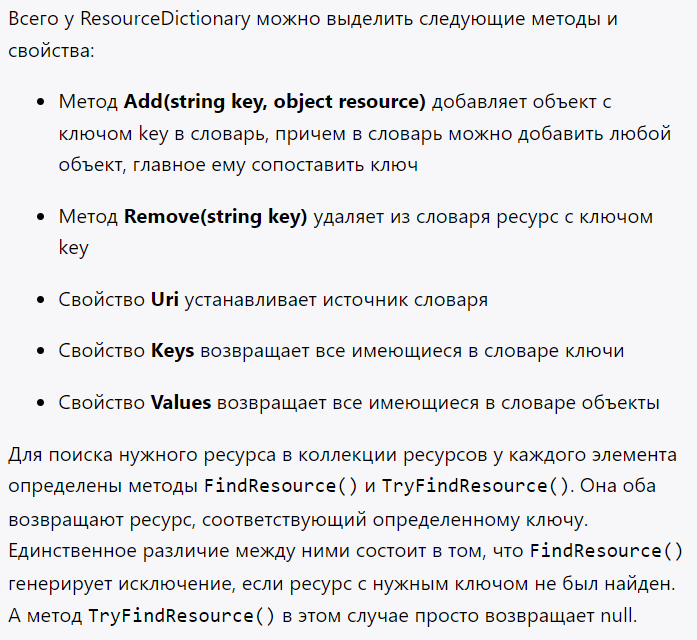
**Определение ресурсов:**

****

Здесь у окна определяются два ресурса: redStyle, который представляет объект SolidColorBrush, и gradientStyle, который представляет кисть с линейным градиентом. У кнопки определен один ресурс darkStyle, представляющий кисть SolidColorBrush. Причем каждый ресурс обязательно имеет свойство **x:Key**, которое и определяе ключ в словаре.

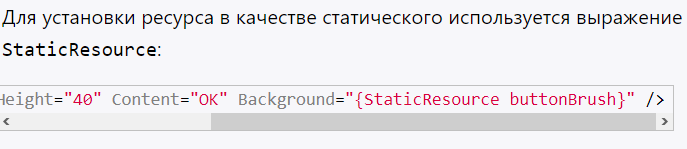
А в свойствах Background соответственно у грида и кнопки мы можем применить эти ресурсы: Background="{StaticResource gradientStyle}" - здесь после выражения StaticResource идет ключ применяемого ресурса.

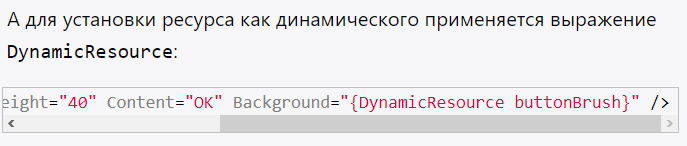
****

****

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

**Статические** ресурсы устанавливается только один раз. А **динамические** ресурсы могут меняться в течение работы программы. Причем один и тот же ресурс может быть и статическим и динамическим.

****

****

В то же время надо отметить, что мы все равно может изменить статический ресурс - для этого нужно менять не сам объект по ключу, а его отдельные свойства.

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 (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.)
* **Триггеры данных**: вызываются в ответ на изменения значений любых свойств (они необязательно должны быть свойствами зависимостей) (Для соединения с отслеживаемыми свойства триггеры данных используют выражения привязки) (в файлике Addgoods)



С помощью свойства Binding триггер данных устанавливает привязку к отслеживаемому свойству. Свойство **Value** задает значение отлеживаемого свойства, при котором сработает триггер

* **Триггеры событий**: вызываются в ответ на генерацию событий (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.) (в файлике MainWindow)
* **Мультитриггеры**: вызываются при выполнении ряда условий (При необходимости отслеживания не одного, а сразу нескольких свойств используют. Он содержит коллекцию элементов **Condition**, каждый из которых, как и обычный триггер, определяет отслеживаемое свойство и его значение) (в файлике Addgoods)

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

Локализация позволяет элементам управления изменяться в соответствии с текущими параметрами культуры в Windows. При локализации в WPF используются интерфейсы API в пространстве имен [System.Windows.Markup.Localizer](https://docs.microsoft.com/ru-RU/dotnet/api/system.windows.markup.localizer). Эти интерфейсы API поддерживают инструмент командной строки [LocBaml Tool Sample](https://github.com/microsoft/WPF-Samples/tree/master/Tools/LocBaml)

Существует несколько вариантов локализации приложения WPF. Например, можно привязать локализованные ресурсы в приложении к файлу XML, хранить локализуемый текст в resx-таблицах или же указать локализатору, использовать файлы XAML

В этом разделе описывается рабочий процесс локализации, использующий BAML-форму XAML, который предоставляет несколько преимуществ.

* Локализация может осуществляться после сборки.
* Можно выполнить обновление до более новой версии BAML-формы XAML с локализациями из старой версии BAML-формы XAML, чтобы обеспечить возможность локализации в процессе разработки.
* Можно проверять элементы и семантику исходного источника во время компиляции, так как BAML-форма XAML является скомпилированной формой XAML.

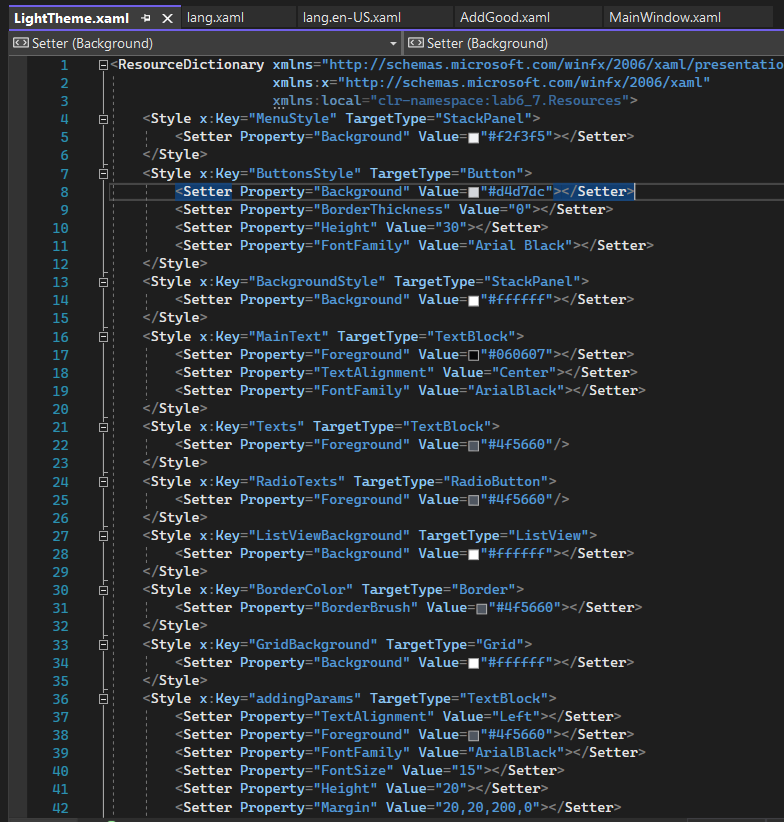
Процесс локализации начинается после сборки нелокализованного файла MyDialog.resources.dll. Элементы и свойства пользовательского интерфейса в исходном коде XAML извлекаются из BAML-формы XAML в пары "ключ — значение" с помощью интерфейсов API в [System.Windows.Markup.Localizer](https://docs.microsoft.com/ru-RU/dotnet/api/system.windows.markup.localizer). Локализаторы используют пары "ключ —значение" для локализации приложения. После завершения локализации можно создать файл .resource.dll на основе новых значений.

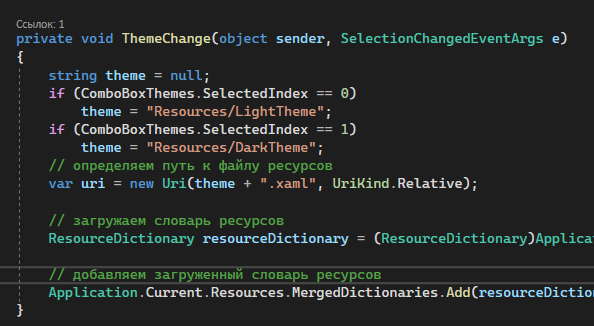
Ключи пар "ключ—значение" — это значения x:Uid, которые разработчик помещает в исходный пользовательский интерфейс после начала локализации локализатором. Изменения разработки можно объединить с уже завершенной работой локализации, чтобы минимальная работа по переводу была потеряна

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.

Для этого добавляется в проект новый файл словаря ресурсов. Здесь указаны все те стили, которые применяются элементами окна. И добавляется еще один файл темы, для того, чтобы менять темы с помощью этих двух файлов (1 типо установлен по-умолчанию)

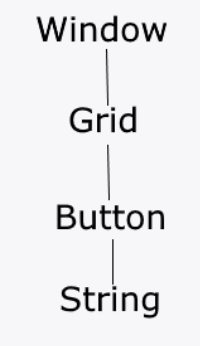
****

****

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

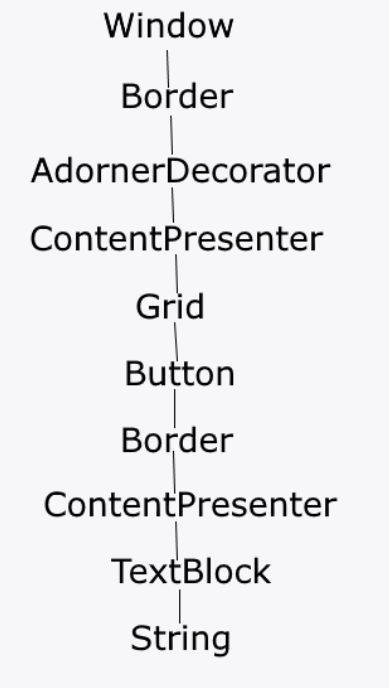
Несмотря на то, что стили существенно облегчают манипулирование внешним видом элементов управления, гораздо более сильным средством в плане визуализации являются **шаблоны**. В отличие от стилей **шаблоны** помогают полностью менять модель визуализации элемента

Визуализация в WPF тесно связана с такими понятиями как логическое и визуальное дерево. Эти деревья являются своего рода каркасом приложения.

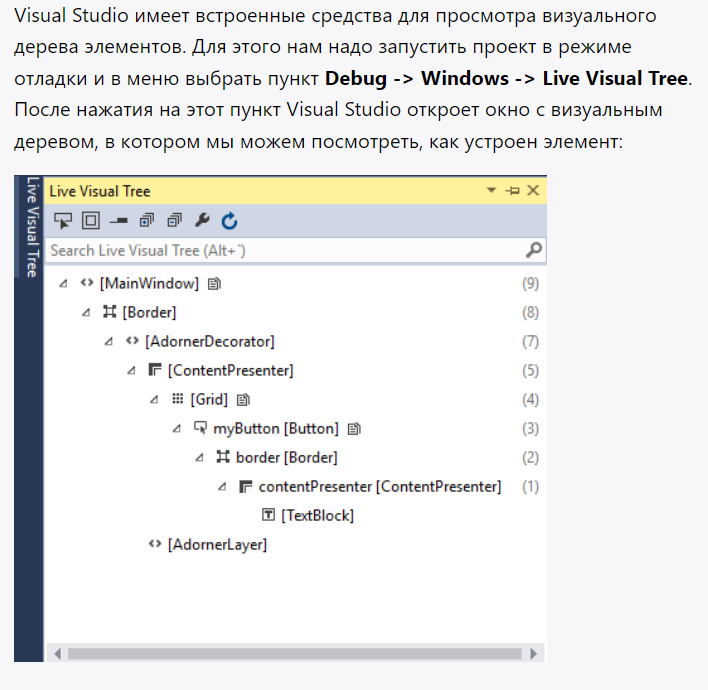
****

То есть в Window есть Grid, в Gride - элемент Button, в кнопке в качестве содержимого установлен некоторый текст в виде объекта String. В итоге получается некое дерево элементов, которое называется **логическим**. В WPF оно представлено классом **System.Windows.LogicalTreeHelper**. Логическое дерево имеет дело с визуализацией как таковой, оно образует модель доступа к дочерним элементам.

От него отличается **визуальное дерево**, представленное классом **System.Windows.Media.VisualTreeHelper**. Так, визуальное дерево для вышеприведенной разметки xaml будет выглядеть следующим образом

****

Визуальное дерево получается гораздо сложнее, оно показывает, как с визуальной точки зрения устроен элемент, из каких частей он состоит.

****

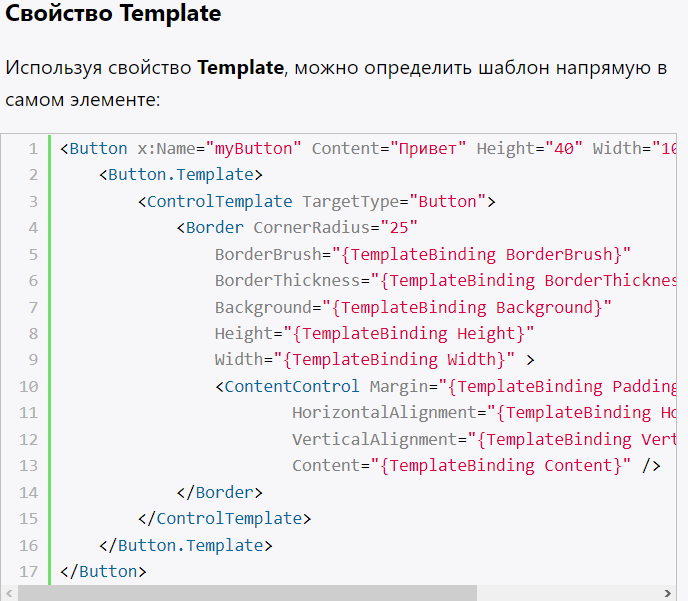
Все визуальные элементы в WPF уже имеют встроенные шаблоны, которые определяют визуальное дерево, структуру и даже поведение элементов. Однако мощь шаблонов состоит в том, что мы можем их переопределить по своему вкусу. Например, сделать круглое окно, а не квадратное, или кнопку в виде морской звезды.

Мы можем определять шаблоны с через стили, а можем в виде отдельных ресурсов.

****

Так, в данном случае с помощью элемента **ControlTemplate** определяется ресурс с ключом "btTemplate".

В ControlTemplate вложены элементы Border и ContentControl, которые через свои свойства определяют, как будет выглядеть кнопка.

****

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.

Первый шаг в создании пользовательского элемента управления — это выбор корректного базового класса для наследования.

* **FrameworkElement**
* **Control**
* **ContentControl**
* **UserControl**
* **ItemsControl и Selector**
* **Panel**
* **Decorator**
* **Специфический класс элемента**

**UserControl**

Это элемент управления с содержимым, который может быть сконфигурирован с применением поверхности времени проектирования. Хотя такой пользовательский элемент управления не настолько отличается от обычного элемента управления с содержимым, обычно он используется тогда, когда необходимо быстро повторно применить неизменный блок пользовательского интерфейса в более чем одном окне (вместо создания действительно отдельного элемента управления, который может быть перенесен из одного приложения в другое)

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

Большую роль при работе с данными играет механизм привязки/

Для создания привязки применяется элемент **Binding** и его свойства:

* **ElementName**: имя элемента, к которому идет привязка. Если мы говорим о привязке данных, то данное свойство задействуется редко за исключением тех случаев, когда данные определены в виде свойства в определенном элементе управления
* **Path**: ссылка на свойство объекта, к которому идет привязка
* **Source**: ссылка на источник данных, который не является элементом управления

Свойства элемента Binding помогают установить источник привязки. Для установки источника или контекста данных в элементах управления WPF предусмотрено свойство **DataContext**.



1. **Объясните назначение интерфейса INotifyPropertyChanged**

**Интерфейс** **INotifyPropertyChanged** используется для уведомления клиентов, как правило, привязки клиентов о том, что значение свойства изменилось