Лабораторная Работа № 6

События, шаблоны, стили и триггеры WPF.

Цель работы: Изучить События, шаблоны, стили и триггеры WPF. Научиться реализовывать приложения визуально с использованием шаблонов и стилей.

Краткие теоретические сведения

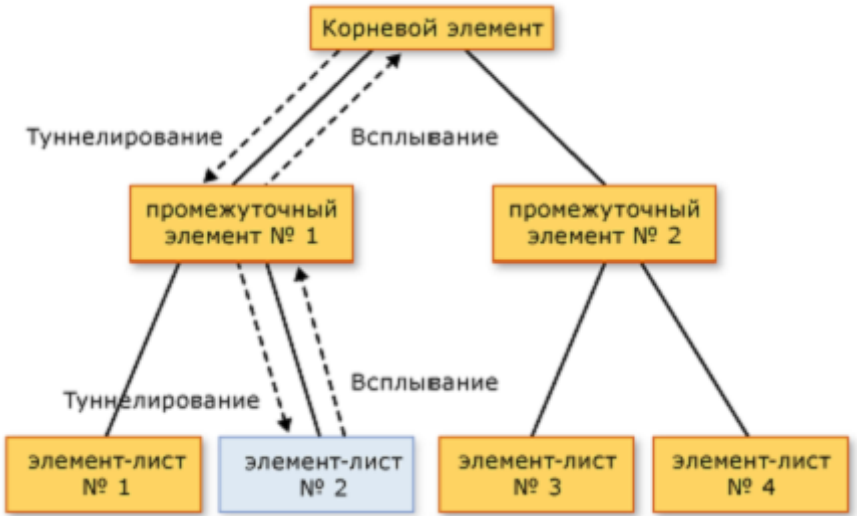
События WPF

События в WPF напоминают события WinForms и события .NET вообще (events)— события определяются в классах элементов управления и вызываются по выполнению некоторого действия (например, нажатия на кнопку). Программисты подписываются на события при помощи оператора «+=» и регистрируют таким образом обработчик события. Однако, в отличие от обыкновенных событий WinForms, события WPF являются переадресуемыми (routed). Переадресуемые события отличаются от обыкновенных тем, что они вызываются не только в тех местах, где возникли события (например, на кнопке), но и на остальных связанных элементах управления (дочерних либо родительских).

В WPF различают три стратегии адресации событий:

- Нисходящая маршрутизация (tunnelling)
- Восходящая маршрутизация (bubbling)
- Прямой вызов

Все стратегии иллюстрируются следующим рисунком:



В случае туннелирования (нисходящая маршрутизация) события, произошедшие на элементах верхнего уровня (родительских) вызываются и для элементов нижнего уровня. В случае всплывания (восходящая маршрутизация) события, произошедшие на элементах нижнего уровня (дочерних) вызываются и у элементов верхнего уровня.

С точки зрения WPF это события с префиксом Preview (туннелирование) и без него. Основная идея заключается в том, чтоб дать необходимую гибкость при обработке различных событий. Например, есть некоторая канва с набором различных элементов. Необходимо логгировать каждый щелчок по канве. В традиционной ситуации для выполнения такого требования пришлось бы подписаться на событие щелчка мышки на канве и на каждом из элементов, принадлежащем канве (поскольку щелчок может выполниться и на них, но это тоже считается как щелчок по канве). В WPF можно сделать проще — зарегистрироваться на PreviewClick событие у канвы; в таком случае перед каждым щелчком по отдельному элементу сначала будет срабатывать событие у канвы. Иногда возникает ситуация, когда необходимо не допустить адресацию события дальше по цепочке родительских или дочерних элементов — для этого существует булевское свойство Handled. Устанавливая это свойство в true можно принудительно прервать маршрутизацию события.

Общий синтаксис событий обыкновенный:

<Button Click="b1SetColor">button</Button>

```
void b1SetColor(object sender, RoutedEventArgs args)
{
  //logic to handle the Click event
    // для прерывания маршрутизации события
    // args.Handled = true;
}
```

Стили WPF

Одна из особенностей WPF — возможность изменять внешний вид элементов управления посредством стилей. Стиль определяет значения свойств элемента управления — ширина рамки, фоновый цвет, цвет текста, отступы, размер шрифта и другие.

По сути, все свойства, выставляемые при помощи XAML напрямую могут быть выставлены в отдельном стиле. Это удобно, когда необходимо переиспользовать одни и те же наборы настроек — например, каждая кнопка должна содержать текст кеглем 15 пунктов. Рассмотрим простейший пример стиля:

```
<Style TargetType="TextBlock">
  <Setter Property="HorizontalAlignment" Value="Center" />
  <Setter Property="FontFamily" Value="Comic Sans MS"/>
  <Setter Property="FontSize" Value="14"/>
</Style>
```

У стиля прежде всего указывается тип, к которому он применяется. Если не указать стиль, будет считаться, что он применяется к базовому для всех элементов управления типу FrameworkElement. В таком случае набор устанавливаемых свойств будет ограничен доступными FrameworkElement. Далее следует набор «установщиков» - каждый указывается значение конкретного свойства. В данном примере устанавливается центрирование по горизонтали и 14 кегль шрифта Comic Sans для всех текстовых подписей.

Стили как правило помещаются в т. н. ресурсы. Ресурсы — это специальное хранилище объектов внутри элементов управления (наподобие tag элемента в Windows Forms). В ресурсы могут помещаться изображения, коллекции объектов, стили, шаблоны и другие элементы. Ключевой особенностью ресурсов является то, что среда выполнения WPF производит поиск по ресурсам элементов управления при необходимости подключить тот или иной объект.

Рассмотрим пример:

```
<Window.Resources>
<!--A Style that extends the previous TextBlock Style-->
<!--This is a "named style" with an x:Key of TitleText-->
<Style BasedOn="{StaticResource {x:Type TextBlock}}"
      TargetType="TextBlock"
      x:Key="TitleText">
  <Setter Property="FontSize" Value="26"/>
  <Setter Property="Foreground">
  <Setter.Value>
     <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
       <LinearGradientBrush.GradientStops>
         <GradientStop Offset="0.0" Color="#90DDDD" />
         <GradientStop Offset="1.0" Color="#5BFFFF" />
       </LinearGradientBrush.GradientStops>
     </LinearGradientBrush>
   </Setter.Value>
  </Setter>
</Style>
...
</Window.Resources>
<TextBlock Style="{StaticResource TitleText}" Name="textblock1">My Pictures</TextBlock>
```

```
<TextBlock>Check out my new pictures!</TextBlock>
```

Прежде всего, для применения ресурса к элементу управления необходимо дать ему ключ в словаре ресурсов посредством установки значения x:Key. У каждого элемента управления есть свойство Style, в котором указывается ключ стиля. В предыдущем примере ключ указан не был — так тоже можно объявлять ресурс, тогда он станет стилем по-умолчанию (базовым) для всех элементов управления указанного типа. Стили имеют также систему наследования — указываемую при помощи аттрибута BasedOn. Отметим также, что в случае сложных значений свойств (например, фон указываемый при помощи градиента с несколькими точками), можно использовать Setter.Value.

Шаблоны WPF

```
<ControlTemplate TargetType="Button">
  <Border Name="RootElement">
    <!--Create the SolidColorBrush for the Background
       as an object elemment and give it a name so
       it can be referred to elsewhere in the
       control template.-->
    <Border.Background>
      <SolidColorBrush x:Name="BorderBrush" Color="Black"/>
    </Border.Background>
    <!--Create a border that has a different color
       by adding smaller grid. The background of
       this grid is specificied by the button's
       Background property.-->
    <Grid Margin="4" Background="{TemplateBinding Background}">
      <!--Use a ContentPresenter to display the Content of
         the Button.-->
      <ContentPresenter
       HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
       VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
       Margin="4,5,4,4" />
    </Grid>
  </Border>
</ControlTemplate>
```

В данном случае кнопка представляет собой рамку черного цвета, в которой находится сетка с отступом в 4 платформо-независимых единицы (в общем случае, 4 пикселя) со всех сторон. Внутри сетки находится ContentPresenter — специальный элемент, который отображает то значение, которое будет написано разработчиков в Button. Шаблон можно переопределить при помощи стиля и установщика значения свойства Template, при помощи непосредственного применения шаблона из словаря ресурсов либо непосредственно. Примеры:

```
<Window.Resources>
<Style TargetType="TextBlock" x:Key="TemplatedTextBlock">
  <Setter Property="Template">
  <Setter.Value>
     <ControlTemplate>
...
</ControlTemplate>
   </Setter.Value>
  </Setter>
</Style>
...
<TextBlock Style="{StaticResource TemplatedTextBlock}" />
<TextBlock Template="{StaticResource TextBlockTemplate}" />
```

```
<TextBlock>
<TextBlock.Template>
<ControlTemplate>
...
</ControlTemplate>
</TextBlock.Template>
</TextBlock>
```

Триггеры WPF

Триггеры — простая замена событийному движку WPF для выполнения некоторых действий или изменения внешнего вида компонента при изменении значений его свойств.

Простейший пример триггера — подсвечивание элемента при наведении мышки. Рассмотрим его:

```
<Style TargetType="ListBoxItem">
  <Setter Property="Opacity" Value="0.5" />
  <Setter Property="MaxHeight" Value="75" />
  <Style.Triggers>
    <Trigger Property="IsSelected" Value="True">
       <Setter Property="Opacity" Value="1.0" />
    </Trigger>
...
  </Style.Triggers>
</Style>
```

Триггеры определяются в стилях или шаблонах элементов. У триггера указывается свойство, к которому он привязывается, в данном случае — это свойство «выбран ли элемент» (IsSelected). Далее указывается значение свойства, по которому срабатывает триггер. И внутри — описание необходимых действий, как правило это установка свойств элемента — в данном случае выделенные элементы становятся непрозрачными. Вот примерный вид интерфейса:

Специально для поддержки механизма триггеров команда Microsoft создала набор базовых булевских свойств у элементов управления — IsSelected, IsMouseOver, IsEnabled и другие.

Триггеры существуют следующих типов:

- Trigger — выполняется по изменению одного из свойств объекта;
- EventTrigger — выполняется по возникновению маршрутизуемого события;
- MultiTrigger — выполняется по выполнению множества условий (аналог нескольких последовательно связанных триггеров);
- DataTrigger — выполняется по выполнению условия не по свойству, а по данным, связанным с элементом управления;
- MultiDataTrigger — условия берутся из множества связанных данных;

Триггеры в целом представляют очень удобный механизм добавления визуальной интерактивности элементов при помощи визуального оформления без использования логики отрисовки.

Задание

1. Ознакомиться с краткими теоретическими сведениями.
2. Реализовать все формы приложения визуально с использованием шаблонов и стилей.
3. Реализовать логику работы одной формы полностью от изначальной загрузки данных из БД до сохранения при помощи событий WPF.
4. Оформить отчет о проделанной работе.

Контрольные вопросы

1. Что такое шаблоны элементов управления WPF ?
2. Что представляют собой триггеры в WPF ?
3. Как определяются стили в WPF ? Что они содержат ?