Основные сведения о языке XAML

Построение пользовательских интерфейсов для WPF-приложений осуществляется с применением языка расширенной разметки приложений *XAML (Extensible Application Markup Language). XAML-документ* содержит разметку, описывающую внешний вид и поведение окна или страницы приложения, а связанные с ним файлы кода *C#* - логику приложения. Язык *XAML* обеспечивает разделение процесса дизайна приложения (графической части) и разработки бизнес-логики (программного кода) между дизайнерами и разработчиками. WPF *XAML* является подмножеством языка *XML* и позволяет описывать WPF-содержимое таких элементов как *векторная графика*, *элементы управления* и документы.

XAML базируется на языке расширенной разметки *XML (Extensible Markup Language)* и его синтакс определяется следующими правилами:

- каждый элемент *XAML*-документа отображается на некоторый экземпляр класса .NET. Имя такого элемента в точности соответствует имени класса. Например, элемент `< Button >` служит для WPF инструкцией для построения объекта класса Button;
- элементы *XAML* можно вкладывать друг в друга. Вложение *элементов разметки* обычно отображает вложенность элементов интерфейса;
- свойства класса определяются с помощью атрибутов или с помощью вложенных дескрипторов со специальным синтаксисом.

Язык *XAML* характеризуется самоописанием. Каждый элемент в *XAML*-документе представляет имя типа (такие как `Button, Window` или `Page`) в рамках заданного пространства имен. Атрибуты элементов используются для задания свойств (`Name, Height, Width` и т.п.) и событий (`Click, Load` и т.д.) соответствующих объектов.

При создании WPF-приложения *WpfApp1* VisualStudio генерирует следующий *XAML*-документ.

```
      <Window x:Class="WpfApp1.MainWindow"


xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
              xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
              xmlns:local="clr-namespace:WpfApp1"
              mc:Ignorable="d"
              Title="MainWindow" Height="450" Width="800">
          <Grid>

          </Grid>
      </Window>
```

XAML-документ WPF-приложения *WpfApp1* начинается с дескриптора `< Window...>`. Все дескрипторы *XAML*-документа начинаются символами <, а завершаются символами >. Любой *XAML*-документ состоит из *XAML*-элементов. Каждый *XAML*-документ (*XAML*-элемент) начинается *открывающимся дескриптором* (например `< Window >`), за которым следует содержимое документа (например, текстовая строка или другие *XAML*-элементы). В открывающем дескрипторе могут присутствовать описания атрибутов (например, `Class, xmlns, Title, Height, Width` и др.). *XAML*-документ (*XAML*-элемент) должен завершаться *закрывающимся дескриптором* (например, `/>` или `< /Window >`). Текст *XAML*-документа должен содержать *один корневой элемент* – элемент верхнего уровня вложенности. В *XAML*-документе WPF-приложения *WpfApp1* таким элементом является `< Window >`. В *корневой элемент* могут быть добавлены другие *XAML*-элементы. В рассматриваемом *XAML*-документе это элемент `< Grid >`.

В процессе компиляции *XAML*-документа WPF-приложения синтаксический анализатор переводит *XAML* файлы в файлы языка двоичной разметки приложений *BAML* (*Binary Application Markup Language*), которые затем встраиваются в виде ресурсов в сборку проекта. Для построения классов WPF-приложения синтаксический анализатор использует *пространство* имен, которое определено в корневом дескрипторе *XAML*-документа.

Пространство имен в *XAML*-документе задается с помощью атрибута xmlns. В приведенном выше документе объявлено несколько базовых пространства имен:

- `xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"` – *это базовое пространство имен WPF, которое охватывает все классы WPF, включая элементы* управления, которые применяются при построении пользовательского интерфейса. Так как данное пространство имен объявлено без префикса, то оно распространяется на весь *XAML*-документ;
- `xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"` – пространство имен *XAML*. Оно включает различные свойства утилит *XAML*, которые позволяют влиять на то, как *XAML*-документ следует интерпретировать. Данное пространство имен отображается на префикс *x*. Этот префикс можно помещать перед именем элемента (*x:ИмяЭлемента*).

Второе *пространство* имен используется для включения специфичных для *XAML* лексем – "ключевых слов"

Ключевое слово	**Назначение**
`x:Array`	Представляет *тип массива* .NET на *XAML*
`x:ClassModifier`	Позволяет определять видимость типа класс (internal или public), обозначенного ключевым словом Class
`x:FieldModifier`	Позволяет определять видимость члена типа (internal, public, private или protected) для любого именованного элемента корня. Именованный элемент определяется с использованием ключевого слова Name
`x:Key`	Позволяет установить значение ключа для элемента *XAML*, которое должно быть помещено в элемент словаря
`x:Name`	Позволяет указывать сгенерированное C# имя заданного элемента *XAML*
`x:Null`	Представляет null-ссылку
`x:Static`	Позволяет ссылаться на статический член типа
`x:Type`	*XAML*-эквивалент операции C# typeof (вызывает System.Type на основе указанного имени)

`x:TypeArgument`	Позволяет устанавливать элемент как обобщенный тип с определенными параметрами

Таблица 1. Наиболее часто используемые ключевые слова XAML

В WPF-приложениях кроме базовых применяют специальные, необязательные пространства имен:

- `http://schemas.openxmlformats.org/markup-compatibility/2006` - пространство имен *XAML*, связанное с проблемой совместимости разметки с рабочей средой. Данное пространство имен используется для информирования синтаксического анализатора *XAML* о том, какую информацию необходимо обработать, а какую – проигнорировать;
- `http://schemas.microsoft.com/expression/blend/2008` - пространство имен *XAML*, поддерживаемое программами Expression Blend и Visual Studio. Данное пространство имен используется для установки размеров графической панели для страницы.

В атрибуты объекта Window могут быть добавлены следующие *XAML*-описания.

```
     xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
     mc:Ignorable="d" d:DesignHeight="450" d:DesignWidth="800"
```

Данное *XAML*-описание объявляет необязательные пространства имен с префиксами `mc` и `d`. Свойства `DesignHeight` и `DesignWidth` находятся в пространстве имен, помеченном префиксом d. Данные свойства определяют, что во время разработки проекта приложения в дизайнере *Visual Studio* окно должно иметь размеры 450x800. Свойство `Ignorable` находится в пространстве имен, помеченном префиксом *mc*, и информирует синтаксический анализатор о том, что он должен игнорировать часть *XAML*-документа, помеченные префиксом *d*.

В *XAML*-документе WPF-приложения часто возникает необходимость обеспечить *доступ* к каким-либо другим пространствам имен проекта. В этом случае необходимо определить новый *префикс* и задать *пространство* имен. Если в проекте имеется *пространство* имен *WpfApp1.Commands*, то подключение его к *XAML*-документу WPF-приложения будет иметь следующий вид (`command` – используется в качестве префикса).

```
xmlns:command="clr-namespace: WpfApp1.Commands"
```

Префикс (*command*) используется для ссылки на *пространство* имен в *XAML*-документе. *Лексеме clr-namespace* присваивается название пространства имен *.NET* в сборке.

Для описания класса в *XAML*-документе используется *атрибут Class*. Строка *XAML*-документа `<Window x:Class="WpfApp1.MainWindow" ...>` предписывает создать *класс* `WpfApp1.MainWindow` на базе класса `Window`. *Префикс* `x` атрибута `Class` определяет то, что данный *атрибут* помещается в *пространство* имен *XAML*. *Класс* `MainWindow` генерируется автоматически во *время компиляции*. Для части класса автоматически генерируется код (частичный (partial) *класс*):

```
namespace WpfApp1
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

Когда выполняется *компиляция* приложения, *XAML-файл*, который определяет пользовательский *интерфейс* (*MainWindow.xaml*), транслируется в объявление типа *CLR* , которое объединяется с логикой приложения из файла класса отдельного кода (*MainWindow.xaml.cs*).

Метод *InitializeComponent()* генерируется во *время компиляции* приложения и в исходном коде не присутствует.

Для программного управления элементами управления, описанными в *XAML*-документе необходимо для элемента управления задать *XAML атрибут Name*. Так для задания имени элементу *Grid* необходимо записать следующую разметку:

```
<Grid Name="grid">


</Grid>
```

Простые свойства задаются в *XAML*-документе в соответствии со следующим синтаксисом:

```
ИмяСвойства ="значение"
```

Например, Name = "grid1"

При необходимости задать свойство, которое является полноценным объектом, используются *сложные свойства* в соответствии с синтаксисом "свойство-элемент":

```
Родитель.ИмяСвойства
```

Например, для контейнера `StackPanel` нам необходимо задать градиентную кисть для заливки панели, что определяется атрибутом `Background`. Это реализуется с помощью дескрипторов

`<StackPanel.Background> . . . </StackPanel.Background>`.

Для задания значения свойства из выделенного класса используется *расширение разметки*, которое обеспечивает расширение грамматики *XAML* новой функциональностью. Расширения разметки могут использоваться во вложенных дескрипторах или в *XAML*-атрибутах. Когда это используется в атрибутах, то необходимо применять *фигурные скобки* {...}.

Расширения разметки используют следующий *синтаксис*:

```
{КлассРасширенияРазметки  Аргумент }
```

Расширения разметки реализуются классами, дочерними от класса `System.Windows.Markup.MarkupExtention`.

Базовый *класс* `MarkupExtention` имеет метод `ProvideValue()`, который предоставляет нужное *значение* для атрибута. Например, для задания атрибуту `Foreground` объекта `Button` статического свойства, определенного в другом классе, необходимо создать следующее *XAML*-описание.

```
<Button Foreground="{x:Static SystemColors.ActiveCaptionBrush}"    />
```

При компиляции синтаксический анализатор создаст экземпляр класса `Static Extention`, затем вызовет метод `ProvideValue()`, который извлечет нужное *значение* и установит его для свойства `Foreground`.

Расширения разметки могут использоваться как *вложенные свойства*.

Присоединенные свойства описывают свойства, которые могут применяться к нескольким элементам управления, но которые определены в другом классе. В WPF-приложениях присоединенные свойства часто используются для управления компоновкой элементов интерфейса. *Синтаксис* присоединенных свойств следующий.

```
ОпределяющийТип.ИмяСвойства
```

Например, если необходимо расположить кнопку в нулевой строке сетки, то необходимо сделать следующее *XAML*-описание.

```
<Button . .. Grid.Row="0" >
....
</Button>
```

Здесь присоединенным свойством является `Grid.Row`, то есть свойство `Row` элемента `Grid`, которое не является свойством объекта `Button`. Свойство `Row` присоединяется к свойствам объекта `Button`, поскольку данный *объект* располагается в контейнере `Grid`.

Атрибуты объектов могут использоваться для присоединения обработчиков событий, используя следующий *синтаксис*.

```
ИмяСобытия = "ИмяМетодаОбработчикаСобытия"
```

Например, для кнопки событию её нажатия `Click` можно установить обработчик события `Exit_Click`.

```
<Button Name="Exit" Content="Выход" Click="Exit_Click" />
```

Определяя в *XAML*-описании обработчик `Exit_Click`, необходимо в коде класса иметь метод с корректной сигнатурой. Ниже приведен код, который генерируется автоматически при создании описания обработчика события в *XAML*-описании.

```
private void Exit_Click(object sender, RoutedEventArgs e)
        {
    //действия по клику на кнопку «Выход»
        }
```

Задания

Создайте новое графическое приложение название приложения – lab2.1.

Замените значение атрибута Title в контейнере Window на значение «lab2.1 OOP (Фамилия и инициалы автора)».

1. Создание меню

1.1Добавьте в XAML-разметку внутрь контейнера `Grid` следующий код

```
            <Menu x:Name="MainMenu" Height="25"
VerticalAlignment="Top" Width="auto" Margin="0,0,0,0"
VerticalContentAlignment="Center" >
                <MenuItem Header="File"
HorizontalAlignment="Center" VerticalAlignment="Center"
HorizontalContentAlignment="Center">
                    <MenuItem Header="Exit"/>
                </MenuItem>
                <MenuItem Header="Help"
HorizontalAlignment="Center" VerticalAlignment="Center"
HorizontalContentAlignment="Center">
                    <MenuItem Header="Open help"/>
                    <MenuItem Header="About..."/>
                </MenuItem>
            </Menu>
```

1.2 Добавьте в получившееся меню следующие пункты:

- Help → Open author page
- Service
- Service → Connect to service
- Service → Leave a complaint
- File → New…

1.3 Добавьте для каждого пункта меню свои собственные события (каждый ссылается на свое, а не на общее для всех) `Click` – вывод сообщения «Sorry, this feature is temporarily unavailable, technical work is underway».

1.4 Для события `Click` по пункту меню Help → About… выведите сообщение об авторе приложения.

1.5 Для события `Click` по пункту меню Help → Connect to service выведите сообщение как можно связаться с автором приложения.

1.6 Измените событие пункта File → Exit на закрытие окна приложения, для этого в коде этого события (C#) используйте следующий код

```
    this.Close();
```

1.7 придумайте и добавьте еще 4 пункта меню (можно добавлять как корневые пункты, так и вложенные)

2. Добавление элементов

2.1 Изучите пример использования элемента ComboBox

```
    <ComboBox x:Name="comboBox" HorizontalAlignment="Left"
Margin="10,30,0,0" VerticalAlignment="Top" Width="120">
                <ComboBoxItem Content="Вариант 1"/>
                <ComboBoxItem Content="Вариант 2"/>
            </ComboBox>
```

2.2 Изучите примеры использования RadioButton.

Пример 1. Использование RadioButton без группировки

```
    <RadioButton x:Name="radioButton1" Content="RadioButton_1"
HorizontalAlignment="Left" Margin="10,57,0,0"
VerticalAlignment="Top" IsChecked="True"/>
            <RadioButton x:Name="radioButton2"
Content="RadioButton_2" HorizontalAlignment="Left"
Margin="10,77,0,0" VerticalAlignment="Top"/>
            <RadioButton x:Name="radioButton3"
Content="RadioButton_3" HorizontalAlignment="Left"
Margin="10,97,0,0" VerticalAlignment="Top" />
            <RadioButton x:Name="radioButton4"
Content="RadioButton_4" HorizontalAlignment="Left"
Margin="10,117,0,0" VerticalAlignment="Top"/>
```

Пример 2. Использование RadioButton с группировкой

```
        <GroupBox x:Name="groupBox_A" Header="GroupBox_A"
HorizontalAlignment="Left" Height="75" Margin="310,183,0,0"
VerticalAlignment="Top" Width="159">
                <Grid>
                    <RadioButton x:Name="radioButton1" Content="RadioButton_1"
HorizontalAlignment="Left" Margin="4,5,0,0" VerticalAlignment="Top"
IsChecked="True"/>
                    <RadioButton x:Name="radioButton2" Content="RadioButton_2"
HorizontalAlignment="Left" Margin="4,25,0,0" VerticalAlignment="Top"/>
                </Grid>
            </GroupBox>
            <GroupBox x:Name="groupBox_B" Header="GroupBox_B"
HorizontalAlignment="Left" Height="75" Margin="474,183,0,0"
VerticalAlignment="Top" Width="159">
                <Grid>
                    <RadioButton x:Name="radioButton3" Content="RadioButton_3"
HorizontalAlignment="Left" Margin="4,5,0,0" VerticalAlignment="Top" />
                    <RadioButton x:Name="radioButton4" Content="RadioButton_4"
HorizontalAlignment="Left" Margin="4,25,0,0" VerticalAlignment="Top"/>
                </Grid>
            </GroupBox>
```

2.3 Создайте в приложении форму для анкетирования. Форма должна включать в себя минимум (можно больше перечисленного, но не меньше)

- 5 `TextBlock` или 5 `Label`,
- 5 `TextBox`,
- 2 `ComboBox`,
- 3 `CheckBox` (можно не в одном вопросе),
- 4 `RadioButton` (можно использовать лишь в одном вопросе).

Для использования многострочного ввода текста используйте `TextBox` с параметрами `TextWrapping="Wrap"` и `AcceptsReturn="True"`, также можете включить `AcceptsTab` и `SpellCheck.IsEnabled`.

В качестве темы для анкетирования возьмите тему из предложенного ниже списка, номер которой совпадает с вашим номером в журнале

1. Оценка мероприятия

2. Узнаваемость бренда
3. Анкета постоянного покупателя
4. Форма обратной связи для сайта
5. Самооценка сотрудника
6. Карьерные ожидания сотрудников
7. Анкета для регистрации волонтера
8. Оценка эффективности размещения интернет-рекламы
9. Оценка дизайна социальной сети
10. Оценка качества работы компании
11. Анкета-заявка для получения скидочной карты
12. Анкета-заявка для заказа товара в интернет-магазине
13. Анкета для сбора идей и предложений
14. Оценка качества работы обслуживающего персонала
15. Исследование уровня сплоченности коллектива
16. Анкета соискателя на замещение должности
17. Контроль качества товара N в нескольких торговых точках
18. Анкета-отзыв о качестве доставки готовых обедов
19. Выявление причин потери клиентов
20. Анкета тайного покупателя
21. Отношение современной молодежи к религии и нравственности
22. Оценка зависимости от гаджетов
23. Оценка зависимости от социальных сетей
24. Выявление популярного тренда в кино
25. Выявление популярного тренда в музыке
26. Обратная связь при ошибке приложения

3. Заполните таблицу

3.1 Заполните таблицу ниже, для каждого элемента укажите не менее четырех его свойств/полей и не менее двух его методов

Объект	***Класс***	***Свойства***	***Методы***
Текстовое поле			
Надпись			

Список	ListBox		
Поле со списком	ComboBox		
Ползунок	Slider		
Переключатель	RadioButton		
	CheckBox		
	Menu		
	ProgressBar		
	ScrollBar		
	StatusBar		
	GroupBox		
	TabControl		

4. Приложение по таблице

4.1 Создайте новое графическое приложение название приложения – lab2.2. Замените значение атрибута Title в контейнере Window на значение «lab2.2 OOP (Фамилия и инициалы автора)».

4.2 Добавьте в приложение все указанные в задании 3 элементы, и используйте все указанные для них свойства/поля.