

ОСНОВЫ XAML

Фреймворк **WPF** использует удобный инструмент для разметки, основанный на применении декларативных языков. Удобное применение языка **HTML** в качестве разметки текста задало тенденцию для разработки графических интерфейсов.

XAML - eXtensible Application Markup Language – расширяемый язык разметки для приложений.

Язык **XAML** является расширением языка **XML**, который для описания объекта использует теги. **XML** является иерархичным – один объект может включать в себя несколько других, как бы группируя их, что очень удобно для разметки графического интерфейса. Язык **XAML** представляет в виде тегов обычные классы **C#**. Разметка интерфейса и превращается (*парсится, переводится*) в обычный частичный класс **C#** при сборке приложения.

Определим синтаксис языка **Xaml**

Объект

Любой объект представляется тегом.

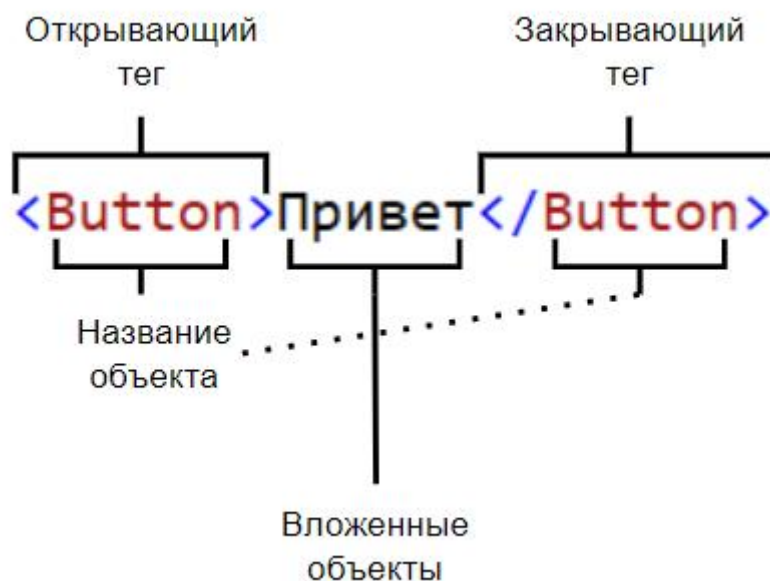


Рисунок 1

Тег описывает объект, тип данных которого указывается между треугольных скобок: `<` и `>`. Закрывающий тег имеет косую черту после первой треугольной скобки.

После сборки приложения, этот тег превращается в конкретный экземпляр класса. Но, если мы привыкли как-то именовать экземпляры, то **WPF** делает это автоматически, однако при необходимости у объекта можно указать атрибут `x:Name`.

Объект имеет вложенную структуру, как в **C#**, где контекст определяется фигурными скобками. Здесь же контекст задаётся парой тегов объекта.

Если объекту не нужно указывать вложенное значение, можно воспользоваться сокращённой записью.

`<Button/>`

Рисунок 2

Здесь закрывающий тег заменяется слешем в конце определения открывающего тега.

Атрибуты

Мы знаем, что у каждого класса в **C#** могут быть определены публичные элементы, такие как свойства, переменные. В языках разметки для составляющих объекта, которым можно задавать значение, такие элементы называются атрибутами.

Каждый атрибут имеет имя, по которому можно обратиться для установки значения. Важным является то, что вне зависимости от типа значения, оно указывается в кавычках.

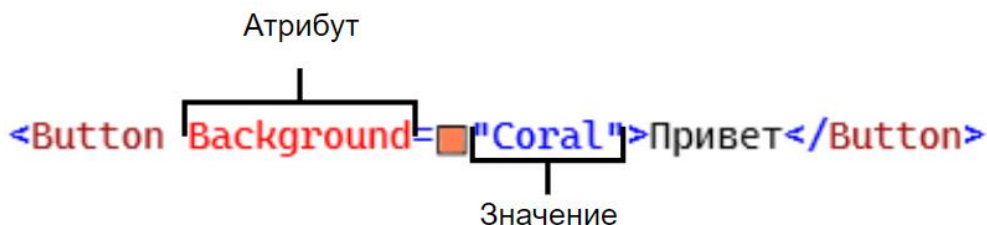


Рисунок 3

Представим тот же пример на языке **C#**.

```
Button knopka = new Button();
knopka.Background = Brushes.Coral;
```

Рисунок 4

Значения атрибутов, которые указываются в **XAML** с помощью внутренних механизмов автоматически типизируются. Этот плюс позволяет не отвлекаться на синтаксические особенности и погрузиться в разработку интерфейса. Но для этого необходимо знать, в каком формате ожидается получить строку для того, чтобы её преобразовать в значение конкретного типа.

● Расширенные значения

Иногда необходимо уточнить то, как отображать значение атрибута или откуда брать данные для свойства. Для этого существует конструкция расширенной разметки, которая описывается в фигурных скобках: { и }.

```
<Button Content="{Binding Path=MyData.Text}"/>
```

Рисунок 5

Для расширенного определения имеются свои правила описания и типизации, с которыми мы познакомимся в дальнейшем.

● Вложенные свойства у атрибута

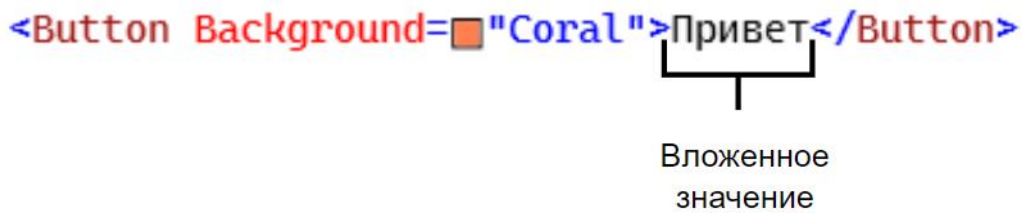
Если мы знаем, что атрибут представляет объект с множеством свойств и нам необходимо получить какое-то конкретное, то можно обратиться к нему через точку.

```
<Button Grid.Row="1"/>
```

Рисунок 6

● Вложенные значения

На самом деле то, что будет описано между тегов тоже является значением некоторого свойства объекта. Это некоторая семантическая (смысловая) особенность **XAML**, обозначающая то, что внутри объекта может что-то храниться, притом, в большинстве случаев, независимо от типа. По примеру мы можем предполагать, что в кнопке мы можем указать её текст, чтобы пользователю было понятно назначение кнопки.

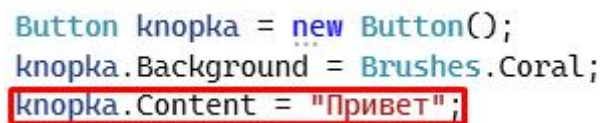


```
<Button Background="Coral">Привет</Button>
```

Вложенное значение

Рисунок 7

Однако конкретно для кнопки, таким образом мы задаём значение свойства **Content** объекта **Button**. Это свойство имеет тип данных **object**. Ну и название **Content** во многом говорит само за себя - это свойство, определяющее содержимое объекта.



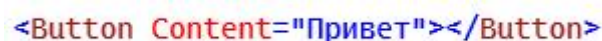
```
Button knopka = new Button();
knopka.Background = Brushes.Coral;
knopka.Content = "Привет";
```

Рисунок 8

В большинстве объектов **XAML** свойство, которое определяет содержимое, называется **Content** или имеет это слово как часть названия свойства.

Так же стоит обратить внимание на тип данных. Тип **object** - это базовый тип всех объектов. При сборке приложения, свойство **Content** типизируется исходя из содержимого.

Так же, доступ к свойству **Content** можно получить через обращение к атрибуту **Content**.



```
<Button Content="Привет"></Button>
```

Рисунок 9

Иногда, вложенное свойство представляет из себя коллекцию объектов. Например, контейнеры компоуют различные элементы. Соответственно свойство **Content** будет представлять не **object**, а **List<object>**. В таких случаях внутри объекта можно указать несколько объектов.



```
<StackPanel>
  <Button/>
  <Button/>
  <Button/>
</StackPanel>
```

Рисунок 10

Элемент свойства

Иногда требуется подробнее описать значение атрибута объекта. В таком случае не обойтись строковым представлением. В **XAML** имеется возможность доступа к свойствам объекта в его контексте (между тегами).

```
<Button>
  <Button.Background>
    <LinearGradientBrush EndPoint="0.5,1" StartPoint="1,0">
      <GradientStop Color="#FF9482F1"/>
      <GradientStop Color="#FFF4F5D6" Offset="1"/>
    </LinearGradientBrush>
  </Button.Background>
  Привет
</Button>
```

В этом примере свойству **Background** присваивается новый объект **LinearGradientBrush**, представляющий градиентную кисть. Такой объект невозможно указать в строке, поэтому для него открывается соответствующий вложенный тег.

Прикрепляемые свойства

Некоторые объекты могут добавлять свойства вложенным в него объектам. Например, объекту **Grid** нужно знать, в какой ячейке сетки расположен вложенный в него элемент. Поэтому вложенные объекты могут получать эти свойства, чтобы установить их значения.

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition/>
  </Grid.RowDefinitions>
  <Button Grid.Row="1"/>
</Grid>
```

Рисунок 11

Это логично, что каждый вложенный элемент должен знать о том, какому родительскому объекту он принадлежит.

В связи с этим, можно определить понятия родительского и дочернего объекта.

Родительский объект - это такой объект, который может иметь один или несколько объектов «внутри». Термин «родительский» применяется только относительно дочернего объекта.

Дочерний объект - объект, который находится «внутри» другого объекта, являясь вложенным.

Описание значений конкретных типов

● Строки

Строка указывается в кавычках «как есть».

```
<TextBox Text="Привет мир!"/>
```

Рисунок 12

**Text - текст*

● Числа

Для указания целочисленного значения, указывают число «как есть»

```
<TextBox Text="Привет мир!"  
FontSize="15"/>
```

Рисунок 13

**FontSize - размер шрифта (может быть представлен и числом с точкой)*

Для указания числа с плавающей точкой, его дробную часть отделяют точкой.

```
<TextBox Text="Привет мир!"
        FontSize="15"
        Opacity="0.5"/>
```

Рисунок 14

**Opacity - прозрачность в диапазоне от 0 до 1*

● Перечисления

Иногда в качестве значения, атрибут ожидает перечисление однотипных объектов. Рассмотрим на свойстве Margin - отступы от границы.

```
<TextBox Text="Привет мир!"
        FontSize="15"
        Opacity="0.5"
        Margin="5"/>
```

Рисунок 15

Указывая одно значение, элемент сделает отступ от всех четырёх границ контейнера, в котором находится. Что нужно сделать, чтобы указать отступ только от левой границы?

Для этого необходимо перечислить значение отступа для каждой из сторон. Разделять значения можно либо через пробел, либо через запятую.

Учитывая то, что свойство **Margin** начинает отсчёт с левой границы, решением поставленной задачи будет либо эта запись:

```
<TextBox Text="Привет мир!"
        FontSize="15"
        Opacity="0.5"
        Margin="5, 0, 0, 0"/>
```

Рисунок 16

Либо такая:

```
<TextBox Text="Привет мир!"
        FontSize="15"
        Opacity="0.5"
        Margin="5 0 0 0"/>
```

Рисунок 17

При указании значений всегда нужно учитывать тип атрибута. Именно он указывает на правила описания значения. Правила для оформления значений атрибутов всегда можно найти в документации **Microsoft**.

Осталось определить, откуда файл **XAML** знает о том, какие объекты можно использовать в разметке.

Как и в языке **C#**, для того чтобы использовать какие-либо типы, надо указать на пространство имён, в котором эти типы определены. В **C#** для этого используется ключевое слово **using** (только при указании в начале кода).

В **XAML** пространство имён указывается как значение атрибута **xmlns** (можно читать это как *XML Namespace*).

Увидеть эти атрибуты вы можете в любом файле **XAML**.



```
<Window x:Class="TestProgWPF.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:TestProgWPF"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="400">
    <Grid>
    </Grid>
</Window>
```

Рисунок 18

В редакторе **VisualStudio** неиспользуемые пространства имён отображаются более бледными, чем обычный текст. Всё по аналогии с синтаксисом **C#**.

Однако названия базовых пространств имён оформлены в виде ссылки на сайт. На самом деле, сайтов с таким **URI** не существует. Это сделано для поддержания принципа единственности указания на ресурсы. Базовые пространства имён **WPF** вне зависимости от места разработки всегда будут одинаковы.

Для доступа к объектам из определённого пространства имён, этому пространству присваивается краткое имя - префикс.

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

Рисунок 19

При необходимости, можно подключить собственное пространство имён.

В общем, XAML представляет удобную древовидную структуру. Привыкнув к ней однажды вы поймёте, что это очень удобно!

☺ **Документация WPF:** <https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/advanced/xaml-in-wpf?view=netframeworkdesktop-4.8>