XAML

Extensible Application Markup Language

1) WPF не нуждается в XAML
2) XAML не нуждается в WPF

Код на XAML

```
<Button x:Name="submitButton" FontFamily="Consolas" FontSize="20" Foreground="Green">
        Submit
</Button>
```

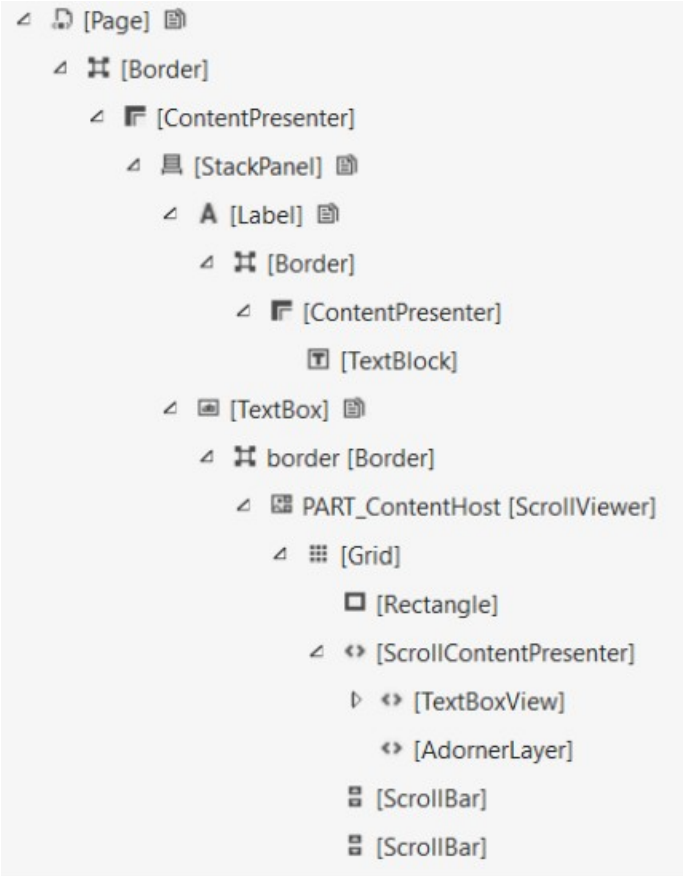
Та же кнопка в C#

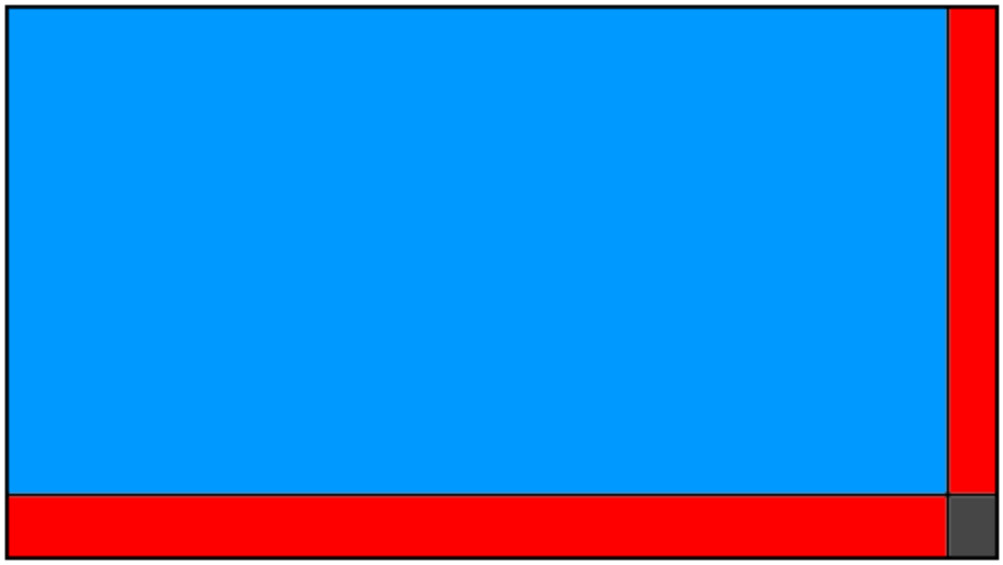
```
var submitButton = new Button
{
        Content = "Submit",
        FontFamily = new FontFamily("Consolas"),
        FontSize = 20.0,
        Foreground = Brushes.Green
};
```

Всё, что может XAML можно сделать используя программный код.

Деревья пользовательского интерфейса

```
<Page>

        <StackPanel Orientation="Horizontal">

                <Label>Message:</Label>

                <TextBox Width="100"/>

        </StackPanel>

</Page>
```





События и команды

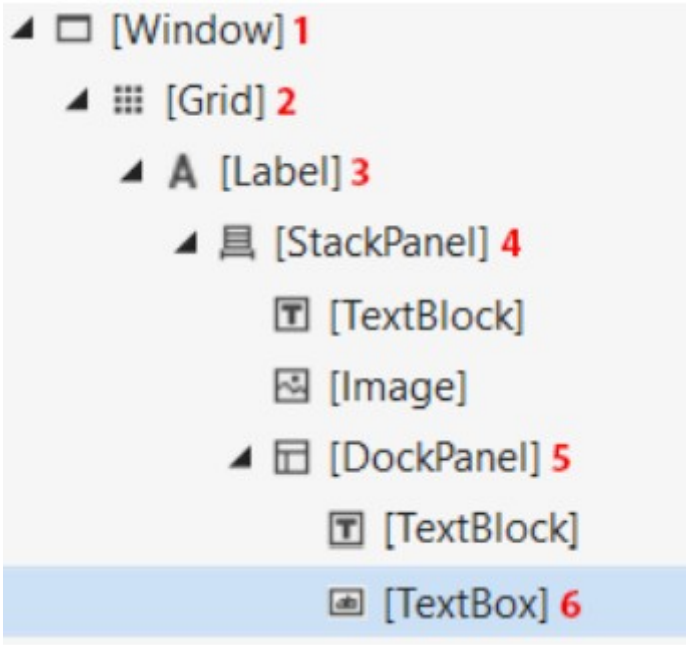
События связаны с визуальным деревом. Они возникают в конкретных элементах пользовательского интерфейса, при взаимодействии с устройствами ввода. Например, клик мыши по элементу или нажатие клавиши клавиатуры в объекте клавиатурного фокуса.

Для того, чтобы не прикреплять обработку событий к каждому конкретному элементу визуального дерева, существует механизм маршрутизации событий.

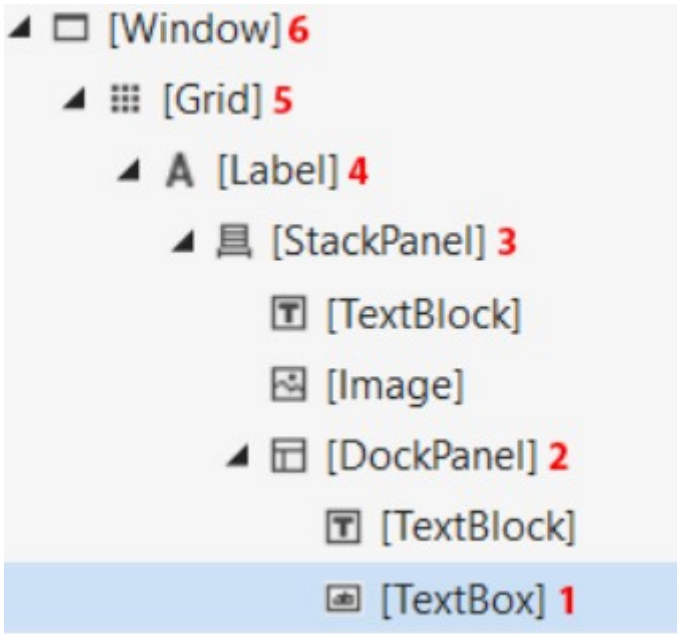
Он позволяет обрабатывать события предками элемента для которого, оно возникло.

Два вида маршрутизации:

туннелирование (tunneling) – событие начинается с корневого элемента и направляется по дочерним, пока не дойдёт до того, которому оно предназначено.



всплывание (bubbling) – наоборот, начинается с элемента, для которого предназначено и всплывает вверх по дереву, пока не найдёт готовый его обработать элемент.



Событие обычно существуют попарно. Есть предварительное событие (preview), оно туннелируется, и есть основное событие, оно всплывает.

События связаны с низкоуровневым взаимодействием — клик мыши, нажатие клавиши на клавиатуре, и т.п.

Но часто требуется более высокий уровень абстракции. Например, нажатие сочетания клавиш Ctrl+P, нажатие кнопки «Печать» и выбор пункта меню «Файл» → «Печать» должен в

любом случае вызывать окно печати на принтер. WPF предоставляет механизм команд, который даёт доступ к такому уровню абстракции.

Элементы управления

Lookless

Элементы управления в WPF не знают, как выглядят. Они не содержат этой информации.

Их вид задаётся с помощью другой сущности — шаблона.

Примитивные элементы

Большинство элементов, которые могут быть отображены на экране, не являются элементами управления.

Существует класс System.Windows.Controls.Control – от него наследуются все элементы управления. Элементы управления требуют наличия интерактивного поведения: кнопки, списки, поля для ввода текста, полосы прокрутки и т.д.

Классы же описывающие такие вещи, как базовый прямоугольник, базовый эллипс, изображения и т.п. являются примитивными и наследуются от более базового System.Windows.FrameworkElement. Более того, Control тоже наследуется от FrameworkElement.

Только наследники Control имеют шаблон, описывающий их визуализацию.

Наследники FrameworkElement не имеют шаблона, а обладают собственной визуализацией.

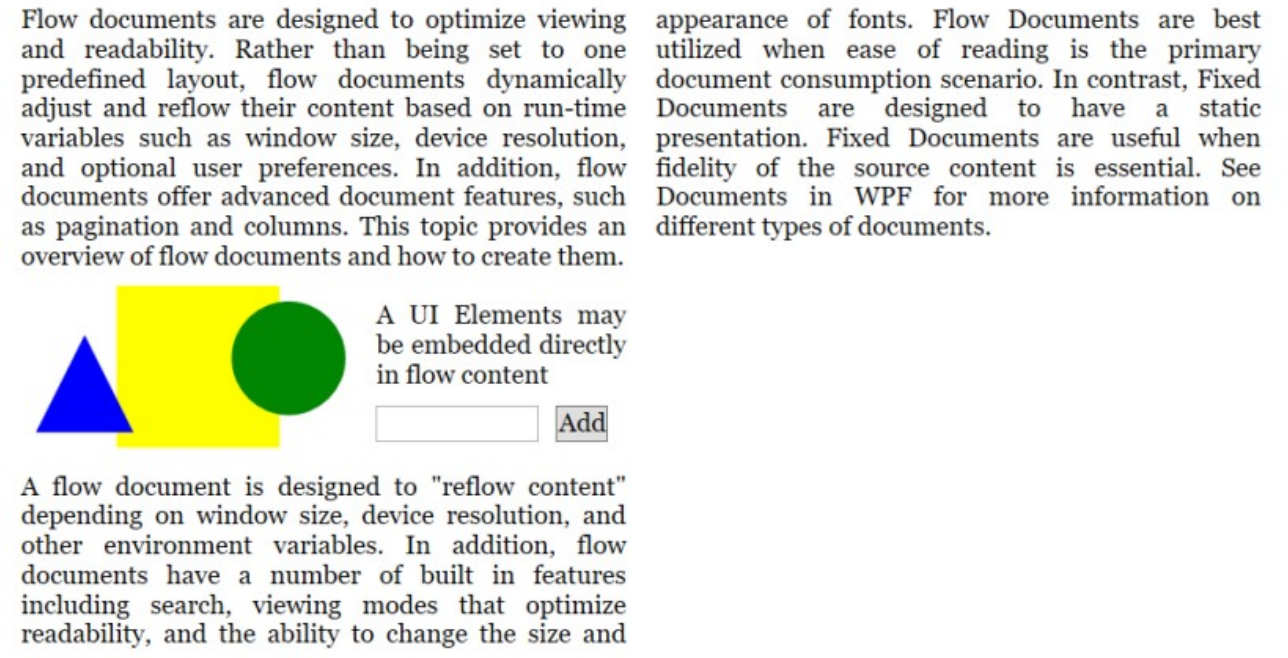
Панели

Панель — это элемент, который может быть добавлен в визуальное дерево и содержать несколько дочерних элементов. Задача панели — это определение местоположения дочерних элементов.

Видов панелей несколько — от простых, который позволяют позиционировать элементы точно по их координатам, или выстраивать в горизонтальные или вертикальные ряды, до более сложных, которые позволяют компоновать объекты в виде сетки.

Документы нефиксированного формата

WPF позволяет оформлять документы, которые подстраивают своё содержимое под доступное им пространство, с учётом заданных правил форматирования.



Можно интегрировать текст в пользовательский интерфейс, а можно интегрировать пользовательский интерфейс в текст.

WPF предоставляет возможности обработки текста, сопоставимые с HTML. Можно использовать разные виды форматирования текста, таблицы, списки, нумерованные и ненумерованные, и так далее. Также есть возможность разбить содержимое на страницы и колонки, а также адаптировать текст под доступную рабочую область.

Обзор XAML

XAML позволяет описать элементы пользовательского интерфейса с помощью языка разметки, поместив описание поведение в отдельные файлы с кодом.

Такое отделение разметки от поведения называется code-behind.

Чувствителен к регистру.

Alice != alice != aLiCe

Элемент

Предназначены для описания экземпляров классов. После того, как разметка будет обработана XAML-анализатором, будут созданы объекты соответствующих типов.

Для их создания применяется конструктор по умолчанию.

<TypeName>Content</TypeName>

Описание состоит из открывающего тэга, закрывающего тэга и содержимого.

TypeName – это имя элемента, которое является по сути названием класса.

После имени могут располагаться атрибуты.

Содержимым может быть обычный текст, или другие элементы.

Обычно содержимое строго описано типом данных элемента.

Если элемент не имеет содержимого, то можно ничего не писать между тэгами, или же записать его в виде самозакрывающиегося тэга:

<TypeName></TypeName>

<TypeName/>

Атрибут

<TypeName PropertyName="Value">Content</TypeName>

<TypeName PropertyName="Value"/>

<TypeName PropertyName1="Value1" PropertyName2="Value2"/>

<Button Height="30" Width="100">Yes</Button>

var button = new Button { Content = "Yes", Height = 30.0, Width = 100.0 };

<TypeName EventName="HandlerName"/>

<Button Click="Button_Click">OK</Button>

Атрибут нельзя описывать более одного раза.

~~<Button Height="100" Height="50">OK</Button>~~

Элемент-свойство

```
<TypeName>
    <TypeName.PropertyName>
        <PropertyTypeName InnerProperty=”Value”/>
    </TypeName.PropertyName>
</TypeName>
```

SolidColorBrush TextBox.Background

```
<TextBox FontFamily=”Consolas” FontSize=”20” Foreground=”Yellow”>
    <TextBox.Background>
        <SolidColorBrush Color=”Aqua” Opacity=”0.5”/>
    </TextBox.Background>
</TextBox>
```

```
var textBox = new TextBox
{
```

```
        Background = new SolidColorBrush{Color = Colors.Aqua, Opacity= 0.5},
        FontFamily = new FontFamily(“Consolas”),
        FontSize = 20.0,
        Foreground = Brushes.Yellow
};
<TextBox FontSize=”20”/>
<TextBox>
        <TextBox.FontSize>20</TextBox.FontSize>
</TextBox>

<TextBox Foreground=”Red”>
        <TextBox.Foreground>Red</TextBox.Foreground>
</TextBox>
```

Свойство-коллекция

```
<TypeName>
        <TypeName.CollectionPropertyName>
                <CollectionName>
                        <Item1/>
                        <Item2/>
                        <Item3/>
                </CollectionName>
        </TypeName.CollectionPropertyName>
</TypeName>
```

Обычно применяется сокращённая запись:

```
<TypeName>
        <TypeName.CollectionPropertyName>
                <Item1/>
                <Item2/>
                <Item3/>
```

```
        </TypeName.CollectionPropertyName>
</TypeName>


<TextBox FontFamily="Consolas" FontSize="20" Foreground="Yellow">
        <TextBox.Background>
                <LinearGradientBrush>
                        <LinearGradientBrush.GradientStops>
                                <GradientStopCollection>
                                        <GradientStop Color="Red" Offset="0.0"/>
                                        <GradientStop Color="Pink" Offset="1.0"/>
                                </GradientStopCollection>
                        </LinearGradientBrush.GradientStops>
                </LinearGradientBrush>
        </TextBox.Background>
</TextBox>


<TextBox FontFamily="Consolas" FontSize="20" Foreground="Yellow">
        <TextBox.Background>
                <LinearGradientBrush>
                        <LinearGradientBrush.GradientStops>
                                <GradientStop Color="Red" Offset="0.0"/>
                                <GradientStop Color="Pink" Offset="1.0"/>
                        </LinearGradientBrush.GradientStops>
                </LinearGradientBrush>
        </TextBox.Background>
</TextBox>
```

Свойство-коллекция должно принадлежать к типу, который реализует System.Collections.IList.

Свойство «содержимое»

XAML позволяет определить одно свойство описываемого им класса, как содержимое, которому присваивается содержимое между тэгами. Чаще всего называется Content.

Свойством «содержимым» Border является его свойство Child.

```
<Border>
    <Border.Child>
        <TextBlock Text="Hello"/>
    </Border.Child>
</Border>
```

```
<Border>
    <TextBlock Text="Hello"/>
</Border>
```

Если у объекта есть элементы-свойства, они должны быть все до или после содержимого.

```
<Button>
    <Button.Background>Red</Button.Background>
    Some long content for button 1
</Button>
```

```
<Button>
    Some long content for button 2
    <Button.Background>Red</Button.Background>
</Button>
```

~~<Button>~~
~~Some long content~~
~~<Button.Background>Red</Button.Background>~~
~~for button 3~~
~~</Button>~~

```
<StackPanel>
    <Button>Button 1</Button>
    <Button>Button 2</Button>
```

</StackPanel>

Присоединённые свойства

<TypeName AnotherTypeName.PropertyName=”Value”/>

```
<Grid>
    <TextBox Grid.Column=”0” Grid.Row=”1”/>
</Grid>
```

```
var textBox = new TextBox();
var grid = new Grid();
grid.Children.Add(textBox);
Grid.SetColumn(textBox, 0);
Grid.SetColumn(textBox, 1);
```

TypeName.PropertyName=”Value” → TypeName.SetPropertyName(obj, Value)

```
<Grid>
    <TextBox>
        <Grid.Column>0</Grid.Column>
        <Grid.Row>0</Grid.Row>
    </TextBox>
</Grid>
```

Преобразователь типа

```
<Button Content=”Submit”>
    <Button.BorderThickness>
        <Thickness Bottom=”10” Left=”15” Right=”10” Top=”5”/>
    </Button.BorderThickness>
</Button>
```

<Button Content=”Submit” BorderThickness=”15,5,10,10”/>

Расширение разметки

Все расширения разметки реализованы, как отдельные классы, которые наследуют от System.Windows.Markup.MarkupExtension.

ProvideValue

<TypeName PropertyName=”{ExtensionName Argument}”/>

<Button Style=”{StaticResource OkButtonStyle}”>OK</Button>

StaticResource – это класс StaticResourceExtension.

Эти классы принято называть так, чтобы они оканчивались суффиксом Extension. Этот суффикс опускается в XAML.

Если у класса расширения разметки есть ещё какие-то свойства:

<TypeName PropertyName=”{ExtensionName Argument, PropertyName=Value}”/>

Если у класса расширения разметки есть конструктор по умолчанию:

<TypeName PropertyName=”{ExtensionName, PropertyName=Value}”/>

Если нет дополнительных свойств и есть конструктор по умочанию, можно указать только название расширения:

<TypeName PropertyName=”{ExtensionName}”/>

Корневой элемент и пространства имён

Корневой элемент чаще всего <Window>, <Page>, <UserControl>, <ResourceDictionary>, <Application>.

<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

</Window>

Допустим есть сборка Controls.dll

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

xmlns:controls="clr-namespace:ProgressBars;assembly=Controls"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <controls:RoundProgressBar/>

</Window>

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

xmlns:c="clr-namespace:WpfApplication.Controls"

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

    <c:FancyButton/>

</Window>

Code-Behind

Описывает каким образом XAML соединяется с программным кодом.

Описание одного класса разделяется на две части.

Одна часть описывает фрагмент интерфейса, который реализуется данным классом в виде разметки. Другая часть содержит программный код, описывающий логику поведения этого интерфейса.

Файл с разметкой .xaml

Файл с кодом .cs

Ограничения:

- Описание частичного класса должно быть наследником класса, который соответствует корневому элементу разметки.

- Методы, которые обрабатывают события, не должны быть статическими.

- Сигнатура метода, который обрабатывает событие должна совпадать с делегатом этого события.

Name и x:Name

x:Name означает, что необходимо создать поле класса в файле с кодом.

Name хранится логическое имя элемента визуального дерева. Оно является псевдонимом для x:Name, но они принадлежат разным технологиям, поэтому они существуют отдельно.

Базовые классы элементов визуального дерева

DispatcherObject

System.Windows.Threading.DispatcherObject

Все объекты с привязкой к потоку, унаследованы от этого класса.

DependencyObject

System.Windows.DependencyObject

Используется для реализации системы свойств WPF. Позволяет оповещать при изменении значения, проверять на валидность, привязать к стилям, задать значение по умолчанию и т. д.

Visual

System.Windows.Media.Visual

Предоставляет набор методов и свойств для отображения объекта на экране.

Также содержит функционал для проверки на визуальное попадание курсора на элемент.

UIElement

System.Windows.UIElement

Содержит свойства, методы и события для взаимодействия с пользователем. Фокус ввода, приём данных с мыши и клавиатуры, видимость, включённость и т.п.

FrameworkElement

System.Windows.FrameworkElement

Расширяет функционал элементов WPF, для реализации системы компоновки элементов, логического дерева, стилей и привязки данных. Также здесь находится набор событий, описывающих жизненный цикл элемента. Можно определить, когда элемент инициализирован, загружен или выгружен.

Panel

System.Windows.Controls.Panel

Базовый для панелей, отвечает за упорядочивание и расположение элементов внутри панелей.

Control

System.Windows.Controls.Control

Базовый класс для всех элементов управления, у которых есть шаблон, отвечающий за визуализацию.

ContentControl

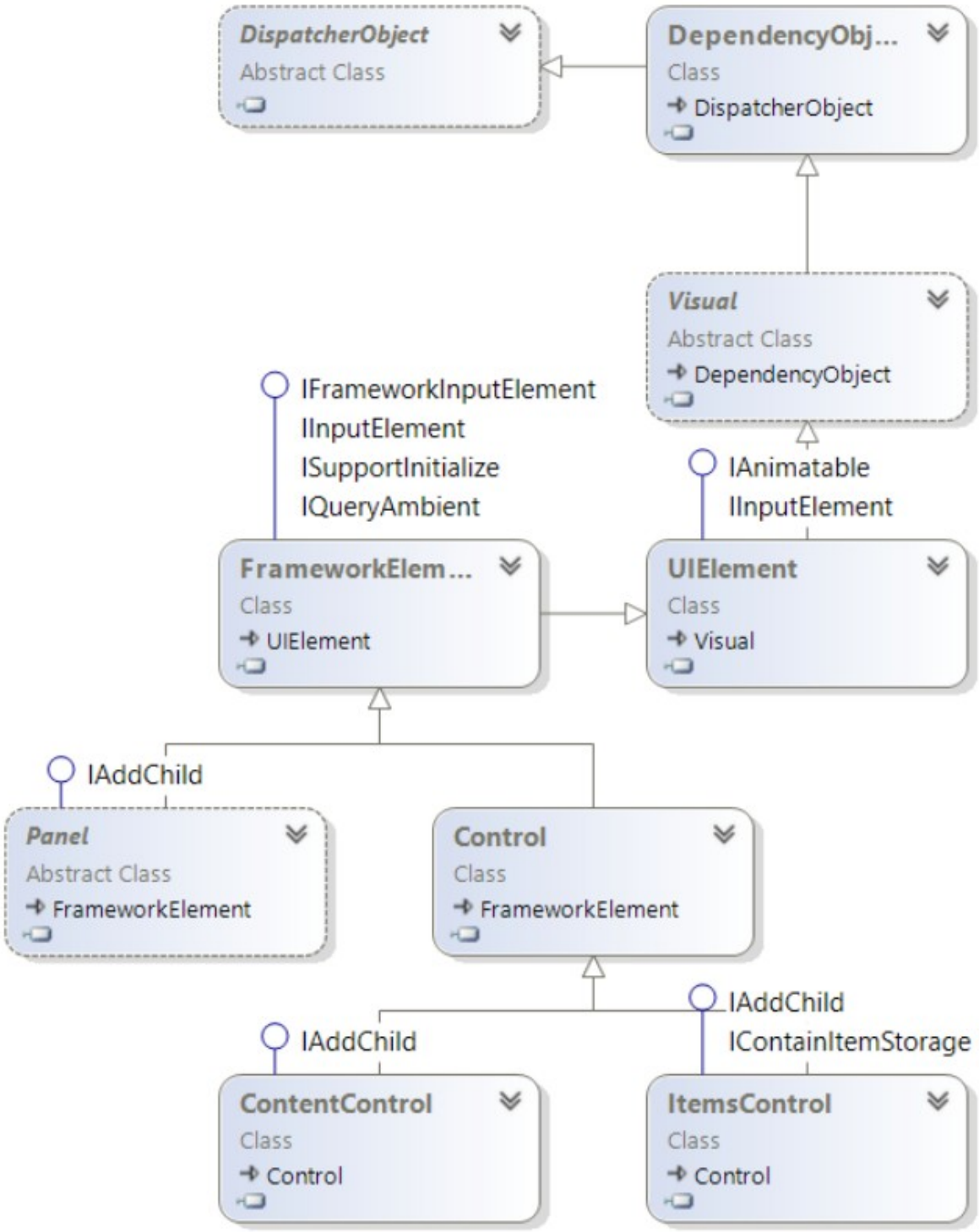
System.Windows.Controls.ContentControl

Базовый для всех элементов управления, у которых есть только один дочерний элемент.

ItemsControl

System.Windows.Controls.ItemsControl

Базовый для всех элементов управления, у которых может быть несколько дочерних элементов.


DispatcherObject
Abstract Class
DependencyObj...
Class
DispatcherObject
Visual
Abstract Class
DependencyObject
IFrameworkInputElement
IInputElement
ISupportInitialize
IQueryAmbient
IAnimatable
IInputElement
FrameworkElem...
Class
UIElement
UIElement
Class
Visual
IAddChild
Panel
Abstract Class
FrameworkElement
Control
Class
FrameworkElement
IAddChild
IAddChild
IContainItemStorage
ContentControl
Class
Control
ItemsControl
Class
Control