



Windows Presentation Foundation

С версии 1.0 платформы .NET для построения графических настольных приложений использовали два API-интерфейса под названиями **Windows Forms** и **GDI+**, упакованные в сборки **System.Windows.Forms.dll** и **System.Drawing.dll**.

С версии .NET 3.0 появился API-интерфейс –

Windows Presentation Foundation.

Windows Presentation Foundation (WPF) —

это набор библиотек, предназначенных для построения настольных приложений, который интегрирует различные API-интерфейсы в единую объектную модель и обеспечивает четкое разделение ответственностей через XAML.

photoSuru - A Microsoft Syndicated Client Sample Application

photoSuru

photoSuru > wpfphotos > Gallery Home

Search

Switch to List View

Landscapes

11.12.2008 11:48:53 | 44 Photos

Photos of beautiful landscapes from around the world.



Animals

11.12.2008 10:48:53



Up Close!

11.12.2008 10:48:53 | 44 Photos

Close up shots of everyday objects and things in nature.



Urban

11.12.2008 10:48:53 | 34 Photos



Architecture

11.12.2008 10:48:53 | 19 Photos

Interesting architecture and sculptures.



Mountains

11.12.2008 10:48:53 | 17 Photos

Gorgeous shots of mountains.



Photo Album

volcanoes


Switch to List View

Switch to List View

Landscapes

| 44 Photos


Photos of beautiful landscapes from around the world.



Up Close!

| 44 Photos


Close up shots of everyday objects and things in nature.



Architecture

| 19 Photos


Interesting architecture and sculptures.



Animals


| 30 Photos

A collection of photos of creatures that walk, crawl, slither,



Urban

| 34 Photos



Mountains

| 17 Photos

Gorgeous shots of mountains.


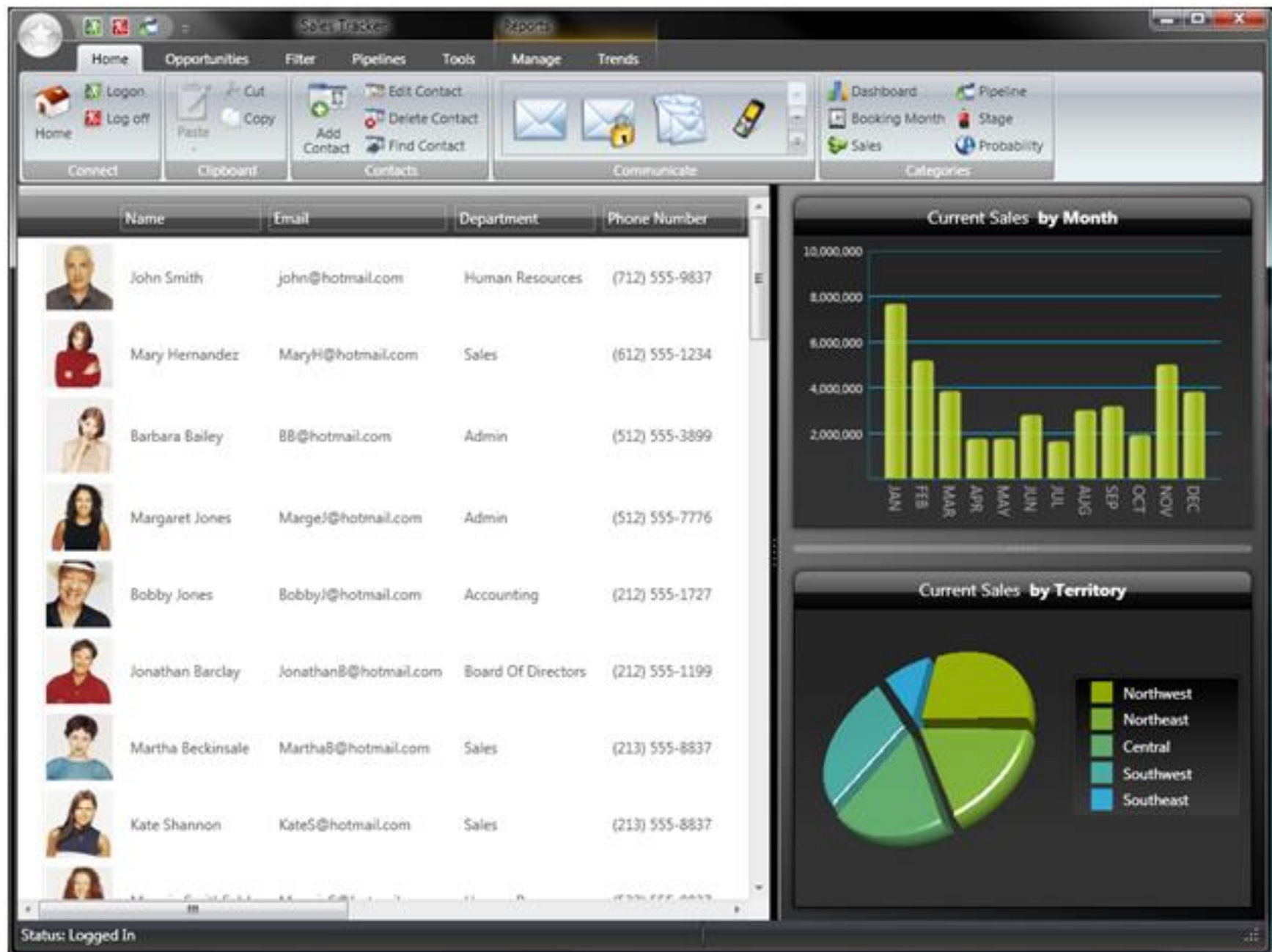


Photo Album



<http://demos.telerik.com/silverlight/salesdashboard/>



<https://www.wiki-os.org/InterfaceDesigner/InterfaceDesigner>



Решения, предшествующие WPF, для обеспечения желаемой функциональности

Желаемая функциональность	Технология
Построение окон с элементами управления	Windows Forms
Поддержка двумерной графики	GDI+ (System.Drawing.dll)
Поддержка трехмерной графики	API-интерфейсы DirectX
Поддержка потокового видео	API-интерфейсы Windows Media Player
Поддержка документов нефиксированного формата	Программное манипулирование PDF-файлами

Решения .NET 3.0 для обеспечения желаемой функциональности

Желаемая функциональность	Технология
Построение форм с элементами управления	WPF
Поддержка двумерной графики	WPF
Поддержка трехмерной графики	WPF
Поддержка потокового видео	WPF
Поддержка документов нефиксированного формата	WPF

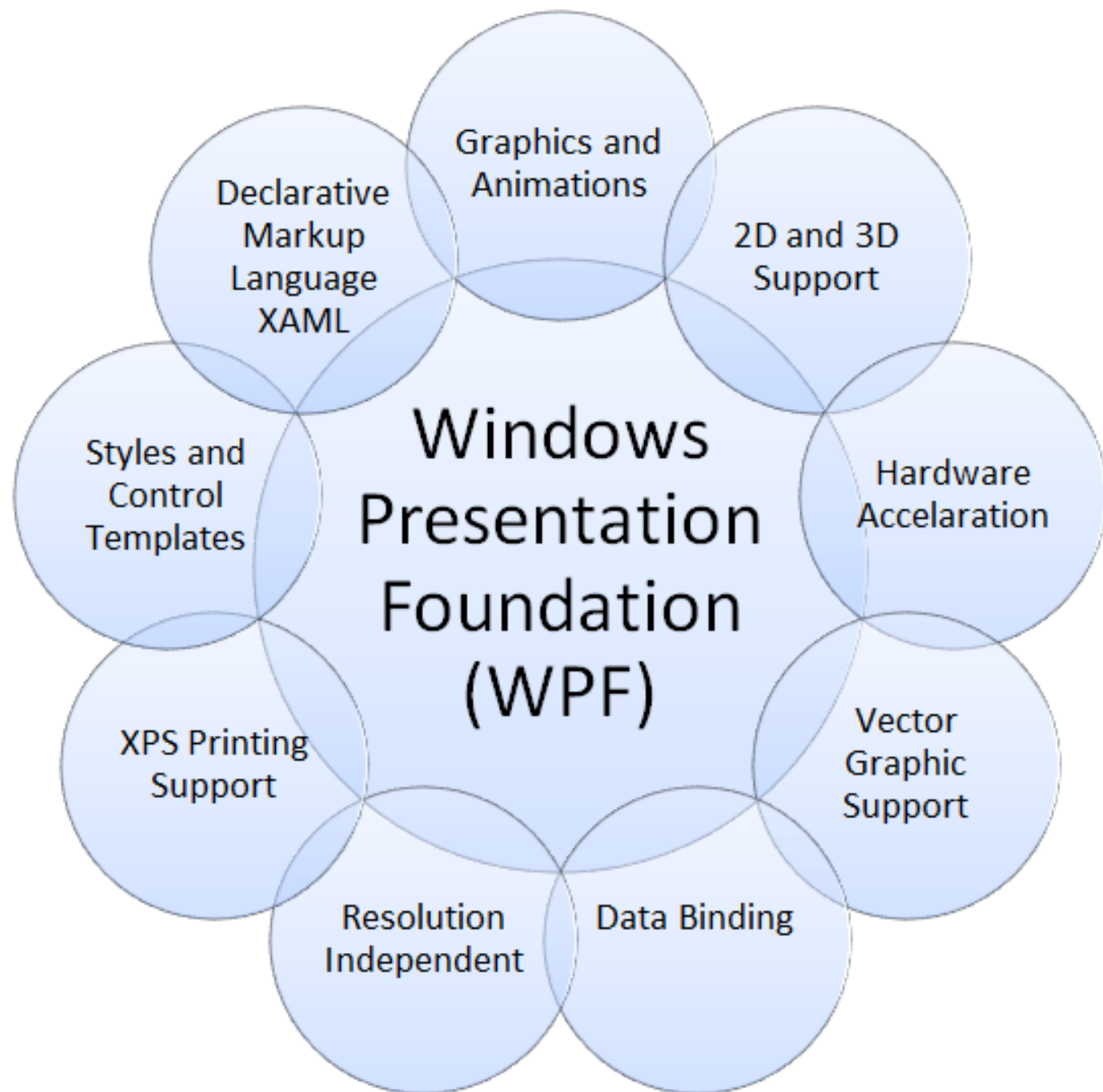
Несколько недостатков Windows Forms :

- 1) Позиции и размеры элементов управления задаются в пикселях, что приводит к риску некорректного отображения приложений на клиентах;
- 2) API - интерфейсом для рисования нестандартных элементов управления является GDI+, который, несмотря на достаточную гибкость, медленно визуализирует крупные области (и без двойной буферизации может привести к мерцанию);
- 3) У элементов управления отсутствует истинная прозрачность;
- 4) Трудно добиться надежности динамической компоновки

Преимущества WPF по сравнению с Windows Forms:

- 1) Она поддерживает развитую графику, включая произвольные трансформации, трехмерную визуализацию и истинную прозрачность.
- 2) Первичная единица измерения основана не на пикселях, поэтому приложения корректно отображаются при любой настройке DPI.
- 3) Она имеет обширную поддержку динамической компоновки, которая означает возможность локализации приложения без опасности того, что элементы будут перекрывать друг друга.
- 4) Визуализация использует DirectX и является быстрой, получая преимущества от аппаратного ускорения графики.
- 5) Пользовательские интерфейсы могут быть описаны декларативно в XAML-файлах, которые поддерживаются независимо от файлов отделенного кода, это помогает отделить внешний вид от функциональности.

WPF Version	Release (YYYY-MM)	.NET Version	Visual Studio Version	Major Features
3.0	2006-11	3.0	N/A	Initial Release. WPF development can be done with VS 2005 (released in Nov 2005).
3.5	2007-11	3.5	VS 2008	Changes and improvements in: Application model, data binding, controls, documents, annotations, and 3D UI elements.
3.5 SP1	2008-08	3.5 SP1	N/A	Native splash screen support, New WebBrowser control, DirectX pixel shader support. Faster startup time and improved performance for Bitmap effects.
4.0	2010-04	4.0	VS 2010	New controls: Calendar, DataGrid, and DatePicker. Multi-Touch and Manipulation
4.5	2012-08	4.5	VS 2012	New Ribbon control New INotifyDataErrorInfo interface
4.5.1	2013-10	4.5.1	VS 2013	No Major Change
4.5.2	2014-05	4.5.2	N/A	No Major Change
4.6	2015-07	4.6	VS 2015	Transparent child window support HDPI and Touch improvements



Подход в Winforms

Дизайнер



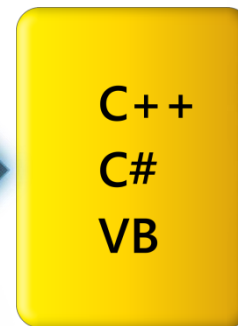
- Разные платформы
- Сложности создания интерактивного функционала

Разработчик



Select Me! 3 ▼

Select Me! 1	<input checked="" type="radio"/>
Select Me! 2	<input type="radio"/>
Select Me! 3	<input type="radio"/>
Select Me! 4	<input type="radio"/>
Select Me! 5	<input type="radio"/>
Select Me! 6	<input type="radio"/>
Select Me! 7	<input type="radio"/>
Select Me! 8	<input type="radio"/>



Select Me! 1 ▼

Select Me! 1
Select Me! 2
Select Me! 3
Select Me! 4
Select Me! 5

Подход в WPF

Дизайнер



- Одна платформа
- Интерактивный дизайн доступен изначально

Разработчик



Select Me! 3 ▾

Select Me! 1	<input checked="" type="radio"/>
Select Me! 2	<input type="radio"/>
Select Me! 3	<input type="radio"/>
Select Me! 4	<input type="radio"/>
Select Me! 5	<input type="radio"/>
Select Me! 6	<input type="radio"/>
Select Me! 7	<input type="radio"/>
Select Me! 8	<input type="radio"/>



Select Me! 3 ▾

Select Me! 1	<input checked="" type="radio"/>
Select Me! 2	<input type="radio"/>
Select Me! 3	<input type="radio"/>
Select Me! 4	<input type="radio"/>
Select Me! 5	<input type="radio"/>
Select Me! 6	<input type="radio"/>
Select Me! 7	<input type="radio"/>
Select Me! 8	<input type="radio"/>

Подход в WPF

Дизайнер



- У каждого свой удобный инструмент
- Декларативная разработка с помощью платформы XAML

Разработчик



Аппаратно-независимые единицы

Аппаратно-независимая единица (device-independent units) составляет 1/96 дюйма и используется для масштабирования окна и его элементов.

[Размер одной единицы] = 1 / 96 дюйма * [DPI системы]

Если создать кнопку с размером 96*96 то при стандартных настройках системных DPI (96 *dots per inch*) кнопка в ширину и в высоту будет 1 дюйм (количество пикселей в высоту и ширину **96**).

Если DPI системы 120 (дисплей с высоким разрешением)

[Размер одной единицы] = 1 / 96 дюйма * 120 = 1.24

Кнопка в ширину и в высоту будет 1 дюйм (количество пикселей в высоту и ширину **119**).

Различные варианты приложений WPF

- Традиционные настольные приложения
- WPF-приложения на основе навигации
- Приложения XBAP
- Отношения между WPF и Silverlight

Структура традиционного приложения

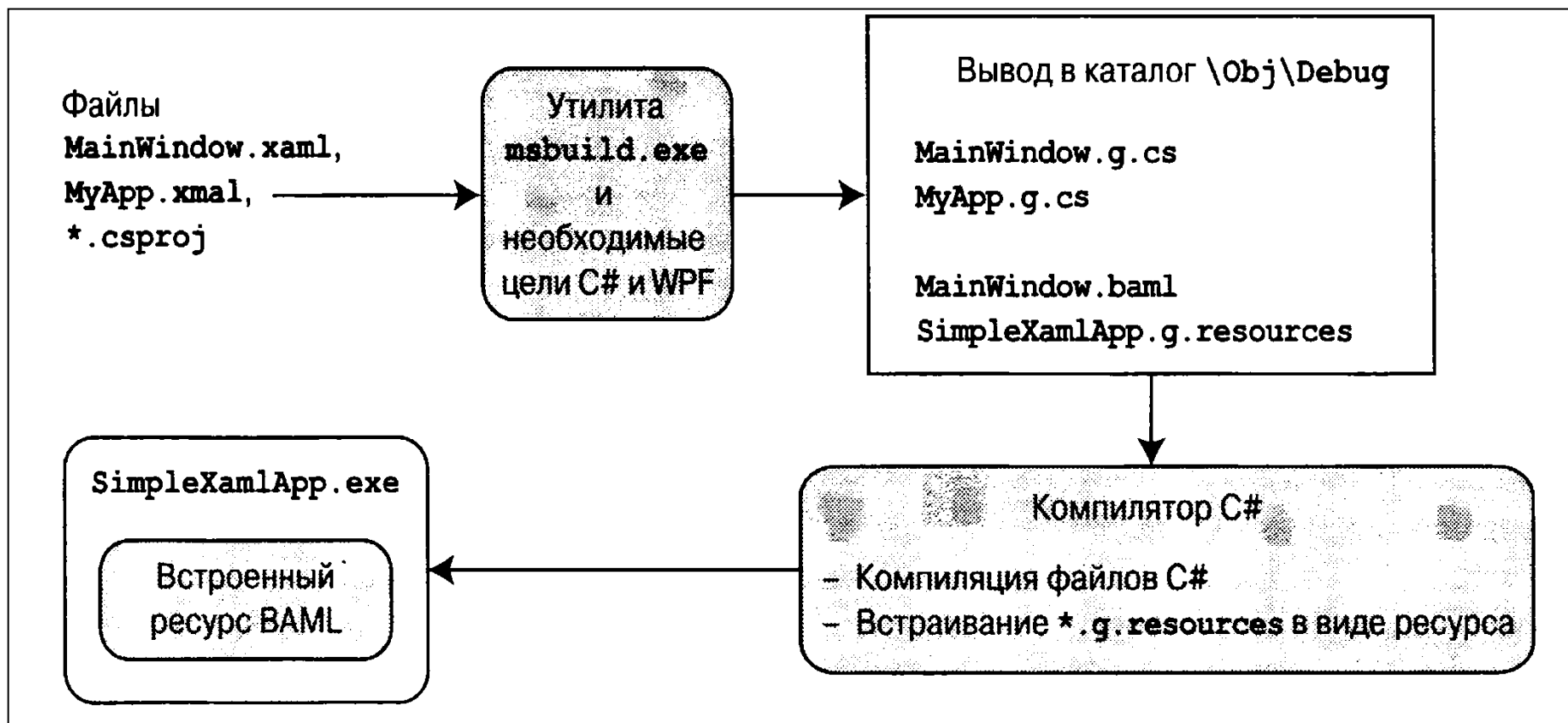
Классическое приложение WPF состоит :

- Файлов XAML, которые содержат разметку, описывающую внешний вид и поведение окна (элементы управления)
- Файлы кода C# — логику реализации.

Данные файлы неразрывно связаны между собой

Демонстрация

- Структуры проекта
- Расположение функции Main()
- Автоматически генерируемый код



Процесс трансформации XAML в сборку во время компиляции

Во время выполнения ресурс **BAML** будет извлечен из контейнера ресурсов и *использован для настройки внешнего вида всех окон и элементов управления.*

Extensible Application Markup Language (XAML) — это основанная на XML грамматика, позволяющая определять состояние (и до некоторой степени функциональность) дерева объектов .NET через разметку.



В WPF XAML используется:

- для определения элементов пользовательского интерфейса,
- привязки данных,
- поддержки событий и др. свойств.

Варианты применения XAML-документа:

1. может быть внедрен в код приложения и интерпретируется исполняющей системой **WPF**;
2. может быть отдельно скомпилирован в **BAML (Binary Application Markup Language)** и добавлен в исполняемый файл в виде ресурса.

Каждый **элемент** в документе XAML является **экземпляром одноименного класса в .NET**.

XAML: <Button></Button>

C#: Button s = new Button();

XAML допускает вложение одного элемента в другой.

Пример: Кнопка, в которой будет поле ввода. Вот так будет выглядеть данный код:

```
<Button>
    <TextBox> button1 </TextBox>
</Button>
```

Вы можете задавать свойства любого из классов через **атрибуты** например:

```
<Button Name="button1" Foreground="Red"></Button>
```

// Определение WPF-элемента Button в коде C#.

```
Button btnClickMe = new Button();
```

```
btnClickMe.Height = 40;
```

```
btnClickMe.Width = 100;
```

```
btnClickMe.Content = "Click Me";
```

<!-- Определение WPF-элемента Button в XAML -->

```
<Button Name = "btnClickMe" Height = "40"
```

```
Width = "100" Content = "Click Me" />
```

!!! конверторы

```
tbText2.Text = "Текст 2"; // текст
tbText2.Foreground = new SolidColorBrush(Colors.Red); // цвет шрифта
tbText2.Background = new SolidColorBrush(Colors.Beige); // цвет фона
tbText2.BorderBrush = new SolidColorBrush(Colors.Aqua); // цвет границы
tbText2.BorderThickness = new Thickness(1, 2, 3, 4); // толщина границы
tbText2.Padding = new Thickness(4,8,4,8); // внутренние отступы
tbText2.HorizontalAlignment = HorizontalAlignment.Center; // выравнивание по горизонтали
tbText2.VerticalAlignment = VerticalAlignment.Top; // выравнивание по вертикали

tbText2.FontFamily = new FontFamily("Arial"); // тип шрифта
tbText2.FontSize = 12; // размер шрифта
tbText2.FontStretch = FontStretches.ExtraCondensed; // степень сжатия/расширения шрифта
tbText2.FontWeight = FontWeights.Bold; // толщина шрифта

tbText2.IsTabStop=false; // включен ли элемент при переходе по Tab
tbText2.TabIndex=2; // № п/п элемента при переходе по Tab
tbText2.Height=50; // высота
tbText2.Width=150; // ширина
tbText2.MaxHeight=50; // максимальная высота
tbText2.MinHeight = 30; // минимальная высота
tbText2.MaxWidth = 150; // максимальная ширина
tbText2.MinWidth = 50; // минимальная ширина
tbText2.Visibility=Visibility.Visible; // видимость элемента
tbText2.Focusable=true; // поддержка фокуса
tbText2.IsEnabled = true; // доступность элемента
```

```
<TextBox x:Name="tbText1" Text="Текст 1"
          Foreground="Red" Background="Beige"
BorderBrush="Aqua"
          BorderThickness="1,2,3,4" Padding="4,8"
HorizontalAlignment="Center" VerticalAlignment="Top"
          FontFamily="Arial" FontSize="12" FontWeight="Bold"
FontStretch="ExtraCondensed"
          IsTabStop="False" TabIndex="1"
          Height="50" Width="150"
          MaxHeight="50" MinHeight="30"
          MaxWidth="150" MinWidth="50"
          Visibility="Visible"
          Focusable="True"
          IsEnabled="True"/>
```



Вложенные элементы

```
<Button Margin="90,120,0,0"  
        Width="150"  
        Height="40"  
        Name="button1"  
        Content="Hello Button">
```



```
<Button.Background>  
  <LinearGradientBrush StartPoint="0,1" EndPoint="1,0">  
    <GradientStop Color="Beige" Offset="0"></GradientStop>  
    <GradientStop Color="Coral" Offset="1"></GradientStop>  
  </LinearGradientBrush>  
</Button.Background>  
</Button>
```

```
button1.Content = "Hello Button";  
GradientStopCollection coll = new GradientStopCollection();  
coll.Add(new GradientStop { Color = Colors.YellowGreen, Offset = 0 });  
coll.Add(new GradientStop { Color = Colors.Yellow, Offset = 1 });  
button1.Background = new LinearGradientBrush(coll);
```


Вложенные элементы

```
<!-- Использование различных синтаксисов для
инициализации свойства Content -->
    <Button>КНОПКА №1</Button>
    <Button Content="КНОПКА №2"></Button>
<!-- Синтаксис "свойство-элемент" -->
    <Button>
<!-- инициализация свойства Content -->
    <Button.Content>    КНОПКА 3    </Button.Content>
    <Button.Height>    50            </Button.Height>
<!-- инициализация свойства Background -->
    <Button.Background>
        <!-- использование класс LinearGradientBrush-->
        <LinearGradientBrush>
            <LinearGradientBrush.GradientStops>
                <GradientStop Color="Bisque" Offset="0"/>
                <GradientStop Color="Red"      Offset="1"/>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Button.Background>
</Button>
<!-- свойства кнопки -->
    <Button x:Name="button1"></Button>
```

Пространства имен

```
<!-- объявление класса MainWindow на основе класса Window -->
<Window x:Class="Example1.MainWindow"
<!-- пространство имен отображает множество пространств имен .NET для
использования (System.Windows, System.Windows.Controls ... ) -->
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

<!-- используется для включения специфичных для XAML “ключевых слов”
вместе с пространством имен System.Windows.Markup -->
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
<!-- установка свойств окна -->
Title="MainWindow" Height="350" Width="525">
<!-- определение контейнера компоновки (Grid, StackPanel и т.д.)-->
  <Grid>
    <!-- Определение WPF-элемента Button в XAML -->
    <Button x:Name="btnExitApp" Width="133" Height="24"
      Content = "Close Window" Click ="btnExitApp_Clicked"/>
  </Grid>
</Window>
```

Ключевое слово XAML	Описание
<code>x:Array</code>	Представляет в XAML тип массива .NET
<code>x:ClassModifier</code>	Позволяет определять видимость типа класса (<code>internal</code> или <code>public</code>), обозначенного ключевым словом <code>Class</code>
<code>x:FieldModifier</code>	Позволяет определять видимость члена типа (<code>internal</code> , <code>public</code> , <code>private</code> или <code>protected</code>) для любого именованного элемента корня (например, <code><Button></code> внутри элемента <code><Window></code>). <i>Именованный элемент</i> определяется с использованием ключевого слова <code>Name</code> в XAML
<code>x:Key</code>	Позволяет устанавливать значение ключа для элемента XAML, которое должно быть помещено в элемент словаря
<code>x:Name</code>	Позволяет указывать сгенерированное C# имя заданного элемента XAML
<code>x:Null</code>	Представляет ссылку <code>null</code>
<code>x:Static</code>	Позволяет ссылаться на статический член типа
<code>x:Type</code>	XAML-эквивалент операции <code>typeof</code> языка C# (она будет выдавать <code>System.Type</code> на основе указанного имени)
<code>x:TypeArguments</code>	Позволяет устанавливать элемент как обобщенный тип с определенным параметром типа (например, <code>List<int></code> или <code>List<bool></code>)

Расширяющая разметка

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
xmlns:system="clr-namespace:System;assembly=mscorlib"
```

```
<!-- Расширение разметки Static позволяет получать  
                               значения статических членов класса -->  
<Label Content="{x:Static system:Environment.OSVersion}" />  
<Label Content="{x:Static system:Environment.ProcessorCount}" />
```

```
<!-- Расширение разметки Type - это XAML версия оператора typeof -->  
<Label Content="{x:Type Button}" />  
<Label Content="{x:Type system:Boolean}" />
```

```
<ListBox>  
  <ListBox.ItemsSource>  
    <x:Array Type="system:String">  
      <system:String>Первый элемент</system:String>  
      <system:String>Второй элемент</system:String>  
      <system:String>Третий элемент</system:String>  
    </x:Array>  
  </ListBox.ItemsSource>  
</ListBox>
```

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
xmlns:my="clr-namespace:Example5ExtendMarkup"
```

```
<Button x:Name="btnDemo"  
        Content="{x:Static my:TextInfo.Test}"  
        Margin="150,120,150,120"  
        x:FieldModifier="public">  
</Button>
```

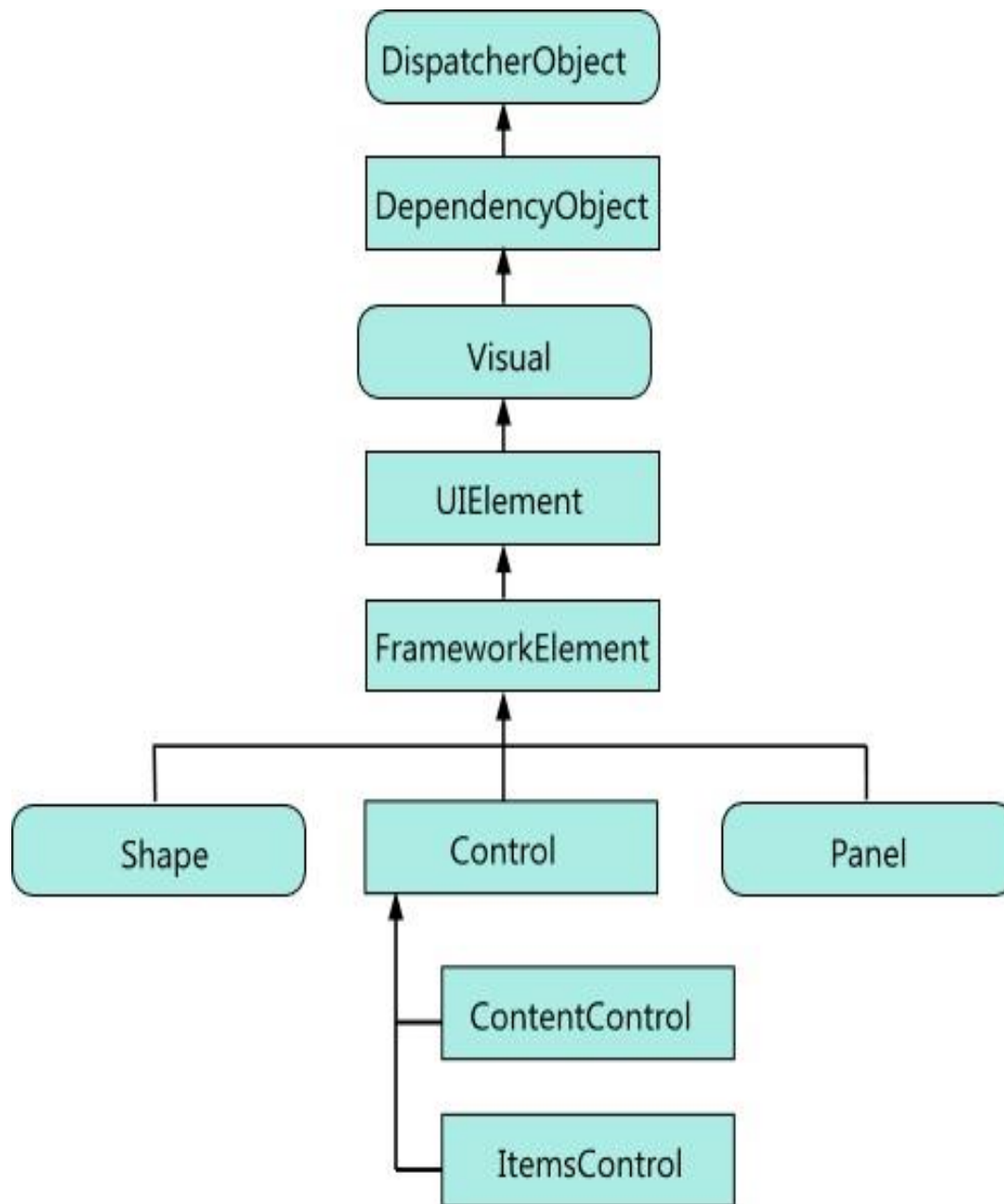
```
public static class TextInfo  
{  
    public static string Test { get; set; }  
  
    static TextInfo()  
    {  
        Test = "Тестовая строка";  
    }  
}
```


Основные сборки WPF

PresentationCore.dll	В этой сборке определены многочисленные пространства имен, образующие фундамент уровня графического пользовательского интерфейса в WPF.
PresentationFramework.dll	Эта сборка содержит большинство элементов управления WPF, классы Application и Window, поддержку интерактивной двухмерной графики и многочисленные типы, применяемые для привязки данных
System.Xaml.dll	Эта сборка предоставляет пространства имен, которые позволяют программно взаимодействовать с документами XAML во время выполнения.
WindowsBase.dll	В этой сборке определены типы, образующие инфраструктуру WPF. Сюда входят типы потоков WPF, типы безопасности, различные преобразователи типов и поддержка свойств зависимости и маршрутизируемых событий.

Основные пространства имен WPF

System.Windows	Основное (содержит классы Window, Application).
System.Windows.Controls	Содержит все ожидаемые графические элементы WPF
System.Windows.Data	Содержит типы для работы с механизмом привязки данных WPF, а также для поддержки шаблонов привязки данных
System.Windows.Markup	Содержит типы, обеспечивающие программный анализ и обработку разметки XAML
System.Windows.Media	Содержит типы для работы с анимацией, визуализацией трехмерной графики, визуализацией текста и прочие мультимедийные примитивы
System.Windows.Navigation	Содержит типы для обеспечения логики навигации, используемой браузерными приложениями XAML (XBAP), а также настольными приложениями, требующими страничной навигационной модели
System.Windows.Shapes	Содержит типы, которые позволяют визуализировать двухмерную графику, автоматически реагирующую на ввод с помощью мыши



класс Application

глобальный экземпляр работающего приложения WPF

Свойство	Описание
Current	Это статическое свойство позволяет получить доступ к работающему объекту Application из любого места кода. Это может быть очень полезно, когда обычному или диалоговому окну необходим доступ к объекту Application, который его создал, обычно для взаимодействия с переменными или функциональностью уровня приложения
MainWindow	Это свойство позволяет программно получать и устанавливать главное окно приложения
Properties	Это свойство позволяет устанавливать и получать данные, доступные через все аспекты приложения WPF (окна, диалоговые окна и т.п.)
StartupUri	Это свойство получает или устанавливает URI, который указывает окно или страницу для автоматического открытия при запуске приложения
Windows	Это свойство возвращает тип WindowCollection, который предоставляет доступ ко всем окнам, которые созданы в потоке, создавшем объект Application. Это может весьма пригодиться, когда необходимо выполнить итерацию по всем открытым окнам приложения и изменить их состояние (например, свернуть все окна)

Свойство ShutdownMode

Имя	Описание
OnLastWindowClose	Поведение по умолчанию — приложение выполняется до тех пор, пока существует хотя бы одно открытое окно. После закрытия главного окна свойство <code>Application.MainWindow</code> по-прежнему ссылается на объект, представляющий закрытое окно. (Дополнительно можно использовать код для переназначения свойства <code>MainWindow</code> , чтобы оно указывало на другое окно.)
OnMainWindowClose	Это традиционный подход — приложение остается активным только пока открыто главное окно
OnExplicitShutdown	Приложение не завершается (даже если все окна закрыты), пока не будет вызван метод <code>Application.Shutdown()</code> . Такой подход может быть оправдан, если приложение является интерфейсом для долго выполняющейся задачи. Также он применяется, если для принятия решения о закрытии приложения должна использоваться более сложная логика (в этом случае будет вызываться метод <code>Application.Shutdown()</code>)

Демонстрация

- Свойство Properties
- Свойство MainWindow, Windows
- Создание элемента в коде

класс Window

Представляет одиночное окно, которым владеет производный от Application класс, включая все диалоговые окна, отображаемые главным окном.

класса System.Windows.Controls.ContentControl

Этот базовый класс обеспечивает производные типы способностью размещать в себе данные (в том числе другой элемент управления) через свойство Content.

```
<Button Height="80" Width="100">  
    <Button.Content>  
        <StackPanel>  
            <Ellipse Fill="Red" Width="25" Height="25"/>  
            <Label Content ="OK!"/>  
        </StackPanel>  
    </Button.Content>  
</Button>
```

класс `System.Windows.Controls.Control`

Этот базовый класс предоставляет множество членов, незаменимых для обеспечения функциональности пользовательского интерфейса.

Член	Описание
<code>Background</code> , <code>Foreground</code> , <code>BorderBrush</code> , <code>BorderThickness</code> , <code>Padding</code> , <code>HorizontalContentAlignment</code> , <code>VerticalContentAlignment</code>	Эти свойства позволяют устанавливать базовые настройки, касающиеся того, как элемент управления будет визуализироваться и позиционироваться
<code>FontFamily</code> , <code>FontSize</code> , <code>FontStretch</code> , <code>FontWeight</code>	Эти свойства управляют настройками шрифтов
<code>IsTabStop</code> , <code>TabIndex</code>	Эти свойства используются для установки порядка обхода элементов управления в окне по нажатию <code><Tab></code>
<code>MouseDoubleClick</code> , <code>PreviewMouseDoubleClick</code>	Эти события обрабатывают двойной щелчок на виджете
<code>Template</code>	Это свойство позволяет получать и устанавливать шаблон элемента, который может быть использован для изменения вывода визуализации виджета

класс `System.Windows.FrameworkElement`

Этот базовый класс предоставляет множество низкоуровневых членов, которые используются повсюду в WPF (привязки данных, возможности именования членов через свойство `Name`, получения ресурсов, установки общих измерений)

Член	Описание
<code>ActualHeight</code> , <code>ActualWidth</code> , <code>MaxHeight</code> , <code>MaxWidth</code> , <code>MinHeight</code> , <code>MinWidth</code> , <code>Height</code> , <code>Width</code>	Управляют размерами производного типа
<code>ContextMenu</code>	Получает или устанавливает всплывающее меню, ассоциированное с производным типом
<code>Cursor</code>	Получает или устанавливает курсор мыши, ассоциированный с производным типом
<code>HorizontalAlignment</code> , <code>VerticalAlignment</code>	Управляет позиционированием типа внутри контейнера (такого как панель или окно списка)
<code>Name</code>	Позволяет назначать имя типу, чтобы обращаться к его функциональности в файле кода
<code>Resources</code>	Предоставляет доступ к любому ресурсу, определенному типом (система ресурсов WPF объясняется в главе 30)
<code>ToolTip</code>	Получает или устанавливает всплывающую подсказку, ассоциированную с производным типом

класс `System.Windows.UIElement`

Предоставляет производному типу многочисленные события, чтобы он мог получать фокус и обрабатывать входные запросы.

Член	Описание
<code>Focusable</code> , <code>IsFocused</code>	Эти свойства позволяют устанавливать фокус на заданный производный тип
<code>IsEnabled</code>	Это свойство позволяет управлять доступностью заданного производного типа
<code>IsMouseDownDirectlyOver</code> , <code>IsMouseOver</code>	Эти свойства предлагают простой способ выполнения логики проверки попадания
<code>IsVisible</code> , <code>Visibility</code>	Эти свойства позволяют работать с установкой видимости производного типа
<code>RenderTransform</code>	Это свойство позволяет устанавливать трансформацию, которая будет использована для визуализации производного типа

класс System.Windows.Media.Visual

Класс Visual предлагает основную поддержку визуализации в WPF. Для рисования данных на экране класс Visual взаимодействует с подсистемой DirectX.

класс System.Windows.DependencyObject

Для поддержки стилей, привязки данных, анимации тип должен быть порожден от базового класса DependencyObject.

класса System.Windows.Threading.DispatcherObject

Класс предоставляет базовые конструкции для работы с параллелизмом и многопоточностью.