# ОСОБЕННОСТИ РАБОТЫ С ДИЗАЙНОМ ПРИ СОЗДАНИИ WPF-ПРИЛОЖЕНИЯ

**Арташкин Евгений Павлович**

студент

Мордовский государственный университет им. Н.П. Огарева

Саранск

**Аннотация.** В данной статье рассматриваются наиболее частые проблемы и сложности при написании дизайна для WPF-приложения.

**Ключевые слова:** Microsoft Visual Studio, C#, API, WPF.

---

# FEATURES WORK WITH DESIGN FOR CREATION WPF-APPLICATIONS

**Artashkin Evgeniy Pavlovich**

student

Ogarev Mordovia State University

Saransk

**Abstract.** This article discusses the most common problems and difficulties in the design of the writing for a WPF-application.

**Keywords:** Microsoft Visual Studio, C#, API, WPF.

Чтобы приложение пользовалось успехом, необходимо, чтобы оно имело качественный и хорошо проработанный дизайн.

Для того, чтобы реализовать приложение, взаимодействующее со сторонними API, используя WPF Windows Presentation Foundation), необходимо использовать следующие технологии и библиотеки.

Технология WPF (Windows Presentation Foundation) – система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая (презентационная) подсистема в составе .NET Framework (начиная с версии 3.0. В основе WPF лежит векторная система визуализации, не зависящая от разрешения устройства вывода и созданная с учётом возможностей современного графического оборудования.

WPF предоставляет средства для создания визуального интерфейса, включая язык XAML (Extensible Application Markup Language), элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление. Графической технологией, лежащей в основе WPF, является DirectX. Производительность WPF выше, чем у GDI+ за счёт использования аппаратного ускорения графики через DirectX.

XAML представляет собой язык декларативного описания интерфейса, основанный на XML. Также реализована модель разделения кода и дизайна. Кроме того, есть встроенная поддержка стилей элементов, а сами элементы легко разделить на элементы управления второго уровня, которые, в свою очередь, разделяются до уровня векторных фигур и свойств/действий. Это позволяет легко задать стиль для любого элемента.

Awesomium – это библиотека, для интеграции браузера на базе Chromium в своё приложение. Достоинство Awesomium состоит в том, что его можно интегрировать в приложение практически любого типа, он обладает широким набором возможностей для разработчика.

Библиотека Hamburger Menu помогает просто реализовать динамическое меню приложения любой сложности. Своё название она получила благодаря кнопке, которая это меню вызывает. В WPF-приложениях нет стандартных средств, с помощью которых можно создать это меню, в отличие от UWP-приложений.

API (интерфейс прикладного программирования) (англ. application programming interface, API) — набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) или операционной системой для использования во внешних программных продуктах. Используется программистами при написании всевозможных приложений.

Приложение должно быть реализовано так, чтобы оно подходило для отображения на экране устройства с любым разрешением. Для этого в определении каждой страницы должно быть указано четыре свойства:

```
MinWidth="800" MinHeight="600"
MaxWidth="1020" MaxHeight="1080"
```

Приложение состоит из нескольких основных страниц. При загрузке каждой из них подгружается шаблон, в котором реализовано Hamburger Menu. Это делается из соображений оптимизации, для того, чтобы при каждом нажатии на кнопку в меню, оно не уничтожалось и создавалось заново.

```
<Window x:Class="GoodAdvice.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:HamburgerMenu="clr-
namespace:HamburgerMenu;assembly=HamburgerMenu"
        xmlns:local="clr-namespace:GoodAdvice"
        mc:Ignorable="d"
        Title="Good Advice"
        Icon="image/icons/title/title.ico"
        Width="1020" Height="540"
        MinWidth="800" MinHeight="600"
        MaxWidth="1020" MaxHeight="1080"
        WindowStartupLocation="CenterScreen"
        x:Name="this_">

    <Grid Name="mainGrid"
      Initialized="mainGrid_Initialized">
        <Frame x:Name="ContentFrame"
            ContentRendered="ContentFrame_ContentRendered"/>
        <HamburgerMenu:HamburgerMenu
          Name="hamMenu"
          Background="Black"
          MenuIconColor="White"
          SelectionIndicatorColor="Orange"
          MenuItemForeground="White"
          HorizontalAlignment="Left"
          Margin="0, 0, 0, 0"
          Opacity="0.65">
          <HamburgerMenu:HamburgerMenu.Content>
            <HamburgerMenu:HamburgerMenuItem      Name="item1"
Icon="image/icons/hamburgerMenu/0.ico"   Text="Главная"   SelectionCom-
```

```
mand="{Binding ElementName=this_}" Select-
ed="HamburgerMenuItem_Selected"/>
            <HamburgerMenu:HamburgerMenuItem Name="item2"
Icon="image/icons/hamburgerMenu/1.ico" Text="Текущее местоположение"
Selected="HamburgerMenuItem_Selected_1"/>
            <HamburgerMenu:HamburgerMenuItem
Icon="image/icons/hamburgerMenu/5.ico" Text="Изменить
местоположение" Selected="HamburgerMenuItem_Selected_5"/>
            <HamburgerMenu:HamburgerMenuItem Name="item3"
Icon="image/icons/hamburgerMenu/2.ico" Text="Поиск автомойки" Select-
ed="HamburgerMenuItem_Selected_2"/>
            <HamburgerMenu:HamburgerMenuItem
Icon="image/icons/hamburgerMenu/3.ico" Text="Настройки" Select-
ed="HamburgerMenuItem_Selected_3"/>
            <HamburgerMenu:HamburgerMenuItem
Icon="image/icons/hamburgerMenu/4.ico" Text="Энциклопедия" Select-
ed="HamburgerMenuItem_Selected_4"/>
        </HamburgerMenu:HamburgerMenu.Content>
    </HamburgerMenu:HamburgerMenu>
  </Grid>
</Window>
```

XAML – язык разметки. Поэтому, для того, чтобы приложение выглядело красиво, необходимо для каждого объекта на странице использовать такие свойства, как margin, padding, align и другие.

```
<Viewbox Stretch="Uniform" VerticalAlignment="Top">
        <TextBox x:Name="textBox1" IsReadOnly="True"
Text="Хотите помыть свой авто?" Background="Black" BorderThick-
```

```
ness="0" Foreground="White" FontFamily="/GoodAdvice;component/fonts/#Bebas Neue Book"/>
            </Viewbox>

            <Viewbox Stretch="Uniform" VerticalAlignment="Top">
              <TextBox x:Name="textBox2" IsReadOnly="True" Margin="0, 20, 85, 0" Text="А вдруг дождь?" Background="Black" BorderThickness="0" Foreground="White" FontFamily="/GoodAdvice;component/fonts/#Bebas Neue Book"/>
            </Viewbox>

            <Viewbox Stretch="Uniform" VerticalAlignment="Top">
              <TextBox x:Name="textBox3" IsReadOnly="True" Margin="95, 44, 0, 0" Text="Получите совет на основе" Background="Black" BorderThickness="0" Foreground="White" FontFamily="/GoodAdvice;component/fonts/#Bebas Neue Bold"/>
            </Viewbox>
```

В приложении реализовано много элементов, в том или ином случае динамически меняющих своё состояние. Например:

```
private void label2_Initialized(object sender, EventArgs e)
    {
        label2.Content = ((App)Application.Current).cityName;
    }
```

При открытии приложения программно устанавливаем:

label2.content = “Не определено”.

После определения местоположения устанавливаем:

label2.Content = ((App)Application.Current).cityName;

При реализации сервиса необходимо использовать те объекты, которые создавались бы не при запуске программы, а при определенных действиях клиента. В качестве примера можно взять RSS-ленту, показывающую прогноз погоды на несколько дней. До определения города этого объекта не существовало. Данная возможность может быть реализована с помощью свойства объекта Visibility.

```
<Grid Name="Wheather" Margin="310, 70, 10, 0" MaxHeight="120" VerticalAlignment="Top" Visibility="Collapsed">
        <Grid.Resources>
          <XmlDataProvider x:Key="rss" XPath="/rss/channel"/>
              <DataTemplate DataType="item">
                 <StackPanel>
                    Foreground="White" FontSize="18"/>
                    <TextBlock TextWrapping="Wrap" Width="520" FontSize="12" Foreground="White" Text="{Binding XPath=description}"/>
                    <Image Margin="15,10,0,0" Width="50" Height="50"
                         HorizontalAlignment="Center"
                     </StackPanel>
              </DataTemplate>
           </Grid.Resources>
```