УДК 004.432

РАЗРАБОТКА НАСТРАИВАЕМЫХ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ WPF

Дмитрий Игоревич Максимов
ассистент
timonpm@mail.ru
г. Елец

Елецкий государственный университет им. И.А. Бунина

Аннотация. В настоящее время при разработке приложений большое внимание стало уделяться дизайну, что в результате привело к появлению элементов управления с нестандартным оформлением и функционалом. Для реализации таких элементов управления используются специальные технологии программирования. В статье представлены принципы создания настраиваемого элемента управления на основе технологии Windows Presentation Foundation (WPF). Данная технология позволяет разрабатывать элементы с различным дизайном, а также добавлять дополнительную функциональность. В качестве примера в статье рассмотрена разработка панели вкладок, характерной для современных приложений.

Ключевые слова: настраиваемый элемент управления; программирование; Windows Presentation Foundation; C#; XAML.

С появлением библиотеки .NET разработчику стало доступно множество технологий для написания приложений. Одной из наиболее важных является библиотека классов для написания Windows-приложений Windows Forms. Эта библиотека позволяет использовать стандартные элементы управления для создания пользовательского интерфейса. Главным недостатком этой технологии является то, что при создании внешнего вида элементов управления она основывается на интерфейсе Windows API, который практически не поддается настройке. Таким образом, создание элемента управления с индивидуальным оформлением представляет собой трудоемкую задачу, решение которой строится на использовании низкоуровневой модели рисования.

В связи с этим, в библиотеку .NET была включена новая технология Windows Presentation Foundation (WPF), которая основана на новой модели отрисовки элементов управления под управлением библиотеки DirectX. Она позволяет применять различные графические как 2D, так и 3D возможности по оформлению интерфейса приложения, а за вывод будет отвечать графический процессор видеокарты [Андерсон, 2008, Петцольд, 2012]. В технологии WPF впервые реализована возможность отделения графического содержимого от кода. Для создания графических объектов в WPF используется специальный язык разметки XAML (Extensible Application Markup Language – расширяемый язык разметки приложений), основанный на XML [Петцольд, 2012, 3]. А программный код пишется на языках C# или Visual Basic. Также WPF позволяет писать приложения традиционно только на основе кода.

Рассмотрим основное преимущество WPF, а именно разработка элемента управления с индивидуальным оформлением и функционалом. В качестве примера будем рассматривать панель вкладок, которая в стандартном виде имеет непривлекательный вид (рис. 1). Также в ней отсутствует возможность закрытия вкладки, что зачастую требуется в разрабатываемых приложениях.

Рис. 1. Стандартный вид панели вкладок

В результате изменения дизайна и добавления новых функций панель будет иметь возможность закрывать вкладки и выбирать одну из вкладок в выпадающем списке (рис. 2).

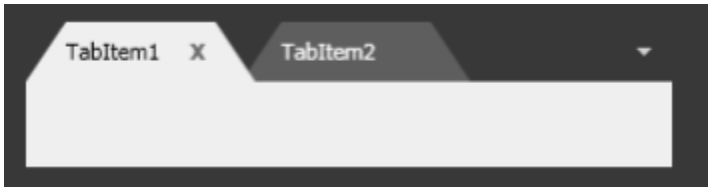

Рис. 2. Измененная панель вкладок

Для реализации данной задачи в Visual Studio создадим новый проект `ApplicationWithCustomTabControl` на языке программирования C#, имеющий тип `Приложение WPF`. Добавим в проект новый элемент управления путем вызова пункта меню `Проект → Добавить новый элемент`. В открывшемся диалоговом окне выбираем `Настраиваемый элемент управления (WPF)` и указываем имя `CustomTabControl.cs`. В результате среда программирования добавит в проект два файла:

– `CustomTabControl.cs`, в котором находится описание класса элемента управления;

– `Generic.xaml`, находящийся в папке `Themes` – шаблон оформления для элемента.

Панель вкладок представляет собой сложный элемент управления и будет состоять из нескольких составляющих:

– Сама панель `CustomTabControl`;

– Вкладки `CustomTabItem`;

– Кнопки закрытия вкладки `CustomButton`;

– Список выбора вкладки `CustomComboBox`.

Кнопка закрытия будет являться составной часть вкладки, а список выбора – панели. Рассмотрим описание стилей всех элементов. Оно будет реализовано в файле `Generic.xaml`. Начнем рассмотрение с более простых элементов, а именно кнопки и списка.

При разработке стиля элемента управления можно производить полное описание вручную, но при данном подходе есть вероятность утери некоторого стандартного функционала элемента. Поэтому лучше использовать уже разработанные шаблоны стандартных элементов и редактировать их. Получить стандартный шаблон можно вставив в окно соответствующий элемент управления и выбрав пункт меню `Формат → Изменить стиль → Изменить копию`. В результате среда программирования добавит шаблон стиля элемента в раздел `Window.Resources` файла описания окна. После этого элемент можно из окна удалить.

Кнопка закрытия вкладки представляет стандартную кнопку с измененным оформлением, поэтому описание стиля будет основано на стиле элемента `Button`. Получим шаблон стиля этого элемента и перенесем его в файл `Generic.xaml`:

```
<Style x:Key="CustomButton" TargetType="{x:Type Button}">
    ...
</Style>
```

Листинг 1. Шаблон стиля элемента Button

Для кнопки необходимо только изменить оформление путем изменения значений свойств в шаблоне. Основное изменение касается блока `ControlTemplate`, в котором находится описание стиля отображения кнопки:

```
<ControlTemplate TargetType = "{x:Type Button}">
    <Border ...>
        <ContentPresenter .../>
    </Border>
    ...
</ControlTemplate>
```

Листинг 2. Измененный шаблон стиля элемента Button

Стандартная кнопка имеет объемное оформление рамки `ButtonChrome`. В нашем примере необходима плоская кнопка, поэтому изменяем рамку просто на `Border`. Другие изменения касаются только цветового оформления (рис. 3).

Рис. 3. Состояния кнопки закрытия вкладки

Список выбора вкладки основан на шаблоне элемента `ComboBox`. Стандартный комбинированный список состоит из нескольких частей: кнопка раскрытия списка `ToggleButton`, текстовое поле ввода `TextBox`, а также раскрывающийся список. В рассматриваемом примере текстовое поле ввода не нужно, поэтому его описание можно удалить. Полученный шаблон также переносим в файл `Generic.xaml`:

```
<Style x:Key = "CustomComboBox" TargetType = "{x:Type ComboBox}">
    ...
</Style>
```

Листинг 3. Шаблон стиля элемента ComboBox

Кнопку открытия списка сделаем плоской и установим выравнивание по правому краю:

```
<ControlTemplate TargetType="{x:Type ToggleButton}">
    <Border ...>
        <Grid HorizontalAlignment="Right" Width="{DynamicResource
{x:Static SystemParameters.VerticalScrollBarWidthKey}}">
            <Path x:Name="Arrow" .../>
        </Grid>
    </Border>
    ...
</ControlTemplate>
```

Листинг 4. Измененный шаблон стиля кнопки открытия списка

Теперь необходимо изменить шаблон самого элемента `ComboBox`. В раскрывающемся списке удаляем обрамление тенью (рис. 4):

```
<Popup x:Name="PART_Popup" ...>
    <Border x:Name="DropDownBorder" ...>
        ...
    </Border>
</Popup>
```

Листинг 5. Измененный шаблон стиля раскрывающегося списка

Рис. 4. Список вкладок

Далее рассмотрим оформление вкладки `CustomTabItem` и панели вкладок `CustomTabControl`. Добавим в окно стандартную панель вкладок, выделим в ней вкладку и получим шаблон:

```
<Style x:Key="CustomTabItem" TargetType="{x:Type TabItem}">
    ...
</Style>
```

Листинг 6. Шаблон стиля вкладки TabItem

Вкладка будет состоять из нескольких элементов: два полигона, представляющие скошенные углы вкладки, кнопка закрытия и подпись:

```
<ControlTemplate TargetType="{x:Type TabItem}">
    <Border ...>
        <StackPanel Orientation="Horizontal" ...>
            <Polygon x:Name="pLeft" Fill="{TemplateBinding Back-
ground}" Points="0,3 2,0 2,3" StrokeThickness="0"
Stretch="UniformToFill" Width="20"/>
            <StackPanel Orientation="Horizontal" ...>
                <ContentPresenter x:Name="Content" .../>
                <Button x:Name="PART_buttonClose" Content="X" Font-
Weight="Bold" Height="18" Width="18" Visibility="Hidden"
Style="{StaticResource CustomButton}"/>
            </StackPanel>
            <Polygon x:Name="pRight" Fill="{TemplateBinding Back-
ground}" Points="0,3 0,0 2,3" StrokeThickness="0"
Stretch="UniformToFill" Width="20"/>
        </StackPanel>
    </Border>
</ControlTemplate>
```

Листинг 7. Измененный шаблон стиля вкладки TabItem

Кнопка закрытия имеет специальное имя `PART_buttonClose`, так как она будет реализовывать функцию закрытия вкладки. Приставка `PART_` в названии указывает на то, что это обязательная составляющая элемента. В качестве ее стиля указан созданный стиль `CustomButton`.

Поскольку необходимо добавить новый функционал для вкладки, а именно ее закрытие при нажатии на кнопку, то для вкладки создадим в файле `CustomTabControl.cs` класс `CustomTabItem`, производный от `TabItem`:

```
    [TemplatePart(Name = "PART_buttonClose", Type = typeof(Button))]
    public class CustomTabItem : TabItem
    {
        static CustomTabItem()
        {        DefaultStyleKeyProper-
ty.OverrideMetadata(typeof(CustomTabItem), new FrameworkPropertyMetada-
ta(typeof(CustomTabItem)));
        }
        public override void OnApplyTemplate()
        {
            base.OnApplyTemplate();
            Button buttonClose = (But-
ton)this.Template.FindName("PART_buttonClose", this);
            buttonClose.Click += new RoutedEv-
entHandler(buttonClose_Click);
        }
        private void buttonClose_Click(object sender, RoutedEventArgs e)
        {
            ItemsControl ic = (ItemsControl)this.Parent;
            ic.Items.Remove(this);
            if (ic.Items.Count == 0)
                ic.Visibility = Visibility.Hidden;
        }
    }
```

Листинг 8. Класс вкладки CustomTabItem

Для класса указан атрибут `TemplatePart`, в котором указана обязательная составляющая шаблона – кнопка закрытия вкладки. В конструкторе устанавливается стиль оформления вкладки по умолчанию, т.е. тот, который описан в файле `Generic.xaml`. В классе перегружается метод `OnApplyTemplate()`, который вызывается после применения шаблона оформления к элементу. В этом методе получаем кнопку закрытия из шаблона и добавляем к ней обработчик события нажатия.

Обработчик события `Click` кнопки получает родителя вкладки, а это панель вкладок (или любой другой контейнер, который может включать в себя вкладки) и удаляем текущую вкладку. Если вкладок не осталось после удаления, то скрывает панель вкладок. Для того, чтобы связать созданный шаблон с классом, необходимо в стиле и шаблоне изменить значение целевого типа `TargetType` на `{x:Type local:CustomTabItem}`.

Аналогично добавим стиль для панели вкладок CustomTabControl:

```
    <Style x:Key="CustomTabControl" TargetType="{x:Type TabControl}">
        ...
    </Style>
```

Листинг 9. Шаблон стиля панели TabControl

И изменим его оформление. В данном шаблоне меняем `TabPanel` на `ToolBarPanel`, для того, чтобы вкладки не выстраивались в несколько рядов. Также добавляем в шаблон открывающийся список, к которому применяется созданный стиль `CustomComboBox`. Списку присваивается специальное имя `PART_comboBoxTabs`, поскольку он будет обязательным в шаблоне:

```
    <ControlTemplate TargetType="{x:Type TabControl}">
        <Grid ...>
            <Grid.ColumnDefinitions>
                <ColumnDefinition x:Name="ColumnDefinition0"/>
                <ColumnDefinition x:Name="ColumnDefinition1"
Width="Auto"/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition x:Name="RowDefinition0" Height="Auto"/>
                <RowDefinition x:Name="RowDefinition1" Height="*"/>
            </Grid.RowDefinitions>
            <ToolBarPanel Orientation="Horizontal" x:Name="HeaderPanel"
Grid.Column="0" Grid.Row="0" .../>
            <ComboBox Style="{StaticResource CustomComboBox}"
Grid.Column="1" Grid.Row="0" x:Name="PART_comboBoxTabs" Margin="5,0"
Height = "18" Width="18"/>
            <Border x:Name="ContentPanel" Grid.Column="0"
Grid.ColumnSpan="2" Grid.Row="1" ...>
                <ScrollViewer VerticalScrollBarVisibility = "Auto" Hor-
izontalScrollBarVisibility="Auto">
                    <ContentPresenter x:Name =
"PART_SelectedContentHost" ContentSource="SelectedContent" .../>
                </ScrollViewer>
            </Border>
        </Grid>
        ...
    </ControlTemplate>
```

Листинг 10. Измененный шаблон стиля панели TabControl

Для реализации нового функционала панели вкладок создадим в файле `CustomTabControl.cs` класс `CustomTabControl`, производный от `TabControl`:

```
    [TemplatePart(Name = "PART_comboBoxTabs", Type = typeof(ComboBox))]
    public class CustomTabControl : TabControl
    {
        private ComboBox comboBoxTabs;
        static CustomTabControl()
        {
            DefaultStyleKeyProperty.OverrideMetadata(
typeof(CustomTabControl), new FrameworkPropertyMetadata(
typeof(CustomTabControl)));
        }
        public override void OnApplyTemplate()
        {
            base.OnApplyTemplate();
            comboBoxTabs = (Com-
boBox)this.Template.FindName("PART_comboBoxTabs", this);
            comboBoxTabs.SelectionChanged += new SelectionChangedEv-
entHandler(comboBoxTabs_SelectionChanged);
            foreach(var item in this.Items)
                comboBoxTabs.Items.Add((item as TabItem).Header);
        }
        void comboBoxTabs_SelectionChanged(object sender, Selection-
ChangedEventArgs e)
        {
```

```
            if(((ComboBox)sender).SelectedIndex != -1)
                if(this.SelectedIndex != ((Com-
boBox)sender).SelectedIndex)
                    this.SelectedIndex = ((Com-
boBox)sender).SelectedIndex;
            ((ComboBox)sender).SelectedIndex = -1;
        }
        protected override void OnItemsChanged( Sys-
tem.Collections.Specialized.NotifyCollectionChangedEventArgs e)
        {
            base.OnItemsChanged(e);
            if (e.NewItems != null)
                comboBoxTabs.Items.Add((e.NewItems[0] as Ta-
bItem).Header);
            else
                if(e.OldItems != null)
                    comboBoxTabs.Items.Remove((e.OldItems[0] as Ta-
bItem).Header);
        }
    }
```

Листинг 11. Класс вкладки CustomTabControl

В этом классе также указан атрибут `TemplatePart` для раскрывающегося списка. В перегруженном методе `OnApplyTemplate()` получаем список, добавляем к нему обработчик изменения выбранного элемента списка `SelectionChanged`, а также добавляем в список названия вкладок. В обработчике проверяем выбранный элемент и устанавливаем для панели вкладок индекс выбранной вкладки в значение индекса выбранного элемента списка.

В классе необходимо перегрузить также метод `OnItemsChanged()`, отвечающий за изменение набора вкладок. В обработчике этого метода проверяем, добавилась ли вкладка. Если вкладка добавлена, то в раскрывающийся список добавим элемент, соответствующий названию вкладки. Если же вкладка была удалена, то удаляем соответствующий элемент из списка. Шаблон панели вкладок также связываем с классом путем изменения свойства `TargetType`.

Для вывода созданного элемента управления в окно можно перенести созданный элемент на окно в дизайнере либо в разметку окна добавить:

```
    <local:CustomTabControl HorizontalAlignment="Left" Height="200"
Margin="60,60,0,0" VerticalAlignment="Top" Width="300">
        <local:CustomTabItem Header="TabItem">

        </local:CustomTabItem>
        <local:CustomTabItem Header="TabItem">

        </local:CustomTabItem>
    </local:CustomTabControl>
```

Листинг 12. Добавление CustomTabControl в окно приложения

В данном примере на окно будет добавлена панель вкладок, содержащая две вкладки (рис. 2). Приставка `local:` означает, что описания элементов управления находятся в текущем проекте. Данная связь указывается в начале файла разметки окна:

```
<Window
    ...
    xmlns:local="clr-namespace: ApplicationWithCustomTabControl"
    ...>
```

Таким образом, используя технологию WPF, разработчик имеет возможность создавать различные элементы управления с особым функционалом и оформлением.

Список литературы

1. Крис Андерсон. Основы Windows Presentation Foundation. СПб.: БХВ-Петербург, 2008.
2. Чарльз Петцольд. Microsoft Windows Presentation Foundation. Базовый курс. СПб.: Питер, 2012.
3. Введение в WPF в Visual Studio [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/dotnet/framework/wpf/getting-started/introduction-to-wpf-in-vs (дата обращения: 15.02.2018)

DEVELOP CUSTOM WPF CONTROLS

D.I. Maximov
assistant
timonpm@mail.ru
Yelets

Bunin Yelets State University

Abstract. Currently, when developing applications, much attention is paid to design, which resulted in the appearance of controls with non-standard design and functionality. To implement such controls, special programming technologies are used. The article describes the principles of creating a custom control based on Windows Presentation Foundation (WPF) technology. This technology allows you to create elements with different designs and additional functionality. As an example, the article considers the development of the tab bar, which is typical for modern applications.

Keywords: custom control element; programming; Windows Presentation Foundation; C#; XAML.

References

1. Chris Anderson. Osnovy` Windows Presentation Foundation [Essential Windows Presentation Foundation]. Spb.: BVH-Peterburg, 2008.
2. Charles Petzold. Applications = Code + Markup: A Guide to the Microsoft Windows Presentation Foundation. Spb.: Piter, 2008.
3. Vvedenie v WPF v Visual Studio [Introduction to WPF in Visual Studio] [Electronic resource]. URL: https://docs.microsoft.com/ru-ru/dotnet/framework/wpf/getting-started/introduction-to-wpf-in-vs (Accessed: 15.02.2018)