Ресурсы

Система ресурсов WPF представляет собой простой способ поддержания вместе набора полезных объектов, таких как наиболее часто используемые кисти, стили или шаблоны, что существенно упрощает работу с ними.

- Эффективность. Ресурсы позволяют определять объект один раз и затем использовать его в нескольких местах внутри разметки. Это упрощает код и делает его намного эффективнее.
- Сопровождаемость. Ресурсы позволяют переносить низкоуровневые детали форматирования (вроде размеров шрифтов) в центральное место, где их легко изменять. Это своего рода XAML-эквивалент создания констант в коде.
- Адаптируемость. После отделения определенной информации от остальной части приложения и ее помещения в раздел ресурсов появляется возможность ее динамической модификации. Например, может понадобиться изменять детали ресурсов на основе пользовательских предпочтений или текущего языка.

Пример

```
<Window.Resources>
    <Button x:Key="test" Content="Test Button" Background="Black"
TextBlock.Foreground="White"/>
  </Window.Resources>
  <Grid>
    <Button>
      <StaticResource ResourceKey="test"/>
    </Button>
  </Grid>
```

Ресурсы приложения

```
<Application.Resources>
    <Button x:Key="_test" Content="Test Button" Background="Red" TextBlock.Foreground="White"/>
</Application.Resources>
```

Несложно догадаться, что ресурсы приложения предоставляют прекрасную возможность для многократного использования объекта по всему приложению.

Ресурсы системы

Динамические ресурсы главным образом предназначены для того, чтобы помочь приложению реагировать на изменения в системных настройках.

- Класс ***SystemColors*** предоставляет доступ к настройкам цвета.
- Класс ***SystemFonts*** обеспечивает доступ к настройкам шрифтов.
- Класс ***SystemParameters*** охватывает огромный список настроек, которые описывают стандартный размер различных экранных элементов, параметры клавиатуры и мыши, размер экрана, а также активные графические эффекты

Пример

```
Button ptr = new Button();

ptr.Content = SystemFonts.CaptionFontFamily;

MessageBox.Show(ptr.Content.ToString());
```

Создание словаря ресурсов

Прикрепление к глобальному файлу

```
<Application x:Class="WpfApplication1.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             StartupUri="MainWindow.xaml">
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="Dictionary1.xaml"></ResourceDictionary>
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

Создание словаря ресурсов

MergedDictionaries — это коллекция объектов ResourceDictionary, которые будут использоваться для пополнения коллекции ресурсов.

Чтобы добавить собственные ресурсы и включить их в словари ресурсов, необходимо просто разместить их перед или после раздела MergedProperties.

Стили

Стилем называется коллекция значений свойств, которые могут применяться к элементу. Система стилей WPF играет ту же роль, которую играет стандарт каскадных таблиц стилей (Cascading Style Sheet — CSS) в HTML-разметке. Подобно CSS, стили WPF позволяют определять общий набор характеристик форматирования и применять его повсюду в приложении для обеспечения согласованного вида. Как и CSS, они могут работать автоматически, предназначаться для элементов конкретного типа и каскадироваться через дерево элементов.

Пример реализиции

```
<Window.Resources>
        <Style x:Key="MyButtonStyle">
            <Setter Property="Control.FontFamily" Value="Calibri"></Setter>
            <Setter Property="Control.FontSize" Value="18"></Setter>
            <Setter Property="Control.FontWeight" Value="Bold"></Setter>
            <Setter Property="Control.Padding" Value="5"></Setter>
            <Setter Property="Control.Margin" Value="5"></Setter>
        </Style>
</Window.Resources>
```

```
cmd.Style = (Style)cmd.FindResource("MyButtonStyle");
```

Setters

Коллекция объектов Setter или EventSetter, которые устанавливают значения для свойств и присоединяют обработчики событий автоматически

Triggers

Коллекция объектов, унаследованных от класса TriggerBase, которые позволяют автоматически изменять настройки стиля. Настройки стиля могут модифицироваться, например, при изменении значения какого-то другого свойства или при поступлении какого-нибудь события

BasedOn

Свойство, которое позволяет создавать более специализированный стиль, наследующий (и дополнительно переопределяющий) параметры другого стиля

TargetType

Свойство, которое идентифицирует тип элемента, к которому применяется данный стиль. Это свойство позволяет создавать объекты Setter, влияющие только на определенные элементы, а также объекты Setter, автоматически вступающие в силу для всех элементов подходящего типа

Множество уровней стилей

Предположим, что группе элементов управления требуется назначить один и тот же шрифт без применения к каждому из них одного и того же стиля. В этом случае можно разместить нужные элементы управления в одной панели (или в контейнере другого типа) и установить стиль контейнера.

В других ситуациях требуется создать стиль, основанный на другом стиле. Для использования наследования стилей необходимо установить атрибут BasedOn соответствующего стиля.

Пример

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:local="clr-namespace:Test_WPF">
    <Style x:Key="Test_Style" TargetType="Button">
        <Setter Property="Control.Background" Value="Black"/>
        <Setter Property="Control.FontSize" Value="36" />
        <Setter Property="TextBlock.Foreground" Value="White" />
    </Style>
    <Style x:Key="New_Style" BasedOn="{StaticResource Test_Style}" TargetType="Button">
        <Setter Property="Control.FontSize" Value="16" />
    </Style>
</ResourceDictionary>
```

Автоматическое применение стилей

```
<Style x:Key="{x:Type Button}" TargetType="Button">
    <Setter Property="Control.Background" Value="Red"/>
</Style>
```

Несмотря на удобство, автоматически применяемые стили усложняют решение. Ниже перечислено несколько возможных причин:

В сложном окне с множеством стилей и уровней стилей становится трудно отслеживать то, устанавливается данное свойство посредством наследования значений свойств или с помощью стиля (и какого именно). В результате изменение даже простой детали может потребовать просмотра разметки всего окна.

В случае создания автоматического стиля, например, для элемента TextBlock, обязательно потребуется модифицировать другие элементы управления, которые используют TextBlock

Триггеры

Триггеры являются еще одним примером такой направленности WPF. С помощью триггеров можно автоматизировать процесс внесения простых изменений в стили, каковой обычно требует написания рутинной логики обработки событий. Например, можно обеспечить реакцию на изменение значения свойства и соответствующим образом автоматически подстроить стиль.

MultiTrigger

Похож на Trigger, но поддерживает проверку множества условий. Этот триггер вступает в действие, только если удовлетворены все заданные условия

DataTrigger

Этот триггер работает с привязкой данных. Он похож на Trigger, но следит за изменением в любых связанных данных

MultiDataTrigger

Этот триггер объединяет множество триггеров данных

EventTrigger

Это наиболее сложный триггер. Он применяет анимацию, когда возникает соответствующее событие

Простой триггер

```
<Style x:Key="{x:Type Button}" TargetType="Button">
    <Setter Property="Control.Background" Value="Red"/>
    <Style.Triggers>
        <Trigger Property="Control.IsFocused" Value="True">
            <Setter Property="TextBlock.Text" Value="New Trigger" />
            <Setter Property="Control.Background" Value="White"/>
        </Trigger>
    </Style.Triggers>
</Style>
```

MultiTrigger

Чтобы создать триггер, срабатывающий только при соблюдении сразу нескольких условий, можно воспользоваться классом MutliTrigger. Этот класс имеет коллекцию Conditions, которая позволяет определять цепочки комбинаций свойств и значений.

Пример

```
<Style x:Key="{x:Type Button}" TargetType="Button">
    <Setter Property="Control.Background" Value="Red"/>
    <Style.Triggers>
        <MultiTrigger>
            <MultiTrigger.Conditions>
                <Condition Property="Control.IsFocused" Value="True" />
                <Condition Property="IsMouseOver" Value="True" />
            </MultiTrigger.Conditions>
            <MultiTrigger.Setters>
                <Setter Property="TextBlock.Foreground" Value="Chocolate" />
            </MultiTrigger.Setters>
        </MultiTrigger>
    </Style.Triggers>
</Style>
```

Data Trigger

```
<Style x:Key="rd_style" TargetType="TextBlock">
    <Setter Property="Text" Value="No" />
        <Setter Property="Foreground" Value="Red" />
        <Style.Triggers>
            <DataTrigger Binding="{Binding ElementName=rd_but, Path=IsChecked}" Value="True">
                <Setter Property="Text" Value="Yes!" />
                <Setter Property="Foreground" Value="Green" />
            </DataTrigger>
        </Style.Triggers>
    </Style>
```

```
<CheckBox Grid.Row="1" Name="rd_but" Content="Тыкни"/>
<TextBlock Grid.Row="1" Grid.Column="0"  FontSize="30" Style="{DynamicResource rd_style}" Margin="0,50,0,0"/>
```

Триггер события

Триггер события (EventTrigger) ожидает возникновения конкретного события. Может показаться, что на этом этапе применяются средства установки для изменения элемента, однако это не так. Вместо этого триггер событий требует предоставления последовательности действий, модифицирующих элемент управления. Эти действия используются для применения анимации.

Пример

```
<Style.Triggers>
    <EventTrigger RoutedEvent="Mouse.MouseEnter">
        <EventTrigger.Actions>
            <BeginStoryboard>
                <Storyboard>
                    <DoubleAnimation Duration="0:0:0.2"
                                     Storyboard.TargetProperty="FontSize"
                                     To="22">
                    </DoubleAnimation>
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger.Actions>
    </EventTrigger>
```

Пример

```
<EventTrigger RoutedEvent="Mouse.MouseLeave">
    <EventTrigger.Actions>
        <BeginStoryboard>
            <Storyboard>
                <DoubleAnimation Duration="0:0:1"
                                 Storyboard.TargetProperty="FontSize">
                </DoubleAnimation>
            </Storyboard>
        </BeginStoryboard>
    </EventTrigger.Actions>
</EventTrigger>
```