

控制

项目 • 2023/02/06

WPF SDK 继续使用术语“控件”泛指任何代表应用程序中可见对象的类，请务必注意，类不必从 [Control](#) 类继承即可具有可见外观。从 [Control](#) 类继承的类包含一个 [ControlTemplate](#)，它允许控件的使用方在无需创建新子类的情况下从根本上改变控件的外观。本主题讨论在 WPF 中使用控件（包括从 [Control](#) 类继承的控件以及不从该类继承的控件）的常见方式。

创建控件的实例

可以通过使用 XAML 向应用程序添加控件。所有控件均可使用相似方式创建。

XAML

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="30"/>
        <RowDefinition Height="30"/>
        <RowDefinition Height="30"/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>

    <Label>
        Enter your first name:
    </Label>
    <TextBox Grid.Row="0" Grid.Column="1"
            Name="firstName" Margin="0,5,10,5"/>

    <Label Grid.Row="1" >
        Enter your last name:
    </Label>
    <TextBox Grid.Row="1" Grid.Column="1"
            Name="lastName" Margin="0,5,10,5"/>

    <Button Grid.Row="2" Grid.Column="0"
            Name="submit" Margin="2">
        View message
    </Button>

    <Button Grid.Row="2" Grid.Column="1"
            Name="Clear" Margin="2">
        Clear Name
    </Button>
```

```
</Button>
</Grid>
```

以下示例在代码中创建相同的应用程序。为简洁起见，示例中省略了 Grid、grid1 的创建过程。grid1 的列和行定义与前面的 XAML 示例中所示的相同。

C#

```
Label firstNameLabel;
Label lastNameLabel;
TextBox firstName;
TextBox lastName;
Button submit;
Button clear;

void CreateControls()
{
    firstNameLabel = new Label();
    firstNameLabel.Content = "Enter your first name:";
    grid1.Children.Add(firstNameLabel);

    firstName = new TextBox();
    firstName.Margin = new Thickness(0, 5, 10, 5);
    Grid.SetColumn(firstName, 1);
    grid1.Children.Add(firstName);

    lastNameLabel = new Label();
    lastNameLabel.Content = "Enter your last name:";
    Grid.SetRow(lastNameLabel, 1);
    grid1.Children.Add(lastNameLabel);

    lastName = new TextBox();
    lastName.Margin = new Thickness(0, 5, 10, 5);
    Grid.SetColumn(lastName, 1);
    Grid.SetRow(lastName, 1);
    grid1.Children.Add(lastName);

    submit = new Button();
    submit.Content = "View message";
    Grid.SetRow(submit, 2);
    grid1.Children.Add(submit);

    clear = new Button();
    clear.Content = "Clear Name";
    Grid.SetRow(clear, 2);
    Grid.SetColumn(clear, 1);
    grid1.Children.Add(clear);
}
```

更改控件的外观

更改控件的外观以适应应用程序的外观，这是很常见的操作。可以根据要达到的效果，通过执行以下操作之一来更改控件的外观：

- 更改控件的属性值。
- 为控件创建 [Style](#)。
- 为控件新建 [ControlTemplate](#)。

更改控件的属性值

许多控件具有支持更改控件外观的属性，例如 [Button](#) 的 [Background](#)。可以在 XAML 和代码中设置值属性。下面的示例在 XAML 中的 [Button](#) 上设置 [Background](#)、[FontSize](#) 和 [FontWeight](#) 属性。

XAML

```
<Button FontSize="14" FontWeight="Bold">
    <!--Set the Background property of the Button to
        a LinearGradientBrush.-->
    <Button.Background>
        <LinearGradientBrush StartPoint="0,0.5"
            EndPoint="1,0.5">
            <GradientStop Color="Green" Offset="0.0" />
            <GradientStop Color="White" Offset="0.9" />
        </LinearGradientBrush>

    </Button.Background>
    View message
</Button>
```

下面的示例在代码中设置相同的属性。

C#

```
LinearGradientBrush buttonBrush = new LinearGradientBrush();
buttonBrush.StartPoint = new Point(0, 0.5);
buttonBrush.EndPoint = new Point(1, 0.5);
buttonBrush.GradientStops.Add(new GradientStop(Colors.Green, 0));
buttonBrush.GradientStops.Add(new GradientStop(Colors.White, 0.9));

submit.Background = buttonBrush;
submit.FontSize = 14;
submit.FontWeight = FontWeights.Bold;
```

为控件创建样式

利用 WPF，通过创建 [Style](#)，可以同时为许多控件指定相同的外观，而不是为应用程序中的每个实例设置属性。下面的示例创建一个 [Style](#)，它将应用于应用程序中的每个 [Button](#)。[Style](#) 定义通常在 [ResourceDictionary](#)（例如 [FrameworkElement](#) 的 [Resources](#) 属性）中以 XAML 形式定义。

XAML

```
<Style TargetType="Button">
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="FontWeight" Value="Bold"/>
    <Setter Property="Background">
        <Setter.Value>
            <LinearGradientBrush StartPoint="0,0.5"
                EndPoint="1,0.5">
                <GradientStop Color="Green" Offset="0.0" />
                <GradientStop Color="White" Offset="0.9" />
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
</Style>
```

通过将键分配给样式并在控件的 [style](#) 属性中指定该键，还可将样式仅应用于某些特定类型的控件。有关样式的详细信息，请参阅[样式设置和模板化](#)。

创建 ControlTemplate

利用 [Style](#)，可以一次为多个控件设置属性，但有时除了通过创建 [Style](#) 可执行的操作之外，可能还希望自定义 [Control](#) 的外观。从 [Control](#) 类继承的类具有 [ControlTemplate](#)，它用于定义 [Control](#) 的结构和外观。[Control](#) 的 [Template](#) 属性是公共属性，因此可以为 [Control](#) 指定非默认的 [ControlTemplate](#)。通常可以为 [Control](#) 指定新的 [ControlTemplate](#)（而不是从控件继承）以自定义 [Control](#) 的外观。

请考虑一个很常用的控件 [Button](#)。[Button](#) 的主要行为是当用户单击它时让应用程序采取某些操作。默认情况下，WPF 中的 [Button](#) 显示为一个凸出的矩形。开发应用程序时，用户可能希望利用 [Button](#) 的行为（即通过处理按钮的单击事件），不过，除了通过更改按钮的属性可以执行的操作外，也可以更改按钮的外观。在这种情况下，可以创建新的 [ControlTemplate](#)。

下面的示例为 [Button](#) 创建了一个 [ControlTemplate](#)。[ControlTemplate](#) 创建一个带圆角和渐变背景的 [Button](#)。[ControlTemplate](#) 包含一个 [Border](#)，其 [Background](#) 是带有两个 [GradientStop](#) 对象的 [LinearGradientBrush](#)。第一个 [GradientStop](#) 使用数据绑定将 [GradientStop](#) 的 [Color](#) 属性绑定到按钮背景的颜色。当设置 [Button](#) 的 [Background](#) 属性时，该值的颜色将用作第一个 [GradientStop](#)。有关数据绑定的详细信息，请参阅[数据绑定概述](#)。此示例还创建一个 [Trigger](#)，用于在 [IsPressed](#) 为 [true](#) 时更改 [Button](#) 的外观。

XAML

```
<!--Define a template that creates a gradient-colored button.-->
<Style TargetType="Button">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="Button">
                <Border
                    x:Name="Border"
                    CornerRadius="20"
                    BorderThickness="1"
                    BorderBrush="Black">
                    <Border.Background>
                        <LinearGradientBrush StartPoint="0,0.5"
                            EndPoint="1,0.5">
                            <GradientStop Color="{Binding Background.Color,
                                RelativeSource={RelativeSource TemplatedParent}}"
                                Offset="0.0" />
                            <GradientStop Color="White" Offset="0.9" />
                        </LinearGradientBrush>
                    </Border.Background>
                    <ContentPresenter
                        Margin="2"
                        HorizontalAlignment="Center"
                        VerticalAlignment="Center"
                        RecognizesAccessKey="True"/>
                </Border>
                <ControlTemplate.Triggers>
                    <!--Change the appearance of
                        the button when the user clicks it.-->
                    <Trigger Property="IsPressed" Value="true">
                        <Setter TargetName="Border" Property="Background">
                            <Setter.Value>
                                <LinearGradientBrush StartPoint="0,0.5"
                                    EndPoint="1,0.5">
                                    <GradientStop Color="{Binding Background.Color,
                                        RelativeSource={RelativeSource TemplatedParent}}"
                                        Offset="0.0" />
                                    <GradientStop Color="DarkSlateGray" Offset="0.9" />
                                </LinearGradientBrush>
                            </Setter.Value>
                        </Setter>
                    </Trigger>

                    </ControlTemplate.Triggers>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
```

XAML

```
<Button Grid.Row="2" Grid.ColumnSpan="2" Name="submitName"
        Background="Green">View message</Button>
```

① 备注

为使此示例正常工作，Button 的 Background 属性必须设置为 SolidColorBrush。

订阅事件

可以使用 XAML 或代码来订阅控件的事件，但只能在代码中处理事件。下面的示例演示如何订阅 Button 的 Click 事件。

XAML

```
<Button Grid.Row="2" Grid.ColumnSpan="2" Name="submitName"
        Click="submit_Click"
        Background="Green">View message</Button>
```

C#

```
submit.Click += new RoutedEventHandler(submit_Click);
```

下面的示例处理 Button 的 Click 事件。

C#

```
void submit_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Hello, " + firstName.Text + " " + lastName.Text);
}
```

控件中的丰富内容

从 Control 类继承的大多数类具有包含丰富内容的能力。例如，Label 可以包含任意对象，例如字符串、Image 或 Panel。下列类支持丰富内容，可以用作 WPF 中大多数控件的基类。

- ContentControl-- 从此类继承的类的部分示例包括 Label、Button 和 ToolTip。
- ItemsControl-- 从此类继承的类的部分示例包括 ListBox、Menu 和 StatusBar。

- [HeaderedContentControl](#)-- 从此类继承的类的部分示例包括 [TabItem](#)、[GroupBox](#) 和 [Expander](#)。
- [HeaderedItemsControl](#)-- 从此类继承的类的部分示例包括 [MenuItem](#)、[TreeViewItem](#) 和 [ToolBar](#)。

有关这些基类的详细信息，请参阅 [WPF 内容模型](#)。

请参阅

- [样式设置和模板化](#)
- [按类别分类的控件](#)
- [控件库](#)
- [数据模板化概述](#)
- [数据绑定概述](#)
- [输入](#)
- [启用命令](#)
- [演练：创建自定义的动画按钮](#)
- [控件自定义](#)

按类别分类的控件

项目 · 2023/02/06

Windows Presentation Foundation (WPF) 控件可以按逻辑分组为多个类别。 可以使用这些类别来为方案选择适当的控件，从而有助于查看具有相同用法模式或功能的控件。

Layout

布局控件用于管理子元素的大小、尺寸、位置和排列。

- [Border](#)
- [BulletDecorator](#)
- [Canvas](#)
- [DockPanel](#)
- [Expander](#)
- [Grid](#)
- [GridSplitter](#)
- [GroupBox](#)
- [Panel](#)
- [ResizeGrip](#)
- [Separator](#)
- [ScrollBar](#)
- [ScrollViewer](#)
- [StackPanel](#)
- [Thumb](#)
- [Viewbox](#)
- [VirtualizingStackPanel](#)
- [Window](#)

- [WrapPanel](#)

按钮

按钮是最基本的用户界面控件之一。当用户单击按钮时，应用程序通常执行 [Click](#) 事件中的某些任务。

- [Button](#)
- [RepeatButton](#)

数据显示

数据显示控件用于显示来自数据源的信息。

- [DataGrid](#)
- [ListView](#)
- [TreeView](#)

日期显示和选项

日期控件用于显示和选择日历信息。

- [Calendar](#)
- [DatePicker](#)

菜单

菜单用于对关联的操作进行分组或提供上下文帮助。

- [ContextMenu](#)
- [Menu](#)
- [ToolBar](#)

选择

选择控件用于使用户选择一个或多个选项。

- [CheckBox](#)
- [ComboBox](#)
- [ListBox](#)
- [RadioButton](#)
- [Slider](#)

导航

导航控件通过创建目标框架或选项卡式的应用程序外观来增强或扩展应用程序导航体验。

- [Frame](#)
- [Hyperlink](#)
- [Page](#)
- [NavigationWindow](#)
- [TabControl](#)

对话框

对话框为常见的用户交互方案（如打印）提供目标支持。

- [OpenFileDialog](#)
- [PrintDialog](#)
- [SaveFileDialog](#)

用户信息

用户信息控件提供上下文反馈或阐明应用程序的用户界面。 用户通常无法与这些控件进行交互。

- [AccessText](#)
- [Label](#)
- [Popup](#)

- [ProgressBar](#)
- [StatusBar](#)
- [TextBlock](#)
- [ToolTip](#)

文档

WPF 包含几个用于查看文档的专用控件。 这些控件基于目标用户方案优化阅读体验。

- [DocumentViewer](#)
- [FlowDocumentPageViewer](#)
- [FlowDocumentReader](#)
- [FlowDocumentScrollView](#)
- [StickyNoteControl](#)

输入

输入控件可以使用户输入文本和其他内容。

- [TextBox](#)
- [RichTextBox](#)
- [PasswordBox](#)

媒体

WPF 不仅包括大多数常见图像格式的 [codecs]，而且还包括对承载音频和视频内容的集成支持。

- [Image](#)
- [MediaElement](#)
- [SoundPlayerAction](#)

数字墨迹

数字墨迹控件提供对墨迹查看和墨迹输入等 Tablet PC 功能的集成支持。

- [InkCanvas](#)
- [InkPresenter](#)

另请参阅

- [控件库](#)

WPF 内容模型

项目 • 2023/02/06

Windows Presentation Foundation (WPF) 是一个演示平台，提供了许多控件和类似控件的类型，主要用于显示不同类型的内容。 若要确定所要使用的控件或要从其派生的控件，应该了解特定控件可以最佳效果显示的对象类型。

本主题概述了适用于 WPF 控件和类似控件的类型的内容模型。 内容模型描述可在控件中使用的内容。 本主题还列出了每个内容模型的内容属性。 内容属性是一种用于存储对象内容的属性。

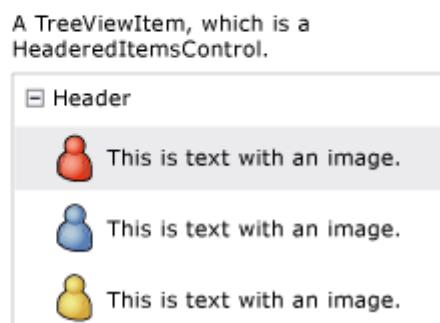
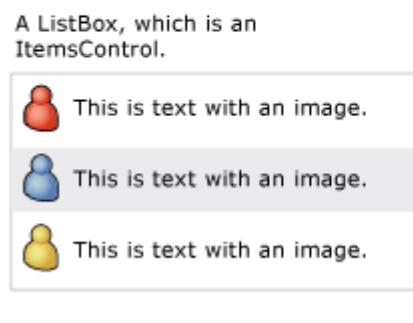
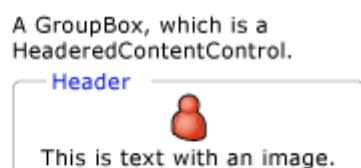
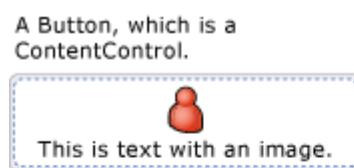
包含任意内容的类

某些控件可以包含任何类型的对象（如字符串、[DateTime](#) 对象或作为其他项的容器的 [UIElement](#)）。 例如，[Button](#) 可以包含图像和一些文本；或者 [CheckBox](#) 可以包含 [DateTime.Now](#) 值。

WPF 有四个可包含任意内容的类。 下表列出了继承自 [Control](#) 的类。

包含任意内容的类	内容
ContentControl	一个任意对象。
HeaderedContentControl	一个标头和一个项（两者都是任意对象）。
ItemsControl	一个任意对象集合。
HeaderedItemsControl	一个标头和一个项集合（全部都是任意对象）。

继承自这些类的控件可以包含相同类型的内容，并可以采用相同方式处理该内容。 下图显示了来自每个内容模型的一个控件，该控件包含了图像和一些文本：

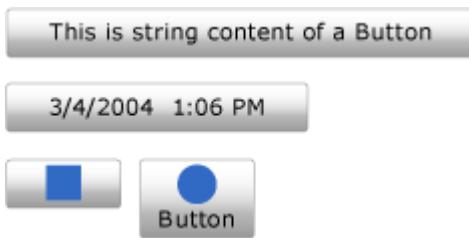


包含一个任意对象的控件

[ContentControl](#) 类一段任意内容。它的内容属性为 [Content](#)。以下控件继承自 [ContentControl](#) 并使用其内容模型：

- [Button](#)
- [ButtonBase](#)
- [CheckBox](#)
- [ComboBoxItem](#)
- [ContentControl](#)
- [Frame](#)
- [GridViewColumnHeader](#)
- [GroupItem](#)
- [Label](#)
- [ListBoxItem](#)
- [ListViewItem](#)
- [NavigationWindow](#)
- [RadioButton](#)
- [RepeatButton](#)
- [ScrollViewer](#)
- [StatusBarItem](#)
- [ToggleButton](#)
- [ToolTip](#)
- [UserControl](#)
- [Window](#)

下图显示了四个按钮，它们的 [Content](#) 设置为字符串、[DateTime](#) 对象、[Rectangle](#) 和包含 [Ellipse](#) 和 [TextBlock](#) 的 [Panel](#)：



有关如何设置 [Content](#) 属性的示例，请参阅 [ContentControl](#)。

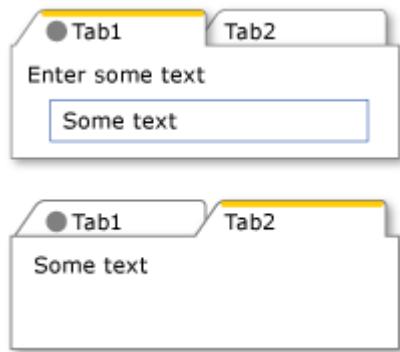
包含一个标头和一个任意对象的控件

[HeaderedContentControl](#) 类继承自 [ContentControl](#) 并显示包含标题的内容。它继承内容属性 [Content](#)，后者继承自 [ContentControl](#) 并定义具有 [Object](#) 类型的 [Header](#)；因此两者都可以是任意对象。

以下控件继承自 [HeaderedContentControl](#) 并使用其内容模型：

- [Expander](#)
- [GroupBox](#)
- [TabItem](#)

下图显示了两个 [TabItem](#) 对象。第一个 [TabItem](#) 具有作为 [Header](#) 和 [Content](#) 的 [UIElement](#) 对象。[Header](#) 设置为包含 [Ellipse](#) 和 [TextBlock](#) 的 [StackPanel](#)。[Content](#) 设置为包含 [TextBlock](#) 和 [Label](#) 的 [StackPanel](#)。第二个 [TabItem](#) 在 [Header](#) 中包含字符串，并在 [Content](#) 中包含 [TextBlock](#)。



有关如何创建 [TabItem](#) 对象的示例，请参阅 [HeaderedContentControl](#)。

包含一个任意对象集合的控件

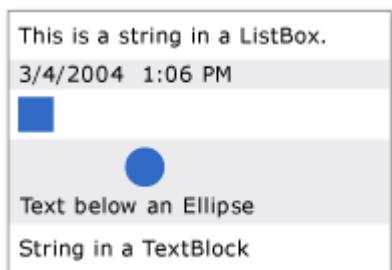
[ItemsControl](#) 类继承自 [Control](#)，可以包含多个项，例如字符串、对象或其他元素。它的内容属性为 [ItemsSource](#) 和 [Items](#)。[ItemsSource](#) 通常用于使用数据集合填充 [ItemsControl](#)。如果不使用集合填充 [ItemsControl](#)，可使用 [Items](#) 属性添加项。

以下控件继承自 [ItemsControl](#) 并使用其内容模型：

- [Menu](#)
- [MenuBase](#)
- [ContextMenu](#)
- [ComboBox](#)
- [ItemsControl](#)
- [ListBox](#)
- [ListView](#)
- [TabControl](#)
- [TreeView](#)
- [Selector](#)
- [StatusBar](#)

下图显示了包含这些类型的项的 [ListBox](#)：

- 一个字符串。
- [DateTime](#) 对象。
- 一个 [UIElement](#)。
- [Panel](#)，包含一个 [Ellipse](#) 和一个 [TextBlock](#)。



包含一个标头和一个任意对象集合的控件

[HeaderedItemsControl](#) 类继承自 [ItemsControl](#)，可以包含多个项，例如字符串、对象或其他元素，也可以包含标题。它继承 [ItemsControl](#) 内容属性 [ItemsSource](#) 和 [Items](#)，并定义可以是任意对象的 [Header](#) 属性。

以下控件继承自 [HeaderedItemsControl](#) 并使用其内容模型：

- [MenuItem](#)
- [ToolBar](#)
- [TreeViewItem](#)

包含一个 UIElement 对象集合的类

[Panel](#) 类定位和排列 [UIElement](#) 子对象。 它的内容属性为 [Children](#)。

以下类继承自 [Panel](#) 类并使用其内容模型：

- [Canvas](#)
- [DockPanel](#)
- [Grid](#)
- [TabPanel](#)
- [ToolBarOverflowPanel](#)
- [ToolBarPanel](#)
- [UniformGrid](#)
- [StackPanel](#)
- [VirtualizingPanel](#)
- [VirtualizingStackPanel](#)
- [WrapPanel](#)

有关详细信息，请参阅[面板概述](#)。

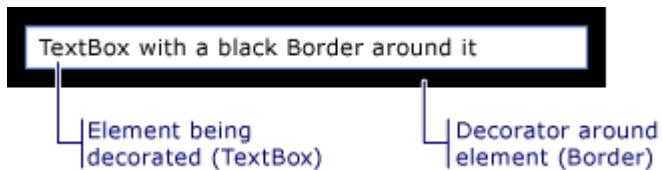
影响 UIElement 外观的类

[Decorator](#) 类将视觉效果应用到单个 [UIElement](#) 子对象上或应用到其周围。 它的内容属性为 [Child](#)。 以下类继承自 [Decorator](#) 并使用其内容模型：

- [AdornerDecorator](#)
- [Border](#)
- [BulletDecorator](#)

- [ButtonChrome](#)
- [ClassicBorderDecorator](#)
- [InkPresenter](#)
- [ListBoxChrome](#)
- [SystemDropShadowChrome](#)
- [Viewbox](#)

下图显示了一个 [TextBox](#)，它周围有一个（装饰有）Border。



具有边框的 TextBlock

提供 UIElement 相关视觉反馈的类

[Adorner](#) 类向用户提供视觉线索。例如，使用 [Adorner](#) 向元素添加功能句柄，或提供有关控件的状态信息。[Adorner](#) 类提供一个框架，以便可以创建自己的装饰器。WPF 不会提供任何实现的装饰器。有关详细信息，请参阅[装饰器概述](#)。

可让用户输入文本的类

WPF 提供了三个可让用户输入文本的主要控件。每个控件都以不同方式显示文本。下表列出了这三个与文本相关的控件、显示文本时的功能以及包含控件文本的属性。

控制	文本显示方式	内容属性
TextBox	纯文本	Text
RichTextBox	带格式文本	Document
PasswordBox	隐藏文本（字符已屏蔽）	Password

显示文本的类

某些类可用于显示纯文本或带格式文本。可使用 [TextBlock](#) 显示少量文本。如果希望显示大量文本，请使用 [FlowDocumentReader](#)、[FlowDocumentPageViewer](#) 或 [FlowDocumentScrollView](#) 控件。

`TextBlock` 具有两个内容属性：`Text` 和 `Inlines`。如果希望显示使用一致格式的文本，`Text` 属性通常是最佳选择。如果计划在文本中使用不同的格式设置，请使用 `Inlines` 属性。`Inlines` 属性是 `Inline` 对象集合，用于指定文本格式的设置方式。

下表列出了 `FlowDocumentReader`、`FlowDocumentPageViewer` 和 `FlowDocumentScrollViewer` 类的内容属性。

控制	内容属性	内容属性类型
<code>FlowDocumentPageViewer</code>	文档	<code>IDocumentPaginatorSource</code>
<code>FlowDocumentReader</code>	文档	<code>FlowDocument</code>
<code>FlowDocumentScrollViewer</code>	文档	<code>FlowDocument</code>

`FlowDocument` 实现 `IDocumentPaginatorSource` 接口；因此这三个类都可使用 `FlowDocument` 作为内容。

设置文本格式的类

`TextElement` 及其相关类可用于对文本设置格式。`TextElement` 对象包含 `TextBlock` 和 `FlowDocument` 对象中的文本并对其设置格式。`TextElement` 对象的两种主要类型为 `Block` 元素和 `Inline` 元素。`Block` 元素表示文本块（如段落或列表）。`Inline` 元素标识块中的部分文本。许多 `Inline` 类都指定它们所应用到的文本的格式。每个 `TextElement` 都有其自己的内容模型。有关详细信息，请参阅 [TextElement 内容模型概述](#)。

另请参阅

- [高级](#)

控件库

项目 • 2023/02/06

Windows Presentation Foundation (WPF) 控件库包含有关 Windows Presentation Foundation (WPF) 提供的控件的信息，这些控件按字母顺序列出。

本节内容

[Border](#)

[BulletDecorator](#)

[Button](#)

[日历](#)

[画布](#)

[CheckBox](#)

[ComboBox](#)

[ContextMenu](#)

[DataGrid](#)

[DatePicker](#)

[DockPanel](#)

[DocumentViewer](#)

[扩展器](#)

[FlowDocumentPageViewer](#)

[FlowDocumentReader](#)

[FlowDocumentScrollViewer](#)

[Frame](#)

[网格](#)

[GridSplitter](#)

[GroupBox](#)

[图像](#)

[标签](#)

[ListBox](#)

[ListView](#)

[菜单](#)

[Panel](#)

[PasswordBox](#)

[Popup](#)

[ProgressBar](#)

[PrintDialog](#)

[RadioButton](#)

[RepeatButton](#)

[RichTextBox](#)

[滚动条](#)

[ScrollViewer](#)

[分隔符](#)

[滑块](#)

[StackPanel](#)

[StatusBar](#)

[TabControl](#)

[TextBlock](#)

[TextBox](#)

[工具栏](#)

[ToolTip](#)

[TreeView](#)

[WrapPanel](#)

[Viewbox](#)

参考

[System.Windows.Controls](#)

[System.Windows.Controls.Primitives](#)

相关章节

[控件自定义](#)

[按类别分类的控件](#)

[WPF 内容模型](#)

边框

项目 • 2023/02/06

以下示例演示如何动态更改 Border 元素的属性。

本节内容

[对 BorderThickness 值进行动画处理](#)

参考

[Decorator](#)

[Border](#)

相关章节

[面板概述](#)

[Alignment、Margin 和 Padding 概述](#)

如何：对 BorderThickness 值进行动画处理

项目 • 2023/02/06

此示例显示如何使用 [ThicknessAnimation](#) 类对边框的粗细更改进行动画处理。

示例

以下示例使用 [ThicknessAnimation](#) 对边框的粗细进行动画处理。该示例使用 [Border](#) 的 [BorderThickness](#) 属性。

C#

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Shapes;
using System.Windows.Media.Animation;
using System.Windows.Media;

namespace SDKSamples
{
    public class ThicknessAnimationExample : Page
    {
        public ThicknessAnimationExample()
        {

            // Create a NameScope for this page so that
            // Storyboards can be used.
            NameScope.SetNameScope(this, new NameScope());

            // Create a Border which will be the target of the animation.
            Border myBorder = new Border();
            myBorder.Background = Brushes.Gray;
            myBorder.BorderBrush = Brushes.Black;
            myBorder.BorderThickness = new Thickness(1);
            myBorder.Margin = new Thickness(0, 60, 0, 20);
            myBorder.Padding = new Thickness(20);

            // Assign the border a name so that
            // it can be targeted by a Storyboard.
            this.RegisterName(
                "myAnimatedBorder", myBorder);

            ThicknessAnimation myThicknessAnimation = new
ThicknessAnimation();
            myThicknessAnimation.Duration = TimeSpan.FromSeconds(1.5);
            myThicknessAnimation.FillBehavior = FillBehavior.HoldEnd;
```

```

        // Set the From and To properties of the animation.
        // BorderThickness animates from left=1, right=1, top=1, and
bottom=1
        // to left=28, right=28, top=14, and bottom=14 over one and a
half seconds.
        myThicknessAnimation.From = new Thickness(1, 1, 1, 1);
        myThicknessAnimation.To = new Thickness(28, 14, 28, 14);

        // Set the animation to target the Size property
        // of the object named "myArcSegment."
        Storyboard.SetTargetName(myThicknessAnimation,
"myAnimatedBorder");
        Storyboard.SetTargetProperty(
            myThicknessAnimation, new
PropertyPath(Border.BorderThicknessProperty));

        // Create a storyboard to apply the animation.
Storyboard ellipseStoryboard = new Storyboard();
ellipseStoryboard.Children.Add(myThicknessAnimation);

        // Start the storyboard when the Path loads.
myBorder.Loaded += delegate(object sender, RoutedEventArgs e)
{
    ellipseStoryboard.Begin(this);
};

StackPanel myStackPanel = new StackPanel();
myStackPanel.HorizontalAlignment = HorizontalAlignment.Center;
myStackPanel.Children.Add(myBorder);

Content = myStackPanel;
}
}
}

```

有关完整示例，请参阅[动画示例库](#)。

另请参阅

- [ThicknessAnimation](#)
- [BorderThickness](#)
- [Border](#)
- [动画概述](#)
- [动画和计时帮助主题](#)
- [使用关键帧对边框的粗细进行动画处理](#)

BulletDecorator

项目 • 2023/02/06

[BulletDecorator](#) 具有两个内容属性：[Bullet](#) 和 [Child](#)。[Bullet](#) 属性定义要用作项目符号的 [UIElement](#)。[Child](#) 属性定义在视觉上与项目符号对齐的 [UIElement](#)。

下图显示了使用 [BulletDecorator](#) 的控件示例。

- A BulletDecorator with a CheckBox Bullet.
- A BulletDecorator with a RadioButton Bullet.
- A BulletDecorator with a TextBox Bullet.

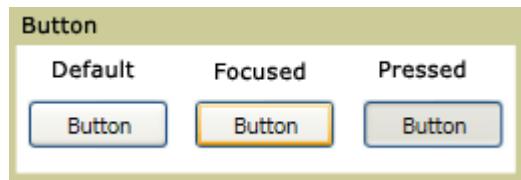
参考

[BulletDecorator](#)

Button

项目 • 2023/02/06

[Button](#) 控件对来自鼠标、键盘、触笔或其他输入设备的用户输入做出反应并引发 [Click](#) 事件。 [Button](#) 是一个基本的用户界面 (UI) 组件，可以包含简单的内容（例如文本），也可以包含复杂的内容（例如图像和 [Panel](#) 控件）。



本节内容

[创建包含图像的按钮](#)

参考

[Button](#)

[ButtonBase](#)

[RadioButton](#)

[RepeatButton](#)

如何：创建包含图像的按钮

项目 • 2023/02/06

此示例演示如何在 [Button](#) 上包含图像。

示例

以下示例创建两个 [Button](#) 控件。一个 [Button](#) 包含文本，另一个包含图像。该图像位于名为 `data` 的文件夹中，该文件夹是示例项目文件夹的子文件夹。当用户单击具有该图像的 [Button](#) 时，其他 [Button](#) 的背景和文本将会更改。

此示例使用标记创建 [Button](#) 控件，但使用代码编写 [Click](#) 事件处理程序。

XAML

```
<Button Name="btn5" Width="50" Height="30" Click="OnClick5">
    <Image Source="data\flower.jpg"></Image>
</Button>
<Button Name="btn6" BorderBrush="Black">Click the picture.</Button>
```

C#

```
void OnClick5(object sender, RoutedEventArgs e)
{
    btn6.FontSize = 16;
    btn6.Content = "This is my favorite photo.";
    btn6.Background = Brushes.Red;
}
```

另请参阅

- [控件](#)
- [控件库](#)

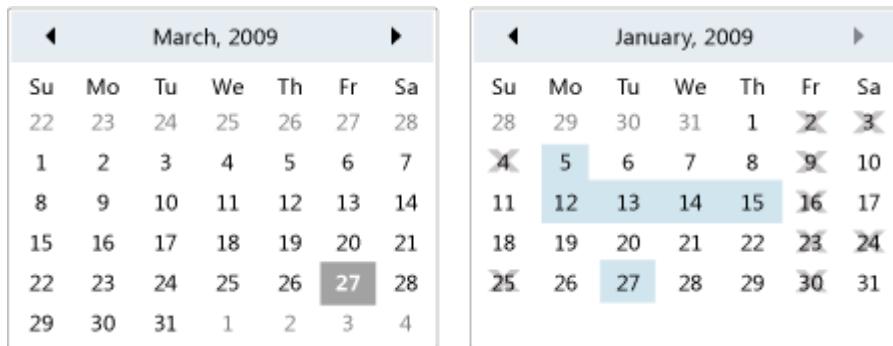
日历

项目 • 2023/02/06

借助日历，用户能够通过使用可视日历显示来选择日期。

Calendar 控件可以单独使用，也可以用作 DatePicker 控件的下拉部分。有关详细信息，请参阅 [DatePicker](#)。

下图显示了两个 Calendar 控件，一个带有选择和中断日期，一个没有。



日历控件

下表提供了有关通常与 Calendar 关联的任务的信息。

任务	实现
指定无法选择的日期。	使用 BlackoutDates 属性。
让 Calendar 显示一个月、一整年或十年。	将 DisplayMode 属性设置为月、年或十年。
指定用户是否可以选择日期、日期范围或多个日期范围。	使用 SelectionMode 。
指定 Calendar 显示的日期范围。	使用 DisplayDateStart 和 DisplayDateEnd 属性。
指定是否突出显示当前日期。	使用 IsTodayHighlighted 属性。默认情况下， IsTodayHighlighted 为 <code>true</code> 。
更改 Calendar 的大小。	使用 Viewbox 或将 LayoutTransform 属性设置为 ScaleTransform 。请注意，如果设置 Calendar 的 Width 和 Height 属性，实际日历不会更改其大小。

Calendar 控件提供了使用鼠标或键盘的基本导航。下表概述了键盘导航。

键组合	DisplayMode	操作
-----	-------------	----

键组合	DisplayMode	操作
箭头	Month	如果 SelectionMode 属性未设置为 <code>None</code> ，则更改 SelectedDate 属性。
箭头	Year	更改 DisplayDate 属性的月份。请注意， SelectedDate 不会更改。
箭头	Decade	更改 DisplayDate 的年份。请注意， SelectedDate 不会更改。
Shift+箭头	Month	如果 SelectionMode 未设置为 <code>SingleDate</code> 或 <code>None</code> ，则扩展所选日期的范围。
Home	Month	将 SelectedDate 更改为当月的第一天。
Home	Year	将 DisplayDate 的月份更改为一年中的第一个月。 SelectedDate 不会更改。
Home	Decade	将 DisplayDate 的年份更改为十年中的第一年。 SelectedDate 不会更改。
End	Month	将 SelectedDate 更改为当月的最后一天。
End	Year	将 DisplayDate 的月份更改为一年中的最后一个月。 SelectedDate 不会更改。
End	Decade	将 DisplayDate 的年份更改为十年中的最后一年。 SelectedDate 不会更改。
CTRL + 向上箭头	任意	切换到下一个更大的 DisplayMode 。如果 DisplayMode 已为 <code>Decade</code> ，则不执行任何操作。
CTRL + 向下箭头	任意	切换到下一个更小的 DisplayMode 。如果 DisplayMode 已为 <code>Month</code> ，则不执行任何操作。
空格键或 Enter	Year 或 Decade	将 DisplayMode 切换到焦点项所表示的 <code>Month</code> 或 <code>Year</code> 。

另请参阅

- [控件](#)
- [样式设置和模板化](#)

画布

项目 • 2023/02/06

[Canvas](#) 是一个布局控件，用于启用子元素的绝对定位。

本节内容

[操作指南主题](#)

参考

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

相关章节

[布局](#)

[演练：我的第一个 WPF 桌面应用程序](#)

[ScrollViewer 概述](#)

画布帮助主题

项目 · 2023/02/06

本节中的主题介绍如何使用 `Canvas` 元素来绝对定位子元素。

本节内容

[使用边框围住画布内容](#)

[获取或设置画布定位属性](#)

[创建和使用画布](#)

[使用画布的附加属性来定位子元素](#)

[使用 Thumb 重设画布大小](#)

参考

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

相关章节

[布局](#)

[演练：我的第一个 WPF 桌面应用程序](#)

[ScrollViewer 概述](#)

如何：在画布内容的周围绘制边框

项目 • 2023/02/06

本示例演示如何使用 [Border](#) 包装 [Canvas](#) 元素。

示例

以下示例演示如何在 [Canvas](#) 元素内显示 `Hello World!`。该 [Canvas](#) 元素由元素 [Border](#) 包装，以便边框勾勒出元素的轮廓。

XAML

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
    <Border
        BorderThickness="2"
        BorderBrush="Black"
        Background="LightGray"
        HorizontalAlignment="Left"
        VerticalAlignment="Top"
        Width="100"
        Height="100">
        <Canvas>
            <TextBlock Canvas.Top="10" Canvas.Left="20">Hello World!</TextBlock>
        </Canvas>
    </Border>
</Page>
```

另请参阅

- [Canvas](#)
- [Border](#)
- [面板概述](#)

如何：获取或设置画布定位属性

项目 • 2023/02/06

此示例演示如何使用 [Canvas](#) 元素的定位方法定位子内容。此示例使用 [ListBoxItem](#) 中的内容来表示定位值，并将值转换为 [Double](#) 的实例，这是定位的必需参数。然后使用 [GetLeft](#) 方法将这些值转换回字符串并在 [TextBlock](#) 元素中显示为文本。

示例

以下示例创建一个具有 11 个可选 [ListBoxItem](#) 元素的 [ListBox](#) 元素。[SelectionChanged](#) 事件触发后续代码块定义的 `ChangeLeft` 自定义方法。

每个 [ListBoxItem](#) 表示一个 [Double](#) 值，该值是 [Canvas](#) 的 [SetLeft](#) 方法接受的参数之一。为了使用 [ListBoxItem](#) 来表示 [Double](#) 的实例，必须首先将 [ListBoxItem](#) 转换为正确的数据类型。

XAML

```
<ListBox Grid.Column="1" Grid.Row="1" VerticalAlignment="Top" Width="60"
Margin="10,0,0,0" SelectionChanged="ChangeLeft">
    <ListBoxItem>Auto</ListBoxItem>
    <ListBoxItem>10</ListBoxItem>
    <ListBoxItem>20</ListBoxItem>
    <ListBoxItem>30</ListBoxItem>
    <ListBoxItem>40</ListBoxItem>
    <ListBoxItem>50</ListBoxItem>
    <ListBoxItem>60</ListBoxItem>
    <ListBoxItem>70</ListBoxItem>
    <ListBoxItem>80</ListBoxItem>
    <ListBoxItem>90</ListBoxItem>
    <ListBoxItem>100</ListBoxItem>
</ListBox>
```

当用户更改 [ListBox](#) 所选内容时，它会调用 `ChangeLeft` 自定义方法。此方法将 [ListBoxItem](#) 传递给 [LengthConverter](#) 对象，该对象将 [ListBoxItem](#) 的 [Content](#) 转换为 [Double](#) 的实例（请注意，此值已通过使用 [ToString](#) 方法转换为 [String](#)）。然后将此值传递回 [Canvas](#) 的 [SetLeft](#) 和 [GetLeft](#) 方法，以更改 `text1` 对象的位置。

C#

```
private void ChangeLeft(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    LengthConverter myLengthConverter = new LengthConverter();
    Double db1 =
```

```
(Double)myLengthConverter.ConvertFromString(li.Content.ToString());
    Canvas.SetLeft(text1, db1);
    String st1 =
(String)myLengthConverter.ConvertToString(Canvas.GetLeft(text1));
    canvasLeft.Text = "Canvas.Left = " + st1;
}
```

另请参阅

- [Canvas](#)
- [ListBoxItem](#)
- [LengthConverter](#)
- [面板概述](#)

如何：创建和使用画布

项目 • 2023/02/06

此示例演示如何创建和使用 [Canvas](#) 的实例。

示例

下面的示例通过使用 [Canvas](#) 的 [SetTop](#) 和 [SetLeft](#) 方法显式定位两个 [TextBlock](#) 元素。该示例还将 `LightSteelBlue` 的 [Background](#) 颜色分配给 [Canvas](#)。

① 备注

使用 Extensible Application Markup Language (XAML) 定位 [TextBlock](#) 元素时，请使用 [Top](#) 和 [Left](#) 属性。

C#

```
private void CreateAndShowMainWindow()
{
    // Create the application's main window
    mainWindow = new Window();

    // Create a canvas sized to fill the window
    Canvas myCanvas = new Canvas();
    myCanvas.Background = Brushes.LightSteelBlue;

    // Add a "Hello World!" text element to the Canvas
    TextBlock txt1 = new TextBlock();
    txt1.FontSize = 14;
    txt1.Text = "Hello World!";
    Canvas.SetTop(txt1, 100);
    Canvas.SetLeft(txt1, 10);
    myCanvas.Children.Add(txt1);

    // Add a second text element to show how absolute positioning works
    // in a Canvas
    TextBlock txt2 = new TextBlock();
    txt2.FontSize = 22;
    txt2.Text = "Isn't absolute positioning handy?";
    Canvas.SetTop(txt2, 200);
    Canvas.SetLeft(txt2, 75);
    myCanvas.Children.Add(txt2);
    mainWindow.Content = myCanvas;
    mainWindow.Title = "Canvas Sample";
    mainWindow.Show();
}
```

另请参阅

- [Canvas](#)
- [TextBlock](#)
- [SetTop](#)
- [SetLeft](#)
- [Top](#)
- [Left](#)
- [面板概述](#)
- [操作指南主题](#)

如何：使用画布的附加属性来定位子元素

项目 • 2023/02/06

此示例演示如何使用 [Canvas](#) 的附加属性来定位子元素。

示例

下面的示例将四个 [Button](#) 元素添加为父 [Canvas](#) 的子元素。 每个元素都由 [Bottom](#)、[Left](#)、[Right](#) 和 [Top](#) 表示。 每个 [Button](#) 相对于父 [Canvas](#) 并根据其分配的属性值定位。

C#

```
// Create the application's main window
mainWindow = new Window();
mainWindow.Title = "Canvas Attached Properties Sample";

// Add a Border
Border myBorder = new Border();
myBorder.HorizontalAlignment = HorizontalAlignment.Left;
myBorder.VerticalAlignment = VerticalAlignment.Top;
myBorder.BorderBrush = Brushes.Black;
myBorder.BorderThickness = new Thickness(2);

// Create the Canvas
Canvas myCanvas = new Canvas();
myCanvas.Background = Brushes.LightBlue;
myCanvas.Width = 400;
myCanvas.Height = 400;

// Create the child Button elements
Button myButton1 = new Button();
Button myButton2 = new Button();
Button myButton3 = new Button();
Button myButton4 = new Button();

// Set Positioning attached properties on Button elements
Canvas.SetTop(myButton1, 50);
myButton1.Content = "Canvas.Top=50";
Canvas.SetBottom(myButton2, 50);
myButton2.Content = "Canvas.Bottom=50";
Canvas.SetLeft(myButton3, 50);
myButton3.Content = "Canvas.Left=50";
Canvas.SetRight(myButton4, 50);
myButton4.Content = "Canvas.Right=50";

// Add Buttons to the Canvas' Children collection
myCanvas.Children.Add(myButton1);
myCanvas.Children.Add(myButton2);
```

```
myCanvas.Children.Add(myButton3);
myCanvas.Children.Add(myButton4);

// Add the Canvas as the lone Child of the Border
myBorder.Child = myCanvas;

// Add the Border as the Content of the Parent Window Object
mainWindow.Content = myBorder;
mainWindow.Show();
```

另请参阅

- [Canvas](#)
- [Bottom](#)
- [Left](#)
- [Right](#)
- [Top](#)
- [Button](#)
- [面板概述](#)
- [操作指南主题](#)
- [附加属性概述](#)

如何：使用 Thumb 调整画布的大小

项目 • 2023/02/06

此示例演示如何使用 `Thumb` 控件调整 `Canvas` 控件的大小。

示例

Thumb 控件提供拖动功能，可用于通过监视 Thumb 的 DragStarted、DragDelta 和 DragCompleted 事件来移动或调整控件大小。

当鼠标指针停在 `Thumb` 控件上时，用户通过按下鼠标左键开始拖动操作。只要按住鼠标左键，拖动操作就会继续执行。在拖动操作期间，`DragDelta` 可多次发生。每次发生时，`DragDeltaEventArgs` 类都会提供与鼠标位置更改相对应的位置更改。用户释放鼠标左键时，拖动操作完成。拖动操作仅提供新坐标；它不会自动重新定位 `Thumb`。

以下示例显示了作为 `Canvas` 控件子元素的 `Thumb` 控件。其 `DragDelta` 事件的事件处理程序提供了移动 `Thumb` 和调整 `Canvas` 大小的逻辑。`DragStarted` 和 `DragCompleted` 事件的事件处理程序在拖动操作期间更改 `Thumb` 的颜色。以下示例定义了 `Thumb`。

XAML

```
<Thumb Name="myThumb" Canvas.Left="80" Canvas.Top="80" Background="Blue"
      Width="20" Height="20" DragDelta="onDragDelta"
      DragStarted="onDragStarted" DragCompleted="onDragCompleted"
      />
```

以下示例显示了移动 `Thumb` 并调整 `Canvas` 大小以响应鼠标移动的 `DragDelta` 事件处理程序。

C#

```
void onDragDelta(object sender, DragDeltaEventArgs e)
{
    //Move the Thumb to the mouse position during the drag operation
    double yadjust = myCanvasStretch.Height + e.VerticalChange;
    double xadjust = myCanvasStretch.Width + e.HorizontalChange;
    if ((xadjust >= 0) && (yadjust >= 0))
    {
        myCanvasStretch.Width = xadjust;
        myCanvasStretch.Height = yadjust;
        Canvas.SetLeft(myThumb, Canvas.GetLeft(myThumb) +
                        e.HorizontalChange);
        Canvas.SetTop(myThumb, Canvas.GetTop(myThumb) +
                        e.VerticalChange);
        changes.Text = "Size: " +
                        myCanvasStretch.Width.ToString() +
                        " x " +
                        myCanvasStretch.Height.ToString();
    }
}
```

```
        ", " +
        myCanvasStretch.Height.ToString());
    }
}
```

以下示例显示了 [DragStarted](#) 事件处理程序。

C#

```
void onDragStarted(object sender, DragStartedEventArgs e)
{
    myThumb.Background = Brushes.Orange;
}
```

以下示例显示了 [DragCompleted](#) 事件处理程序。

C#

```
void onDragCompleted(object sender, DragCompletedEventArgs e)
{
    myThumb.Background = Brushes.Blue;
}
```

有关完整示例，请参阅 [Thumb 拖动功能示例](#)。

另请参阅

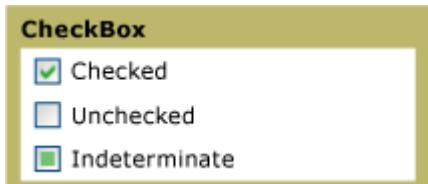
- [Thumb](#)
- [DragStarted](#)
- [DragDelta](#)
- [DragCompleted](#)

CheckBox

项目 • 2023/02/06

可以在应用程序的用户界面 (UI) 中使用 [CheckBox](#) 来表示用户可以选择或清除的选项。可以使用一个复选框，也可以对两个或多个复选框进行分组。

下图显示了 [CheckBox](#) 的不同状态。



处于不同状态的 CheckBox 控件

参考

[CheckBox](#)

[RadioButton](#)

[ButtonBase](#)

[RepeatButton](#)

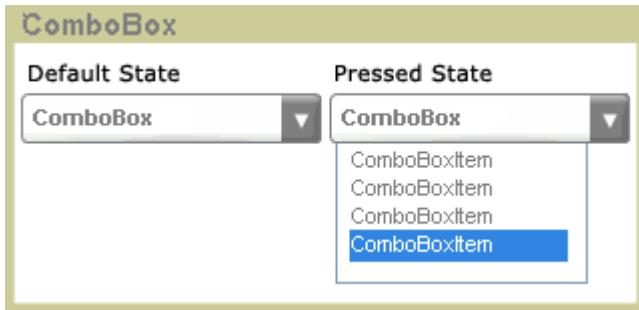
相关章节

ComboBox

项目 • 2023/02/06

ComboBox 控件向用户显示选项列表。当控件展开和折叠时，列表将显示和隐藏。在默认状态下，列表是折叠的，仅显示一个选项。用户可单击按钮查看完整的选项列表。

下图显示了处于不同状态的 ComboBox。



折叠和展开

参考

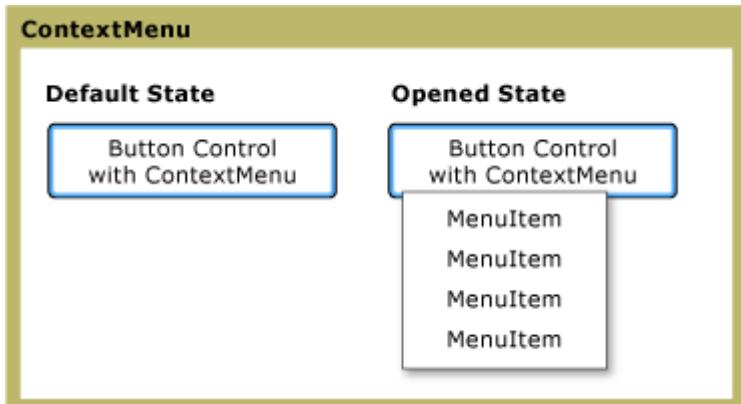
[ComboBox](#)

ContextMenu

项目 • 2023/02/06

[ContextMenu](#) 允许控件显示特定于该控件上下文的 Menu。通常，[ContextMenu](#) 通过鼠标鼠标右键或键盘的菜单按钮在用户界面 (UI) 中公开。

下图演示了处于两种不同状态的 [ContextMenu](#)：默认状态和打开状态。在默认状态下，控件是折叠的。当在菜单的父级上按下鼠标右键时，控件将展开并显示菜单项。



处于不同状态的 ContextMenu

本节内容

[ContextMenu 概述](#)

参考

[ContextMenu](#)

相关章节

ContextMenu 概述

项目 • 2023/02/06

ContextMenu 类表示使用特定于上下文的 Menu 来公开功能的元素。通常，用户通过单击鼠标右键在用户界面 (UI) 中公开 ContextMenu。本主题介绍 ContextMenu 元素，并提供有关如何在 Extensible Application Markup Language (XAML) 和代码中使用它的示例。

ContextMenu 控件

ContextMenu 会附加到特定控件。通过使用 ContextMenu 元素，可以向用户呈现一个项列表，这些项指定与特定控件（例如 Button）相关联的命令或选项。用户通过右键单击控件来显示菜单。通常，单击 MenuItem 即可打开子菜单或导致应用程序执行某个命令。

创建 ContextMenus

下面的示例展示了如何创建具有子菜单的 ContextMenu。ContextMenu 控件会附加到按钮控件。

XAML

```
<Button Name="cmButton" Height="30">
    Button with Context Menu
    <Button.ContextMenu>
        <ContextMenu Name="cm" Opened="OnOpened" Closed="OnClosed"
        StaysOpen="true">
            <MenuItem Header="File"/>
            <MenuItem Header="Save"/>
            <MenuItem Header="SaveAs"/>
            <MenuItem Header="Recent Files">
                <MenuItem Header="ReadMe.txt"/>
                <MenuItem Header="Schedule.xls"/>
            </MenuItem>
        </ContextMenu>
    </Button.ContextMenu>
</Button>
```

C#

```
btn = new Button();
btn.Content = "Created with C#";
contextmenu = new ContextMenu();
btn.ContextMenu = contextmenu;
```

```
mi = new MenuItem();
mi.Header = "File";
mia = new MenuItem();
mia.Header = "New";
mi.Items.Add(mia);
mib = new MenuItem();
mib.Header = "Open";
mi.Items.Add(mib);
mib1 = new MenuItem();
mib1.Header = "Recently Opened";
mib.Items.Add(mib1);
mib1a = new MenuItem();
mib1a.Header = "Text.xaml";
mib1.Items.Add(mib1a);
contextmenu.Items.Add(mi);
cv2.Children.Add(btn);
```

将样式应用于 ContextMenu

通过使用控件 [Style](#)，无需编写自定义控件即可显著改变 [ContextMenu](#) 的外观和行为。除了设置可视化属性以外，还可以将样式应用于控件的各个部分。例如，可以使用属性来更改控件各个部件的行为，也可以向 [ContextMenu](#) 添加部件或更改其布局。以下示例展示了向 [ContextMenu](#) 控件添加样式的几种方法。

第一个示例定义一个名为 `SimpleSysResources` 的样式，它演示如何在样式中使用当前的系统设置。此示例将 `MenuHighlightBrushKey` 分配为 [ContextMenu](#) 的 `Background` 颜色并将 `MenuTextBrushKey` 分配为 `Foreground` 颜色。

XAML

```
<Style x:Key="SimpleSysResources" TargetType="{x:Type MenuItem}">
  <Setter Property = "Background" Value=
    "{DynamicResource {x:Static SystemColors.MenuHighlightBrushKey}}"/>
  <Setter Property = "Foreground" Value=
    "{DynamicResource {x:Static SystemColors.MenuTextBrushKey}}"/>
</Style>
```

以下示例使用 `Trigger` 元素来更改 `Menu` 的外观，从而响应在 [ContextMenu](#) 上引发的事件。当用户将鼠标移到菜单上时，[ContextMenu](#) 项的外观将随之更改。

XAML

```
<Style x:Key="Triggers" TargetType="{x:Type MenuItem}">
  <Style.Triggers>
    <Trigger Property="MenuItem.IsMouseOver" Value="true">
      <Setter Property = "FontSize" Value="16"/>
      <Setter Property = "FontStyle" Value="Italic"/>
      <Setter Property = "Foreground" Value="Red"/>
    </Trigger>
  </Style.Triggers>
</Style>
```

```
</Trigger>
</Style.Triggers>
</Style>
```

另请参阅

- [ContextMenu](#)
- [Style](#)
- [Menu](#)
- [MenuItem](#)
- [ContextMenu](#)
- [ContextMenu 样式和模板](#)
- [WPF 控件库示例](#)

DataGrid

项目 • 2023/02/06

通过 [DataGrid](#) 控件可显示和编辑来自许多不同源的数据，例如来自 SQL 数据库、LINQ 查询或任何其他可绑定数据源的数据。有关详细信息，请参阅[绑定源概述](#)。

列可以显示文本、控件（例如 [ComboBox](#)）或任何其他 WPF 内容（例如图像、按钮或模板中包含的任何内容）。可以使用 [DataGridTemplateColumn](#) 显示模板中定义的数据。下表列出了默认提供的列类型。

生成的列类型	数据类型
DataGridTextColumn	String
DataGridCheckBoxColumn	Boolean
DataGridComboBoxColumn	Enum
DataGridHyperlinkColumn	Uri

[DataGrid](#) 可以自定义外观，例如单元格字体、颜色和大小。[DataGrid](#) 支持其他 WPF 控件的所有样式设置和模板功能。[DataGrid](#) 还包括用于编辑、排序和验证的默认和可自定义行为。

下表列出了 [DataGrid](#) 的一些常见任务以及如何完成这些任务。通过查看相关的 API，可以找到详细信息和示例代码。

场景	方法
交替背景色	将 AlternationIndex 属性设置为 2 或更多，然后将 Brush 分配给 RowBackground 和 AlternatingRowBackground 属性。
定义单元格和行选择行为	设置 SelectionMode 和 SelectionUnit 属性。
自定义图标、单元格和行的可视外观	将新的 Style 应用于 ColumnHeaderStyle 、 RowHeaderStyle 、 CellStyle 或 RowStyle 属性。
设置大小调整选项	设置 Height 、 MaxHeight 、 MinHeight 、 Width 、 MaxWidth 或 MinWidth 属性。有关详细信息，请参阅 DataGrid 控件中的调整大小选项 。

场景	方法
访问项	检查 SelectedCells 属性以获取选定的单元格，检查 SelectedItems 属性以获取选定的行。有关详细信息，请参阅 所选 SelectedCells。
自定义最终权用	设置 CanUserAddRows 、 CanUserDeleteRows 、 CanUserReorderColumns 、 CanUserResizeColumns 、 CanUserResizeRows 和 CanUserSortColumns 属性。
户交互	
取消或更改自动生成的列	处理 AutoGeneratingColumn 事件。
冻结列	将 FrozenColumnCount 属性设置为 1，并通过将 DisplayIndex 属性设置为 0 将列移动到最左边的位置。
使用数据作为数据源	将 DataGrid 上的 .ItemsSource 绑定到表示项集合的 XPath 查询。在 DataGrid 中创建每一列。通过将绑定上的 XML 设置为获取项源属性的查询来绑定每一列。有关示例，请参见 DataGridTextColumn 。

相关主题

Title	描述
演练：在 DataGrid 控件中显示 SQL Server 数据库中的数据	介绍如何设置新的 WPF 项目、添加实体框架元素、设置源以及在 DataGrid 中显示数据。
如何：向 DataGrid 控件中添加行详细信息	介绍如何为 DataGrid 创建行详细信息。
如何：使用 DataGrid 控件实现验证	介绍如何验证 DataGrid 单元格和行中的值，并显示验证反馈。
DataGrid 控件中的默认键盘和鼠标行为	介绍如何使用键盘和鼠标与 DataGrid 控件进行交互。
如何：在 DataGrid 控件中对数据进行分组、排序和筛选	介绍如何通过对数据进行分组、排序和筛选，以不同方式查看 DataGrid 中的数据。
DataGrid 控件中的调整大小选项	介绍如何在 DataGrid 中控制绝对和自动大小调整。

请参阅

- [DataGrid](#)
- [样式设置和模板化](#)
- [数据绑定概述](#)
- [数据模板化概述](#)

- 控件
- WPF 内容模型

DataGrid 控件中的默认键盘和鼠标行为

项目 • 2023/02/06

本主题介绍用户如何使用键盘和鼠标与 [DataGrid](#) 控件进行交互。

与 [DataGrid](#) 的典型交互包括导航、选择和编辑。选择行为受 [SelectionMode](#) 和 [SelectionUnit](#) 属性的影响。导致本主题所述行为的默认值是 [DataGridSelectionMode.Extended](#) 和 [DataGridSelectionUnit.FullRow](#)。更改这些值可能会导致行为与描述不同。单元格处于编辑模式时，编辑控件可能会覆盖 [DataGrid](#) 的标准键盘行为。

默认键盘行为

下表列出了 [DataGrid](#) 的默认键盘行为。

键或键组	描述
合	
向下键	将焦点移至当前单元格正下方的单元格。如果焦点在最后一行，则按向下箭头不执行任何操作。
向上键	将焦点移至当前单元格正上方的单元格。如果焦点在第一行，则按向上箭头不执行任何操作。
向左键	将焦点移至行中的上一个单元格。如果焦点在行中的第一个单元格，则按向左箭头不执行任何操作。
向右键	将焦点移至行中的下一个单元格。如果焦点在行中的最后一个单元格，则按向右箭头不执行任何操作。
Home	将焦点移至当前行中的第一个单元格。
End	将焦点移至当前行中的最后一个单元格。
Page Down	如果行未分组，则按完全显示的行数向下滚动控件。在不更改列的情况下，将焦点移至最后一个完全显示的行。 如果行已分组，则在不更改列的情况下，将焦点移至 DataGrid 中的最后一行。
Page Up	如果行未分组，则按完全显示的行数向上滚动控件。在不更改列的情况下，将焦点移至显示的第一行。 如果行已分组，则在不更改列的情况下，将焦点移至 DataGrid 中的第一行。

键或键组合	描述
Tab	将焦点移至当前行中的下一个单元格。如果焦点位于该行的最后一个单元格，则将焦点移至下一行的第一个单元格。如果焦点位于控件中的最后一个单元格，则将焦点移至父容器的 Tab 键顺序中的下一个控件。 如果当前单元格处于编辑模式并且按 TAB 使焦点从当前行移开，则在焦点更改之前会提交对该行所做的任何更改。
Shift+Tab	将焦点移至当前行中的上一个单元格。如果焦点已经位于该行的第一个单元格，则将焦点移至上一行的最后一个单元格。如果焦点位于控件中的第一个单元格，则将焦点移至父容器的 Tab 键顺序中的上一个控件。 如果当前单元格处于编辑模式并且按 TAB 使焦点从当前行移开，则在焦点更改之前会提交对该行所做的任何更改。
Ctrl+向下键	将焦点移至当前列中的最后一个单元格。
CTRL + 向上箭头	将焦点移至当前列中的第一个单元格。
Ctrl+向右键	将焦点移至当前行中的最后一个单元格。
Ctrl+向左键	将焦点移至当前行中的第一个单元格。
Ctrl+Home	将焦点移至控件中的第一个单元格。
Ctrl+End	将焦点移至控件中的最后一个单元格。
Ctrl+PAGE DOWN	与 PAGE DOWN 相同。
Ctrl+PAGE UP	与 PAGE UP 相同。
F2	如果当前列的 <code>DataGrid.IsReadOnly</code> 属性为 <code>false</code> 且 <code>DataGridColumn.IsReadOnly</code> 属性为 <code>false</code> ，则将当前单元格置于单元格编辑模式。
Enter	提交对当前单元格和行的任何更改，并将焦点移至当前单元格正下方的单元格。如果焦点在最后一行，则在不移动焦点的情况下提交所有更改。
ESC	如果控件处于编辑模式，则取消编辑并还原在控件中所做的所有更改。如果基础数据源实现 <code>IEditableObject</code> ，则再次按 ESC 将取消整行的编辑模式。
Backspace	编辑单元格时删除光标前的字符。
删除	编辑单元格时删除光标后的字符。

键或键组	描述
Ctrl+Enter	在不移动焦点的情况下，提交对当前单元格所做的任何更改。
CTRL + A	如果 SelectionMode 设置为 Extended ，则选择 DataGrid 中的所有行。

选择键

如果 [SelectionMode](#) 属性设置为 [Extended](#)，则导航行为不会改变，但在按下 SHIFT (包括 CTRL+SHIFT) 的同时使用键盘导航将修改多行选定内容。导航开始前，控件将当前行标记为定位行。如果在按下 Shift 的同时进行导航，所选内容将包括定位行和当前行之间的所有行。

以下选择键可修改多行选定内容。

- SHIFT + 向下键
- Shift+向上键
- SHIFT + Page Down
- SHIFT + PAGE UP
- Ctrl+Shift+向下键
- Ctrl+Shift+向上键
- Ctrl+Shift+Home
- Ctrl+Shift+End

默认鼠标行为

下表列出了 [DataGrid](#) 的默认鼠标行为。

鼠标操作	描述
单击未选中的行	将单击的行设置为当前行，并将单击的单元格设置为当前单元格。
单击当前单元格	将当前单元格置于编辑模式下。
拖动列标头单元格	如果当前列的 DataGrid.CanUserReorderColumns 属性为 <code>true</code> 且 DataGridColumn.CanUserReorder 属性为 <code>true</code> ，则移动该列，将其拖放到新位置。

鼠标操作	描述
拖动列标头分隔符	如果当前列的 <code>DataGridView.CanUserResizeColumns</code> 属性为 <code>true</code> 且 <code>DataGridViewColumn.CanUserResize</code> 属性为 <code>true</code> ，则调整该列的大小。
双击列标头分隔符	如果当前列的 <code>DataGridView.CanUserResizeColumns</code> 属性为 <code>true</code> 且 <code>DataGridViewColumn.CanUserResize</code> 属性为 <code>true</code> ，则使用 <code>Auto</code> 大小调整模式自动调整该列大小。
单击列标头单元格	如果当前列的 <code>DataGridView.CanUserSortColumns</code> 属性为 <code>true</code> 且 <code>DataGridViewColumn.CanUserSort</code> 属性为 <code>true</code> ，则对该列进行排序。 单击已排序的列的标头会反转该列的排序方向。 单击多个列标头的同时按下 <code>SHIFT</code> 键会按单击的顺序对多个列进行排序。
对行进行 <code>CTRL+click</code> 操作	如果 <code>SelectionMode</code> 设置为 <code>Extended</code> ，则修改非连续的多行选定内容。 如果该行已选中，则取消选中该行。
对行进行 <code>SHIFT+click</code> 操作	如果 <code>SelectionMode</code> 设置为 <code>Extended</code> ，则修改连续的多行选定内容。
单击行组标头	展开或折叠组。
单击 <code>DataGridView</code> 左上角的“全选”按钮	如果 <code>SelectionMode</code> 设置为 <code>Extended</code> ，则选择 <code>DataGridView</code> 中的所有行。

鼠标选择

如果 `SelectionMode` 属性设置为 `Extended`，则在按下 `CTRL` 或 `SHIFT` 的同时单击某一行将修改多行选定内容。

如果在按下 `CTRL` 的同时单击某一行，该行将更改其选择状态，而所有其他行保持其当前选择状态。执行此操作可选择不相邻的行。

按下 `SHIFT` 的同时单击某一行时，所选内容包括当前行与定位行（单击前当前行所在行）之间的所有行。按下 `SHIFT` 时的后续单击会更改当前行，但不会更改定位行。执行此操作可选择相邻行范围。

可以结合使用 `CTRL+SHIFT` 选择相邻行的不相邻范围。若要实现这一点，则如前所述，使用 `SHIFT+click` 选择第一个范围。选择第一个行范围后，使用 `CTRL+click` 选择下一个范围中的第一行，然后在按下 `CTRL+SHIFT` 的同时单击下一个范围中的最后一行。

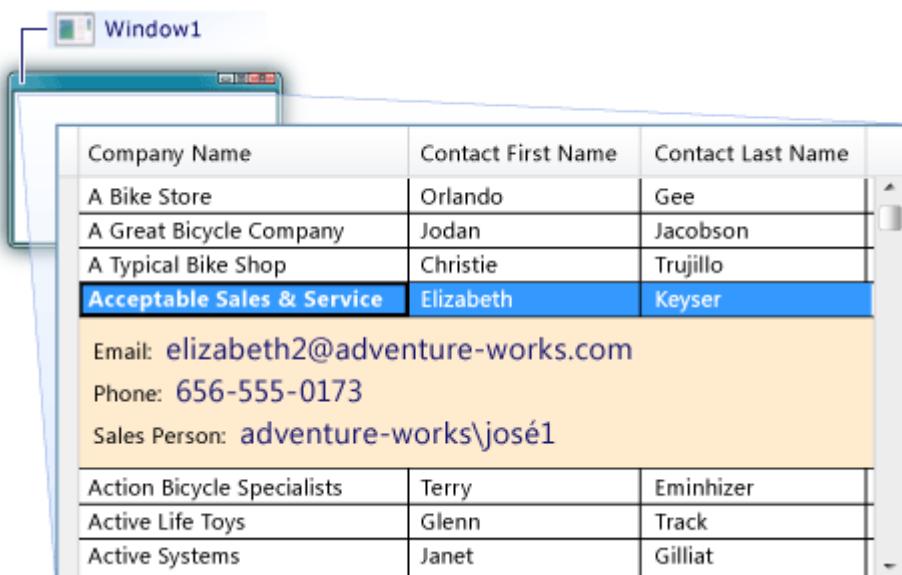
另请参阅

- [DataGrid](#)
- [SelectionMode](#)

如何：向 DataGrid 控件中添加行详细信息

项目 • 2023/02/06

当使用 [DataGrid](#) 控件时，可以通过添加行详细信息部分来自定义数据表示形式。通过添加行详细信息部分，可对模板中的一些可见或折叠（可选）数据进行分组。例如，可以向 [DataGrid](#) 添加行详细信息，它仅显示 [DataGrid](#) 中每一行的数据摘要，但在用户选择行时将显示更多数据字段。在 [RowDetailsTemplate](#) 属性中定义行详细信息部分的模板。下图显示行详细信息部分的示例。



将行详细信息模板定义为内联 XAML 或资源。以下过程中显示了这两种方法。可在整个项目中使用作为资源添加的数据模板，而无需重新创建模板。作为内联 XAML 添加的数据模板只能从定义它的控件访问。

使用内联 XAML 显示行详细信息

1. 创建显示来自数据源的数据的 [DataGrid](#)。
2. 在 [DataGrid](#) 元素内，添加一个 [RowDetailsTemplate](#) 元素。
3. 创建定义行详细信息部分外观的 [DataTemplate](#)。

以下 XAML 显示 [DataGrid](#) 以及如何定义 [RowDetailsTemplate](#) 内联。[DataGrid](#) 将在每行中显示三个值，并在选择该行时显示另外三个值。

XAML

```
<Window x:Class="WpfApplication1.MainWindow"
```

```

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="525"
    Loaded="Window_Loaded">
    <Grid>
        <DataGrid Name="dataGrid1" IsReadOnly="True"
        AutoGenerateColumns="False" >
            <DataGrid.Columns>
                <DataGridTextColumn Header="Company Name" Binding=""
{Binding CompanyName}></DataGridTextColumn>
                <DataGridTextColumn Header="Contact First Name"
Binding="{Binding FirstName}"></DataGridTextColumn>
                <DataGridTextColumn Header="Contact Last Name"
Binding="{Binding LastName}"></DataGridTextColumn>
            </DataGrid.Columns>
            <DataGrid.RowDetailsTemplate>
                <DataTemplate>
                    <Border BorderThickness="0"
Background="BlanchedAlmond" Padding="10">
                        <StackPanel Orientation="Vertical">
                            <StackPanel Orientation="Horizontal">
                                <TextBlock FontSize="12" Text="Email: "
VerticalAlignment="Center" />
                                    <TextBlock FontSize="16"
Foreground="MidnightBlue" Text="{Binding EmailAddress}"
VerticalAlignment="Center" />
                                </StackPanel>
                                <StackPanel Orientation="Horizontal">
                                    <TextBlock FontSize="12" Text="Phone: "
VerticalAlignment="Center" />
                                    <TextBlock FontSize="16"
Foreground="MidnightBlue" Text="{Binding Phone}"
VerticalAlignment="Center" />
                                </StackPanel>
                                <StackPanel Orientation="Horizontal">
                                    <TextBlock FontSize="12" Text="Sales
Person: " VerticalAlignment="Center" />
                                    <TextBlock FontSize="16"
Foreground="MidnightBlue" Text="{Binding SalesPerson}"
VerticalAlignment="Center" />
                                </StackPanel>
                            </StackPanel>
                        </Border>
                    </DataTemplate>
                </DataGrid.RowDetailsTemplate>
            </DataGrid>
        </Grid>
    </Window>

```

以下代码显示了一个查询，该查询用于选择在 DataGrid 中显示的数据。在此示例中，查询从包含客户信息的实体中选择数据。

C#

```

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    AdventureWorksLT2008Entities advenWorksEntities = new
AdventureWorksLT2008Entities();

    ObjectQuery<Customer> customers = advenWorksEntities.Customers;

    var query =
        from customer in customers
        orderby customer.CompanyName
        select new
        {
            customer.LastName,
            customer.FirstName,
            customer.CompanyName,
            customer.Title,
            customer.EmailAddress,
            customer.Phone,
            customer.SalesPerson
        };

    dataGrid1.ItemsSource = query.ToList();
}

```

使用资源显示行详细信息

1. 创建显示来自数据源的数据的 [DataGrid](#)。
2. 将 [Resources](#) 元素添加到根元素（例如 [Window](#) 控件或 [Page](#) 控件），或将 [Resources](#) 元素添加到 [App.xaml](#)（或 [Application.xaml](#)）文件中的 [Application](#) 类。
3. 在资源元素中，创建定义行详细信息部分外观的 [DataTemplate](#)。

以下 XAML 显示了 [Application](#) 类中定义的 [RowDetailsTemplate](#)。

XAML

```

<Application.Resources>
    <DataTemplate x:Key="CustomerDetail">
        <Border BorderThickness="0" Background="BlanchedAlmond"
Padding="10">
            <StackPanel Orientation="Vertical">
                <StackPanel Orientation="Horizontal">
                    <TextBlock FontSize="12" Text="Email: " 
VerticalAlignment="Center" />
                    <TextBlock FontSize="16" Foreground="MidnightBlue"
Text="{Binding EmailAddress}" VerticalAlignment="Center" />
                </StackPanel>
                <StackPanel Orientation="Horizontal">
                    <TextBlock FontSize="12" Text="Phone: " 
VerticalAlignment="Center" />

```

```

        <TextBlock FontSize="16" Foreground="MidnightBlue"
Text="{Binding Phone}" VerticalAlignment="Center" />
    </StackPanel>
    <StackPanel Orientation="Horizontal">
        <TextBlock FontSize="12" Text="Sales Person: " VerticalAlignment="Center" />
        <TextBlock FontSize="16" Foreground="MidnightBlue" Text="{Binding SalesPerson}" VerticalAlignment="Center" />
    </StackPanel>
</Border>
</DataTemplate>
</Application.Resources>

```

4. 在 `DataTemplate` 上，将 `x:Key` 指令设置为唯一标识数据模板的值。
5. 在 `DataGridView` 元素中，将 `RowDetailsTemplate` 属性设置为在前面步骤中定义的资源。将资源分配为静态资源。

以下 XAML 显示了为上一个示例中的资源设置的 `RowDetailsTemplate` 属性。

XAML

```

<DataGrid Name="dataGridView1" IsReadOnly="True"
AutoGenerateColumns="False" RowDetailsTemplate="{StaticResource CustomerDetail}" >
    <DataGrid.Columns>
        <DataGridTextColumn Header="Company Name" Binding="{Binding CompanyName}"></DataGridTextColumn>
        <DataGridTextColumn Header="Contact First Name" Binding="{Binding FirstName}"></DataGridTextColumn>
        <DataGridTextColumn Header="Contact Last Name" Binding="{Binding LastName}"></DataGridTextColumn>
    </DataGrid.Columns>
</DataGrid>

```

设置可见性并防止水平滚动行详细信息

1. 如有必要，请将 `RowDetailsVisibilityMode` 属性设置为 `DataGridViewRowDetailsVisibilityMode` 值。

默认情况下，该值设置为 `VisibleWhenSelected`。可以将其设置为 `Visible` 以显示所有行的详细信息，或将其设置为 `Collapsed` 以隐藏所有行的详细信息。

2. 如有必要，请将 `AreRowDetailsFrozen` 属性设置为 `true`，以防止行详细信息部分水平滚动。

如何：在 DataGrid 控件中对数据进行分组、排序和筛选

项目 • 2022/09/27

通过对数据进行分组、排序和筛选，以不同的方式查看 DataGrid 中的数据通常很有用。要对 DataGrid 中的数据进行分组、排序和筛选，可以将其绑定到支持这些函数的 CollectionView。然后，可以在不影响基础源数据的情况下处理 CollectionView 中的数据。集合视图中的更改反映在 DataGrid 用户界面 (UI) 中。

CollectionView 类为实现 `IEnumerable` 接口的数据源提供分组和排序功能。

CollectionViewSource 类允许你从 XAML 设置 CollectionView 的属性。

在此示例中，`Task` 对象的集合绑定到一个 CollectionViewSource。

CollectionViewSource 用作 DataGrid 的 `ItemsSource`。对 CollectionViewSource 执行分组、排序和筛选，并在 DataGrid UI 中显示。

Group, Sort, and Filter Example				
ProjectName	TaskName	DueDate	Complete	
Project 1		4		
Active		2		
Project 1	Task 3	3/7/2010 10:51:13 AM	<input type="checkbox"/>	
Project 1	Task 9	3/13/2010 10:51:13 AM	<input type="checkbox"/>	
Complete		2		
Project 1	Task 6	3/10/2010 10:51:13 AM	<input checked="" type="checkbox"/>	
Project 1	Task 12	3/16/2010 10:51:13 AM	<input checked="" type="checkbox"/>	
Project 2		5		
Active		3		
Project 2	Task 1	3/5/2010 10:51:13 AM	<input type="checkbox"/>	
Project 2	Task 7	3/11/2010 10:51:13 AM	<input type="checkbox"/>	
Project 2	Task 13	3/17/2010 10:51:13 AM	<input type="checkbox"/>	
Complete		2		
Project 2	Task 4	3/8/2010 10:51:13 AM	<input checked="" type="checkbox"/>	
Project 2	Task 10	3/14/2010 10:51:13 AM	<input checked="" type="checkbox"/>	
Project 3		5		
Active		2		
Project 3	Task 5	3/9/2010 10:51:13 AM	<input type="checkbox"/>	
Project 3	Task 11	3/15/2010 10:51:13 AM	<input type="checkbox"/>	
Complete		3		
Project 3	Task 2	3/6/2010 10:51:13 AM	<input checked="" type="checkbox"/>	
Project 3	Task 8	3/12/2010 10:51:13 AM	<input checked="" type="checkbox"/>	
Project 3	Task 14	3/18/2010 10:51:13 AM	<input checked="" type="checkbox"/>	

分组数据

DataGrid 中的

将CollectionViewSource 用作 ItemsSource

要对 [DataGrid](#) 控件中的数据进行分组、排序和筛选，可以将 [DataGrid](#) 绑定到支持这些函数的 [CollectionView](#)。在此示例中，[DataGrid](#) 绑定到 [CollectionViewSource](#)，后者为 [Task](#) 对象的 [List<T>](#) 提供这些函数。

将 DataGrid 绑定到 CollectionViewSource

1. 创建实现 [IEnumerable](#) 接口的数据集合。

如果使用 [List<T>](#) 创建集合，则应创建继承自 [List<T>](#) 的新类，而不是实例化 [List<T>](#) 实例。这使你能够在 XAML 中将数据绑定到集合。

① 备注

集合中的对象必须实现 [INotifyPropertyChanged](#) 已更改的接口和 [IEditableObject](#) 接口，以便 [DataGrid](#) 能够正确响应属性的更改和编辑。有关详细信息，请参阅[实现属性更改通知](#)。

C#

```
// Requires using System.Collections.ObjectModel;
public class Tasks : ObservableCollection<Task>
{
    // Creating the Tasks collection in this way enables data binding
    // from XAML.
}
```

2. 在 XAML 中，创建集合类的实例并设置 [x:Key](#) 指令。

3. 在 XAML 中，创建 [CollectionViewSource](#) 类的实例并设置 [x:Key](#) 指令，然后将集合类的实例设置为 [Source](#)。

XAML

```
<Window.Resources>
    <local:Tasks x:Key="tasks" />
    <CollectionViewSource x:Key="cvsTasks" Source="{StaticResource
tasks}" Filter="CollectionViewSource_Filter">
    </CollectionViewSource>
</Window.Resources>
```

4. 创建 `DataGridView` 类的一个实例，并将 `ItemsSource` 属性设置为 `CollectionViewSource`。

XAML

```
<DataGrid x:Name="dataGridView1"
          ItemsSource="{Binding Source={StaticResource cvsTasks}}"
          CanUserAddRows="False">
```

5. 要从代码中访问 `CollectionViewSource`，请使用 `GetDefaultView` 方法获取对 `CollectionViewSource` 的引用。

C#

```
ICollectionView cvTasks =
CollectionViewSource.GetDefaultView(dataGridView1.ItemsSource);
```

对 `DataGridView` 中的项进行分组

要指定如何对 `DataGridView` 中的项分组，请使用 `PropertyGroupDescription` 类型对源视图中的项进行分组。

使用 XAML 对 `DataGridView` 中的项进行分组

1. 创建一个 `PropertyGroupDescription`，指定要按其分组的属性。可以在 XAML 或代码中指定该属性。
 - a. 在 XAML 中，将 `PropertyName` 设置要按其分组的属性的名称。
 - b. 在代码中，将按其分组的属性的名称传递给构造函数。
2. 将 `PropertyGroupDescription` 添加到 `CollectionViewSource.GroupDescriptions` 集合中。
3. 将 `PropertyGroupDescription` 的其他实例添加到 `GroupDescriptions` 集合，以添加更多级别的分组。

XAML

```
<CollectionViewSource.GroupDescriptions>
  <PropertyGroupDescription PropertyName="ProjectName"/>
  <PropertyGroupDescription PropertyName="Complete"/>
</CollectionViewSource.GroupDescriptions>
```

C#

```
ICollectionView cvTasks =  
CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource);  
if (cvTasks != null && cvTasks.CanGroup == true)  
{  
    cvTasks.GroupDescriptions.Clear();  
    cvTasks.GroupDescriptions.Add(new  
PropertyGroupDescription("ProjectName"));  
    cvTasks.GroupDescriptions.Add(new  
PropertyGroupDescription("Complete"));  
}
```

4. 要删除该组，请从 [GroupDescriptions](#) 集合中删除 [PropertyGroupDescription](#)。

5. 要删除所有组，请调用 [GroupDescriptions](#) 集合的 [Clear](#) 方法。

C#

```
ICollectionView cvTasks =  
CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource);  
if (cvTasks != null)  
{  
    cvTasks.GroupDescriptions.Clear();  
}
```

在 [DataGrid](#) 中对项进行分组后，可以定义一个 [GroupStyle](#) 来指定每个组的外观。通过将 [GroupStyle](#) 添加到 [DataGrid](#) 的 [GroupStyle](#) 集合来应用它。如果你有多个级别的分组，则可以对每个级别的组应用不同的样式。样式按照定义顺序应用。例如，如果定义两种样式，则第一个样式将应用于顶级行组。第二种样式将应用于第二级或更低级别的所有行组。[GroupStyle](#) 的 [DataContext](#) 就是该组所代表的 [CollectionViewGroup](#)。

更改行组标题的外观

1. 创建定义行组外观的 [GroupStyle](#)。

2. 将 [GroupStyle](#) 放入 [`<DataGrid.GroupStyle>`](#) 标记中。

XAML

```
<DataGrid.GroupStyle>  
    <!-- Style for groups at top level. -->  
    <GroupStyle>  
        <GroupStyle.ContainerStyle>  
            <Style TargetType="{x:Type GroupItem}">  
                <Setter Property="Margin" Value="0,0,0,5"/>  
                <Setter Property="Template">  
                    <Setter.Value>
```

```

        <ControlTemplate TargetType="{x:Type
GroupItem}">
            <Expander IsExpanded="True"
Background="#FF112255" BorderBrush="#FF002255" Foreground="#FFFFFF"
BorderThickness="1,1,1,5">
                <Expander.Header>
                    <DockPanel>
                        <TextBlock FontWeight="Bold"
Text="{Binding Path=Name}" Margin="5,0,0,0" Width="100"/>
                        <TextBlock FontWeight="Bold"
Text="{Binding Path=ItemCount}"/>
                    </DockPanel>
                </Expander.Header>
                <Expander.Content>
                    <ItemsPresenter />
                </Expander.Content>
            </Expander>
        </ControlTemplate>
    </Setter.Value>
</Setter>
</Style>
</GroupStyle.ContainerStyle>
</GroupStyle>
<!-- Style for groups under the top level. --&gt;
&lt;GroupStyle&gt;
    &lt;GroupStyle.HeaderTemplate&gt;
        &lt;DataTemplate&gt;
            &lt;DockPanel Background="LightBlue"&gt;
                &lt;TextBlock Text="{Binding Path=Name, Converter=
{StaticResource completeConverter}}" Foreground="Blue"
Margin="30,0,0,0" Width="100"/&gt;
                &lt;TextBlock Text="{Binding Path=ItemCount}"
Foreground="Blue"/&gt;
            &lt;/DockPanel&gt;
        &lt;/DataTemplate&gt;
    &lt;/GroupStyle.HeaderTemplate&gt;
&lt;/GroupStyle&gt;
&lt;/DataGrid.GroupStyle&gt;
</pre>

```

对 DataGrid 中的项进行排序

要指定如何对 DataGrid 中的项进行排序，请使用 [SortDescription](#) 类型对源视图中的项进行排序。

对 DataGrid 中的项进行排序

1. 创建一个 [SortDescription](#)，指定要按其排序的属性。可以在 XAML 或代码中指定该属性。
 - a. 在 XAML 中，将 [PropertyName](#) 设置要按其排序的属性的名称。

- b. 在代码中，将按其排序的属性的名称和 `ListSortDirection` 传递给构造函数。
- 2. 将 `SortDescription` 添加到 `CollectionViewSource.SortDescriptions` 集合中。
- 3. 将 `SortDescription` 的其他实例添加到 `SortDescriptions` 集合，以按其他属性排序。

XAML

```
<CollectionViewSource.SortDescriptions>
    <!-- Requires 'xmlns:scm="clr-
namespace:System.ComponentModel;assembly=WindowsBase"' declaration. -->
    <scm:SortDescription PropertyName="ProjectName" />
    <scm:SortDescription PropertyName="Complete" />
    <scm:SortDescription PropertyName="DueDate" />
</CollectionViewSource.SortDescriptions>
```

C#

```
// Requires using System.ComponentModel;
ICollectionView cvTasks =
CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource);
if (cvTasks != null && cvTasks.CanSort == true)
{
    cvTasks.SortDescriptions.Clear();
    cvTasks.SortDescriptions.Add(new SortDescription("ProjectName",
ListSortDirection.Ascending));
    cvTasks.SortDescriptions.Add(new SortDescription("Complete",
ListSortDirection.Ascending));
    cvTasks.SortDescriptions.Add(new SortDescription("DueDate",
ListSortDirection.Ascending));
}
```

对 DataGrid 中的项进行筛选

要使用 `CollectionViewSource` 筛选 `DataGrid` 中的项，需要在处理程序中为 `CollectionViewSource.Filter` 事件提供筛选逻辑。

筛选 DataGrid 中的项

1. 为 `CollectionViewSource.Filter` 事件添加处理程序。
2. 在 `Filter` 事件处理程序中，定义筛选逻辑。

每次刷新视图时，都会应用该筛选器。

XAML

```
<CollectionViewSource x:Key="cvsTasks" Source="{StaticResource tasks}"
    Filter="CollectionViewSource_Filter">
```

C#

```
private void CollectionViewSource_Filter(object sender, FilterEventArgs e)
{
    Task t = e.Item as Task;
    if (t != null)
        // If filter is turned on, filter completed items.
        {
            if (this.cbCompleteFilter.IsChecked == true && t.Complete ==
true)
                e.Accepted = false;
            else
                e.Accepted = true;
        }
}
```

或者，可以创建一个提供筛选逻辑的方法并设置 `CollectionView.Filter` 属性来应用筛选器，从而筛选 `DataGrid` 中的项。要查看此方法的示例，请参阅[筛选视图中的数据](#)。

示例

下面的示例演示如何对 `CollectionViewSource` 中的 `Task` 数据进行分组、排序和筛选，并在 `DataGrid` 中显示经过分组、排序和筛选的 `Task` 数据。`CollectionViewSource` 用作 `DataGrid` 的 `ItemsSource`。对 `CollectionViewSource` 执行分组、排序和筛选，并在 `DataGrid` UI 中显示。

要测试此示例，需要调整 `DGGroupSortFilterExample` 名称以匹配项目名称。如果使用 Visual Basic，则需要将 `Window` 的类名更改为以下形式。

```
<Window x:Class="MainWindow"
```

XAML

```
<Window x:Class="DGGroupSortFilterExample.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:DGGroupSortFilterExample"
    xmlns:scm="clr-namespace:System.ComponentModel;assembly=WindowsBase"
    Title="Group, Sort, and Filter Example" Height="575" Width="525">
<Window.Resources>
    <local:CompleteConverter x:Key="completeConverter" />
    <local:Tasks x:Key="tasks" />
    <CollectionViewSource x:Key="cvsTasks" Source="{StaticResource
```

```

tasks}"                                     Filter="CollectionViewSource_Filter">
    <CollectionViewSource.SortDescriptions>
        <!-- Requires 'xmlns:scm="clr-
namespace:System.ComponentModel;assembly=WindowsBase"' declaration. -->
        <scm:SortDescription PropertyName="ProjectName"/>
        <scm:SortDescription PropertyName="Complete" />
        <scm:SortDescription PropertyName="DueDate" />
    </CollectionViewSource.SortDescriptions>
    <CollectionViewSource.GroupDescriptions>
        <PropertyGroupDescription PropertyName="ProjectName"/>
        <PropertyGroupDescription PropertyName="Complete"/>
    </CollectionViewSource.GroupDescriptions>
</CollectionViewSource>
</Window.Resources>
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition Height="30" />
    </Grid.RowDefinitions>
    <DataGrid x:Name="dataGrid1"
              ItemsSource="{Binding Source={StaticResource cvsTasks}}"
              CanUserAddRows="False">
        <DataGrid.GroupStyle>
            <!-- Style for groups at top level. -->
            <GroupStyle>
                <GroupStyle.ContainerStyle>
                    <Style TargetType="{x:Type GroupItem}">
                        <Setter Property="Margin" Value="0,0,0,5"/>
                        <Setter Property="Template">
                            <Setter.Value>
                                <ControlTemplate TargetType="{x:Type
GroupItem}">
                                    <Expander IsExpanded="True"
Background="#FF112255" BorderBrush="#FF002255" Foreground="#FFFFFF"
BorderThickness="1,1,1,5">
                                        <Expander.Header>
                                            <DockPanel>
                                                <TextBlock
FontWeight="Bold" Text="{Binding Path=Name}" Margin="5,0,0,0" Width="100"/>
                                                <TextBlock
FontWeight="Bold" Text="{Binding Path=ItemCount}"/>
                                            </DockPanel>
                                        </Expander.Header>
                                        <Expander.Content>
                                            <ItemsPresenter />
                                        </Expander.Content>
                                    </Expander>
                                </ControlTemplate>
                            </Setter.Value>
                        </Style>
                    </GroupStyle.ContainerStyle>
                </GroupStyle>
            <!-- Style for groups under the top level. -->

```

```

<GroupStyle>
    <GroupStyle.HeaderTemplate>
        <DataTemplate>
            <DockPanel Background="LightBlue">
                <TextBlock Text="{Binding Path=Name,
Converter={StaticResource completeConverter}}" Foreground="Blue"
Margin="30,0,0,0" Width="100"/>
                <TextBlock Text="{Binding Path=ItemCount}" Foreground="Blue"/>
            </DockPanel>
        </DataTemplate>
    </GroupStyle.HeaderTemplate>
</GroupStyle>
</DataGrid.GroupStyle>
<DataGrid.RowStyle>
    <Style TargetType="DataGridRow">
        <Setter Property="Foreground" Value="Black" />
        <Setter Property="Background" Value="White" />
    </Style>
</DataGrid.RowStyle>
</DataGrid>
<StackPanel Orientation="Horizontal" Grid.Row="1">
    <TextBlock Text=" Filter completed items "
VerticalAlignment="Center" />
    <CheckBox x:Name="cbCompleteFilter" VerticalAlignment="Center"
Checked="CompleteFilter_Changed"
Unchecked="CompleteFilter_Changed" />
    <Button Content="Remove Groups" Margin="10,2,2,2"
Click="UngroupButton_Click" />
    <Button Content="Group by Project/Status" Margin="2"
Click="GroupButton_Click" />
</StackPanel>
</Grid>
</Window>

```

C#

```

using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Windows;
using System.Windows.Data;

namespace DGGroupSortFilterExample
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();

```

```

// Get a reference to the tasks collection.
Tasks _tasks = (Tasks)this.Resources["tasks"];

// Generate some task data and add it to the task list.
for (int i = 1; i <= 14; i++)
{
    _tasks.Add(new Task()
    {
        ProjectName = "Project " + ((i % 3) + 1).ToString(),
        TaskName = "Task " + i.ToString(),
        DueDate = DateTime.Now.AddDays(i),
        Complete = (i % 2 == 0)
    });
}

private void UngroupButton_Click(object sender, RoutedEventArgs e)
{
    ICollectionView cvTasks =
CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource);
    if (cvTasks != null)
    {
        cvTasks.GroupDescriptions.Clear();
    }
}

private void GroupButton_Click(object sender, RoutedEventArgs e)
{
    ICollectionView cvTasks =
CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource);
    if (cvTasks != null && cvTasks.CanGroup == true)
    {
        cvTasks.GroupDescriptions.Clear();
        cvTasks.GroupDescriptions.Add(new
PropertyGroupDescription("ProjectName"));
        cvTasks.GroupDescriptions.Add(new
PropertyGroupDescription("Complete"));
    }
}

private void CompleteFilter_Changed(object sender, RoutedEventArgs e)
{
    // Refresh the view to apply filters.

CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource).Refresh();
}

private void CollectionViewSource_Filter(object sender,
FilterEventArgs e)
{
    Task t = e.Item as Task;
    if (t != null)
        // If filter is turned on, filter completed items.
}

```

```

        {
            if (this.cbCompleteFilter.IsChecked == true && t.Complete ==
true)
                e.Accepted = false;
            else
                e.Accepted = true;
        }
    }
}

[ValueConversion(typeof(Boolean), typeof(String))]
public class CompleteConverter : IValueConverter
{
    // This converter changes the value of a Tasks Complete status from
    // true/false to a string value of
    // "Complete"/"Active" for use in the row group header.
    public object Convert(object value, Type targetType, object
parameter, System.Globalization.CultureInfo culture)
    {
        bool complete = (bool)value;
        if (complete)
            return "Complete";
        else
            return "Active";
    }

    public object ConvertBack(object value, Type targetType, object
parameter, System.Globalization.CultureInfo culture)
    {
        string strComplete = (string)value;
        if (strComplete == "Complete")
            return true;
        else
            return false;
    }
}

// Task Class
// Requires using System.ComponentModel;
public class Task : INotifyPropertyChanged, IEditableObject
{
    // The Task class implements INotifyPropertyChanged and
    // IEditableObject
    // so that the datagrid can properly respond to changes to the
    // data collection and edits made in the DataGrid.

    // Private task data.
    private string m_ProjectName = string.Empty;
    private string m_TaskName = string.Empty;
    private DateTime m_DueDate = DateTime.Now;
    private bool m_Complete = false;

    // Data for undoing canceled edits.
    private Task temp_Task = null;
    private bool m_Editing = false;
}

```

```
// Public properties.
public string ProjectName
{
    get { return this.m_ProjectName; }
    set
    {
        if (value != this.m_ProjectName)
        {
            this.m_ProjectName = value;
            NotifyPropertyChanged("ProjectName");
        }
    }
}

public string TaskName
{
    get { return this.m_TaskName; }
    set
    {
        if (value != this.m_TaskName)
        {
            this.m_TaskName = value;
            NotifyPropertyChanged("TaskName");
        }
    }
}

public DateTime DueDate
{
    get { return this.m_DueDate; }
    set
    {
        if (value != this.m_DueDate)
        {
            this.m_DueDate = value;
            NotifyPropertyChanged("DueDate");
        }
    }
}

public bool Complete
{
    get { return this.m_Complete; }
    set
    {
        if (value != this.m_Complete)
        {
            this.m_Complete = value;
            NotifyPropertyChanged("Complete");
        }
    }
}

// Implement INotifyPropertyChanged interface.
```

```
public event PropertyChangedEventHandler PropertyChanged;

private void NotifyPropertyChanged(string propertyName)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new
PropertyChangedEventArgs(propertyName));
    }
}

// Implement IEditableObject interface.
public void BeginEdit()
{
    if (m_Editing == false)
    {
        temp_Task = this.MemberwiseClone() as Task;
        m_Editing = true;
    }
}

public void CancelEdit()
{
    if (m_Editing == true)
    {
        this.ProjectName = temp_Task.ProjectName;
        this.TaskName = temp_Task.TaskName;
        this.DueDate = temp_Task.DueDate;
        this.Complete = temp_Task.Complete;
        m_Editing = false;
    }
}

public void EndEdit()
{
    if (m_Editing == true)
    {
        temp_Task = null;
        m_Editing = false;
    }
}

// Requires using System.Collections.ObjectModel;
public class Tasks : ObservableCollection<Task>
{
    // Creating the Tasks collection in this way enables data binding
from XAML.
}
```

另请参阅

- 数据绑定概述
- 创建和绑定到 ObservableCollection
- 在视图中筛选数据
- 在视图中对数据进行排序
- 在 XAML 中使用视图对数据进行排序和分组

如何：使用 DataGrid 控件实现验证

项目 • 2023/02/06

借助 [DataGrid](#) 控件，可以在单元格级别和行级别执行验证。通过单元格级别验证，可以在用户更新值时验证绑定数据对象的单个属性。通过行级别验证，可以在用户提交对行的更改时验证整个数据对象。还可以提供针对验证错误的自定义可视化反馈，或使用 [DataGrid](#) 控件提供的默认可视化反馈。

以下过程介绍如何将验证规则应用于 [DataGrid](#) 绑定并自定义可视化反馈。

验证各单元格值

- 针对与列一起使用的绑定指定一个或多个验证规则。这类似于在简单控件中验证数据，如[数据绑定概述](#)中所述。

以下示例显示了一个 [DataGrid](#) 控件，其中四列绑定到业务对象的不同属性。其中三列通过将 [ValidatesOnExceptions](#) 属性设置为 `true` 来指定 [ExceptionValidationRule](#)。

```
XAML

<Grid>

    <Grid.Resources>
        <local:Courses x:Key="courses"/>
    </Grid.Resources>

    <DataGrid Name="dataGrid1" FontSize="20"
        ItemsSource="{StaticResource courses}"
        AutoGenerateColumns="False">
        <DataGrid.Columns>
            <DataGridTextColumn Header="Course Name"
                Binding="{Binding Name, TargetNullValue=(enter a course
name)}"/>
            <DataGridTextColumn Header="Course ID"
                Binding="{Binding Id, ValidatesOnExceptions=True}"/>
            <DataGridTextColumn Header="Start Date"
                Binding="{Binding StartDate, ValidatesOnExceptions=True,
StringFormat=d}"/>
            <DataGridTextColumn Header="End Date"
                Binding="{Binding EndDate, ValidatesOnExceptions=True,
StringFormat=d}"/>
        </DataGrid.Columns>
    </DataGrid>

</Grid>
```

当用户输入无效值（例如在课程 ID 列中输入非整数）时，单元格周围边框会显示为红色。可以更改此默认验证反馈，如以下过程中所述。

自定义单元格验证反馈

- 将列的 `EditingStyle` 属性设置为适合列的编辑控件的样式。由于编辑控件在运行时创建，因此不能像对简单控件一样使用 `Validation.ErrorTemplate` 附加属性。

以下示例添加了使用验证规则的三列共享的错误样式，从而更新了上一个示例。当用户输入无效值时，样式将更改单元格背景色并添加工具提示。请注意，使用触发器来确定是否存在验证错误。此步骤是必需的，因为当前没有针对单元格的专用错误模板。

XAML

```
<DataGrid.Resources>
    <Style x:Key="errorStyle" TargetType="{x:Type TextBox}">
        <Setter Property="Padding" Value="-2"/>
        <Style.Triggers>
            <Trigger Property="Validation.HasError" Value="True">
                <Setter Property="Background" Value="Red"/>
                <Setter Property="ToolTip"
                    Value="{Binding RelativeSource={RelativeSource Self},
                        Path=(Validation.Errors)[0].ErrorContent}"/>
            </Trigger>
        </Style.Triggers>
    </Style>
</DataGrid.Resources>

<DataGrid.Columns>
    <DataGridTextColumn Header="Course Name"
        Binding="{Binding Name, TargetNullValue=(enter a course name)}"/>
    <DataGridTextColumn Header="Course ID"
        EditingElementStyle="{StaticResource errorStyle}"
        Binding="{Binding Id, ValidatesOnExceptions=True}"/>
    <DataGridTextColumn Header="Start Date"
        EditingElementStyle="{StaticResource errorStyle}"
        Binding="{Binding StartDate, ValidatesOnExceptions=True,
            StringFormat=d}"/>
    <DataGridTextColumn Header="End Date"
        EditingElementStyle="{StaticResource errorStyle}"
        Binding="{Binding EndDate, ValidatesOnExceptions=True,
            StringFormat=d}"/>
</DataGrid.Columns>
```

可以通过替换列使用的 `CellStyle` 来实现更广泛的自定义。

验证单个行中的多个值

- 实现一个检查绑定数据对象的多个属性的 `ValidationRule` 子类。在 `Validate` 方法实现中，将 `value` 参数值强制转换为 `BindingGroup` 实例。然后，可以通过 `Items` 属性访问数据对象。

以下示例演示了验证 `course` 对象的 `StartDate` 属性值是否早于其 `EndDate` 属性值的过程。

C#

```
public class CourseValidationRule : ValidationRule
{
    public override ValidationResult Validate(object value,
        System.Globalization.CultureInfo cultureInfo)
    {
        Course course = (value as BindingGroup).Items[0] as Course;
        if (course.StartDate > course.EndDate)
        {
            return new ValidationResult(false,
                "Start Date must be earlier than End Date.");
        }
        else
        {
            return ValidationResult.ValidResult;
        }
    }
}
```

- 将验证规则添加到 `DataGrid.RowValidationRules` 集合。可以通过 `RowValidationRules` 属性直接访问 `BindingGroup` 实例的 `ValidationRules` 属性，该实例对控件使用的所有绑定进行分组。

以下示例在 XAML 中设置 `RowValidationRules` 属性。`ValidationStep` 属性设置为 `UpdatedValue`，以便仅在更新绑定数据对象后进行验证。

XAML

```
<DataGrid.RowValidationRules>
    <local:CourseValidationRule ValidationStep="UpdatedValue"/>
</DataGrid.RowValidationRules>
```

若用户指定的结束日期早于开始日期，行标题中将显示一个红色感叹号 (!)。可以更改此默认验证反馈，如以下过程中所述。

自定义行验证反馈

- 设置 `DataGrid.RowValidationErrorTemplate` 属性。使用此属性可以自定义各个 `DataGrid` 控件的行验证反馈。还可以使用隐式行样式设置

`DataGridRow.ValidationErrorTemplate` 属性来影响多个控件。

以下示例将默认行验证反馈替换为更明显的标记。若用户输入无效值，行标题中将显示带有白色感叹号的红色圆圈。行和单元格验证错误时都将发生这种情况。关联的错误消息将显示在工具提示中。

XAML

```
<DataGrid.RowValidationErrorTemplate>
<ControlTemplate>
    <Grid Margin="0,-2,0,-2"
        ToolTip="{Binding RelativeSource={RelativeSource
            FindAncestor, AncestorType={x:Type DataGridRow}},
            Path=(Validation.Errors)[0].ErrorContent}">
        <Ellipse StrokeThickness="0" Fill="Red"
            Width="{TemplateBinding FontSize}"
            Height="{TemplateBinding FontSize}" />
        <TextBlock Text="!" FontSize="{TemplateBinding FontSize}"
            FontWeight="Bold" Foreground="White"
            HorizontalAlignment="Center" />
    </Grid>
</ControlTemplate>
</DataGrid.RowValidationErrorTemplate>
```

示例

以下示例完整演示了单元格验证和行验证。`Course` 类提供了示例数据对象，以实现 `IEditableObject` 来支持事务。`DataGrid` 控件可与 `IEditableObject` 交互，使用户可以通过按 `ESC` 来还原更改。

① 备注

如果使用的是 Visual Basic，则在 `MainWindow.xaml` 的第一行中，将 `x:Class="DataGridValidation.MainWindow"` 替换为 `x:Class="MainWindow"`。

若要测试验证，请尝试以下操作：

- 在“课程 ID”列中输入一个非整数值。
- 在“结束日期”列中输入一个早于开始日期的日期。
- 删除“课程 ID”、“开始日期”或“结束日期”中的值。
- 若要撤消无效的单元格值，请将光标放回到单元格中，然后按 `ESC` 键。
- 若要在当前单元格处于编辑模式时撤消整个行的更改，请按 `ESC` 键两次。

- 发生验证错误时，将鼠标指针移到行标题中的标记上，以查看关联的错误消息。

C#

```

using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;

namespace DataGridValidation
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            dataGrid1.InitializingNewItem += (sender, e) =>
            {
                Course newCourse = e.NewItem as Course;
                newCourse.StartDate = newCourse.EndDate = DateTime.Today;
            };
        }
    }

    public class Courses : ObservableCollection<Course>
    {
        public Courses()
        {
            this.Add(new Course
            {
                Name = "Learning WPF",
                Id = 1001,
                StartDate = new DateTime(2010, 1, 11),
                EndDate = new DateTime(2010, 1, 22)
            });
            this.Add(new Course
            {
                Name = "Learning Silverlight",
                Id = 1002,
                StartDate = new DateTime(2010, 1, 25),
                EndDate = new DateTime(2010, 2, 5)
            });
            this.Add(new Course
            {
                Name = "Learning Expression Blend",
                Id = 1003,
                StartDate = new DateTime(2010, 2, 8),
                EndDate = new DateTime(2010, 2, 19)
            });
            this.Add(new Course
            {
                Name = "Learning LINQ",
            });
        }
    }
}

```

```
        Id = 1004,
        StartDate = new DateTime(2010, 2, 22),
        EndDate = new DateTime(2010, 3, 5)
    );
}
}

public class Course : IEditableObject, INotifyPropertyChanged
{
    private string _name;
    public string Name
    {
        get
        {
            return _name;
        }
        set
        {
            if (_name == value) return;
            _name = value;
            OnPropertyChanged("Name");
        }
    }

    private int _number;
    public int Id
    {
        get
        {
            return _number;
        }
        set
        {
            if (_number == value) return;
            _number = value;
            OnPropertyChanged("Id");
        }
    }

    private DateTime _startDate;
    public DateTime StartDate
    {
        get
        {
            return _startDate;
        }
        set
        {
            if (_startDate == value) return;
            _startDate = value;
            OnPropertyChanged("StartDate");
        }
    }

    private DateTime _endDate;
```

```

public DateTime EndDate
{
    get
    {
        return _endDate;
    }
    set
    {
        if (_endDate == value) return;
        _endDate = value;
        OnPropertyChanged("EndDate");
    }
}

#region IEditableObject

private Course backupCopy;
private bool inEdit;

public void BeginEdit()
{
    if (inEdit) return;
    inEdit = true;
    backupCopy = this.MemberwiseClone() as Course;
}

public void CancelEdit()
{
    if (!inEdit) return;
    inEdit = false;
    this.Name = backupCopy.Name;
    this.Id = backupCopy.Id;
    this.StartDate = backupCopy.StartDate;
    this.EndDate = backupCopy.EndDate;
}

public void EndEdit()
{
    if (!inEdit) return;
    inEdit = false;
    backupCopy = null;
}

#endregion

#region INotifyPropertyChanged

public event PropertyChangedEventHandler PropertyChanged;

private void OnPropertyChanged(string propertyName)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this,
            new PropertyChangedEventArgs(propertyName));
    }
}

```

```

        }

    }

    #endregion

}

public class CourseValidationRule : ValidationRule
{
    public override ValidationResult Validate(object value,
        System.Globalization.CultureInfo cultureInfo)
    {
        Course course = (value as BindingGroup).Items[0] as Course;
        if (course.StartDate > course.EndDate)
        {
            return new ValidationResult(false,
                "Start Date must be earlier than End Date.");
        }
        else
        {
            return ValidationResult.ValidResult;
        }
    }
}
}

```

XAML

```

<Window x:Class="DataGridValidation.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:DataGridValidation"
    Title="DataGrid Validation Example" Height="240" Width="600">

    <Grid>

        <Grid.Resources>
            <local:Courses x:Key="courses"/>
        </Grid.Resources>

        <DataGrid Name="dataGrid1" FontSize="20" RowHeaderWidth="27"
            ItemsSource="{StaticResource courses}"
            AutoGenerateColumns="False">

            <DataGrid.Resources>
                <Style x:Key="errorStyle" TargetType="{x:Type TextBox}">
                    <Setter Property="Padding" Value="-2"/>
                    <Style.Triggers>
                        <Trigger Property="Validation.HasError" Value="True">
                            <Setter Property="Background" Value="Red"/>
                            <Setter Property="ToolTip"
                                Value="{Binding RelativeSource={RelativeSource Self},
                                Path=(Validation.Errors)[0].ErrorContent}"/>
                        

```

```

        </Trigger>
    </Style.Triggers>
</Style>
</DataGrid.Resources>

<DataGrid.Columns>
    <DataGridTextColumn Header="Course Name"
        Binding="{Binding Name, TargetNullValue=(enter a course name)}"/>
    <DataGridTextColumn Header="Course ID"
        EditingElementStyle="{StaticResource errorStyle}"
        Binding="{Binding Id, ValidatesOnExceptions=True}"/>
    <DataGridTextColumn Header="Start Date"
        EditingElementStyle="{StaticResource errorStyle}"
        Binding="{Binding StartDate, ValidatesOnExceptions=True,
            StringFormat=d}"/>
    <DataGridTextColumn Header="End Date"
        EditingElementStyle="{StaticResource errorStyle}"
        Binding="{Binding EndDate, ValidatesOnExceptions=True,
            StringFormat=d}"/>
</DataGrid.Columns>

<DataGrid.RowValidationRules>
    <local:CourseValidationRule ValidationStep="UpdatedValue"/>
</DataGrid.RowValidationRules>

<DataGrid.RowValidationErrorTemplate>
    <ControlTemplate>
        <Grid Margin="0,-2,0,-2"
            ToolTip="{Binding RelativeSource={RelativeSource
                FindAncestor, AncestorType={x:Type DataGridRow}},
            Path=(Validation.Errors)[0].ErrorContent}">
            <Ellipse StrokeThickness="0" Fill="Red"
                Width="{TemplateBinding FontSize}"
                Height="{TemplateBinding FontSize}" />
            <TextBlock Text="!" FontSize="{TemplateBinding FontSize}"
                FontWeight="Bold" Foreground="White"
                HorizontalAlignment="Center" />
        </Grid>
    </ControlTemplate>
</DataGrid.RowValidationErrorTemplate>

</DataGrid>

</Grid>
</Window>

```

另请参阅

- [DataGrid](#)
- [DataGrid](#)
- [数据绑定](#)

- 实现绑定验证
- 在自定义对象上实现验证逻辑

演练：在 DataGrid 控件中显示 SQL Server 数据库中的数据

项目 • 2022/09/27

演示如何从 SQL Server 数据库检索数据并在 [DataGrid](#) 控件中显示该数据。可以使用 ADO.NET Entity Framework 创建表示数据的实体类，并使用 LINQ 编写查询，以便从实体类检索指定数据。

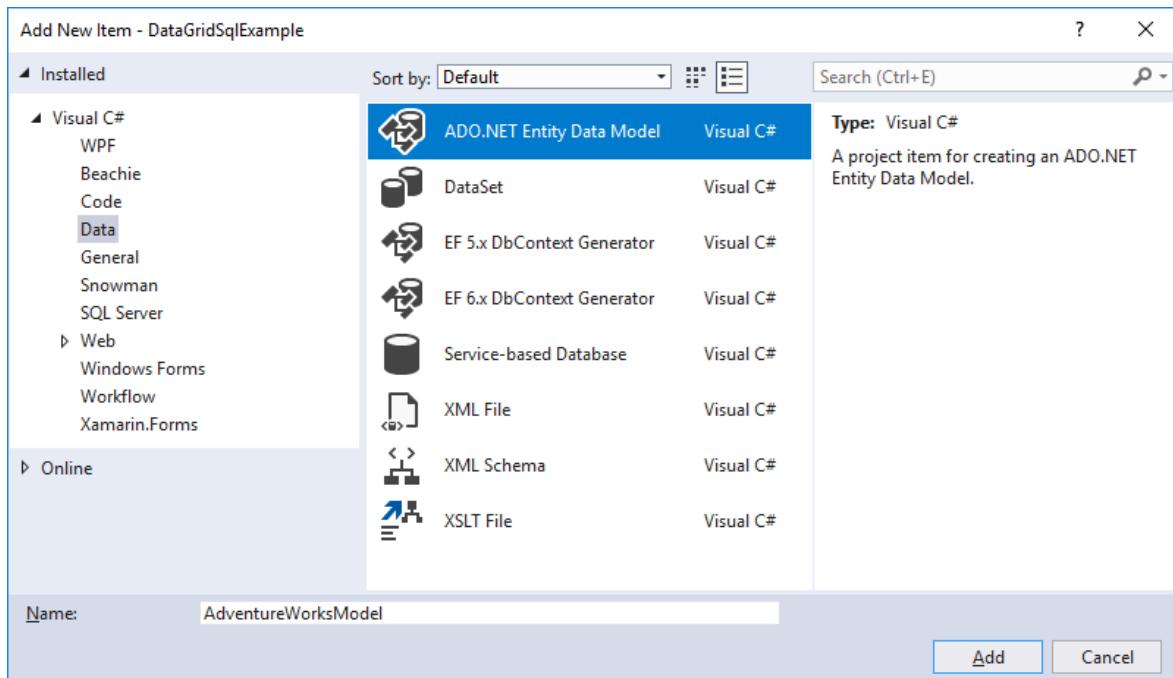
先决条件

您需要满足以下条件才能完成本演练：

- Visual Studio。
- 可访问附加了 AdventureWorksLT 示例数据库的 SQL Server 或 SQL Server Express 的正在运行的实例。可以从 [GitHub](#) 下载 AdventureWorks 数据库。

创建实体类

1. 在 Visual Basic 或 C# 中创建一个新的 WPF 应用程序项目，并将其命名为 `DataGridSQLEExample`。
2. 在解决方案资源管理器中，右键单击项目，指向“添加”，然后选择“新建项目”。
将显示“添加新项”对话框。
3. 在“已安装的模板”窗格中，选择“数据”，然后在模板列表中，选择“ADO.NET 实体数据模型”。



4. 将文件命名为 `AdventureWorksModel.edmx`，然后单击“添加”。

此时将显示实体数据模型向导。

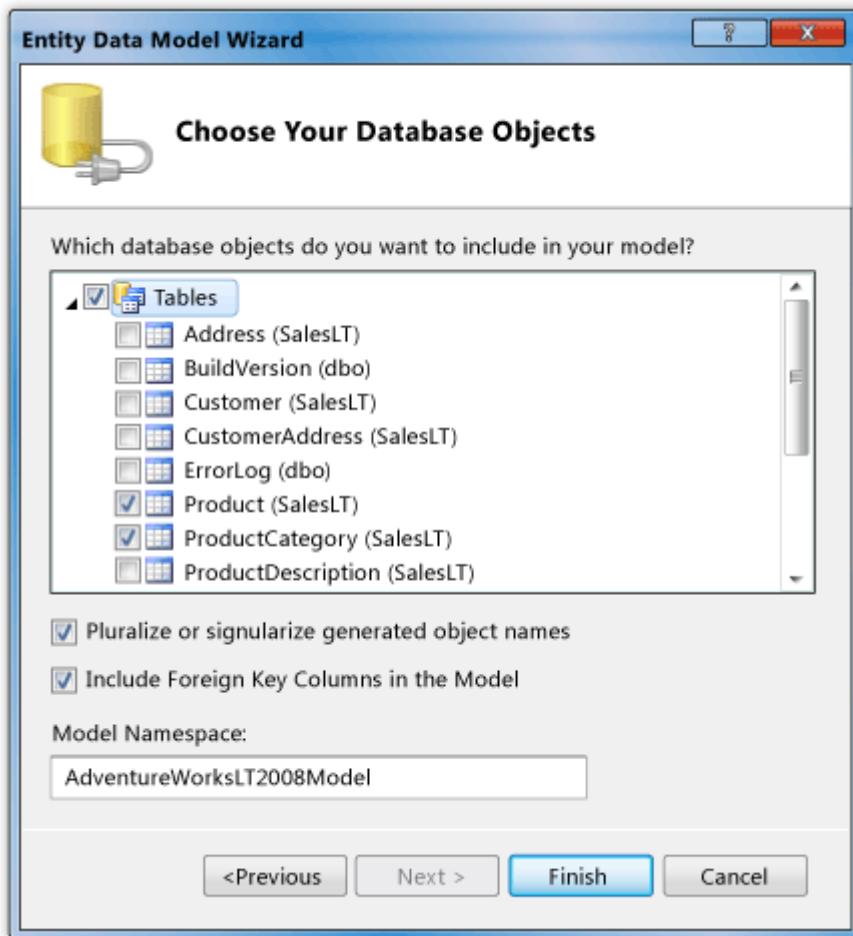
5. 在“选择模型内容”屏幕中，选择“数据库中的 EF 设计器”，然后单击“下一步”。

6. 在“选择数据连接”屏幕中，提供与 AdventureWorksLT2008 数据库的连接。有关详细信息，请参阅[选择“数据连接”对话框](#)。

确保名称为 `AdventureWorksLT2008Entities` 并且选中了“将 App.Config 中的实体连接设置另存为”复选框，然后单击“下一步”。

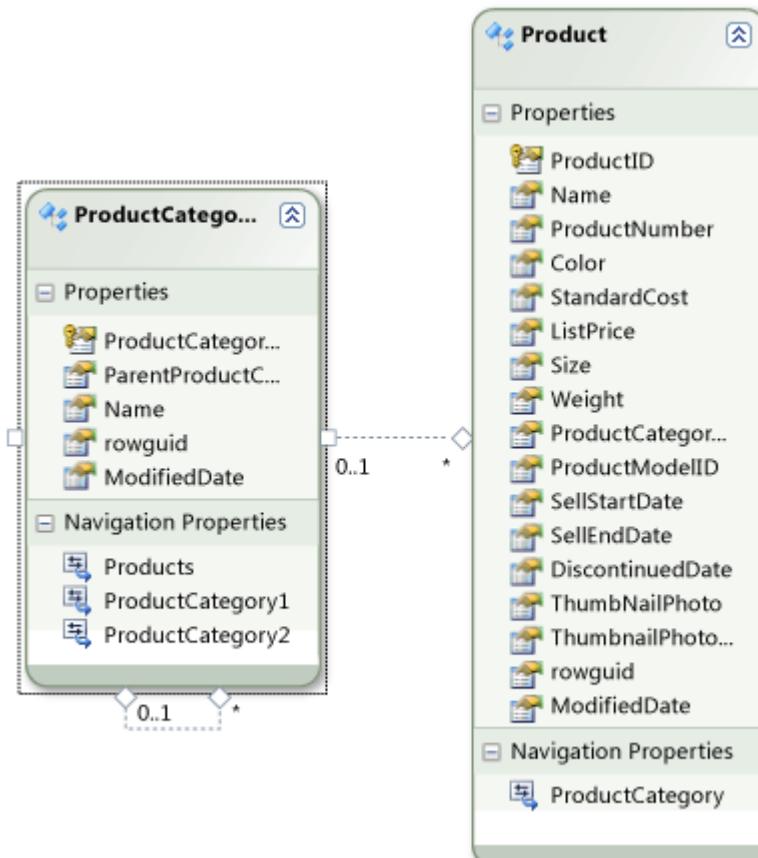
7. 在“选择数据库对象”屏幕中，展开“表”节点，然后选择“Product”和“ProductCategory”表。

可以为所有表生成实体类；但是，在此示例中，仅从这两个表检索数据。



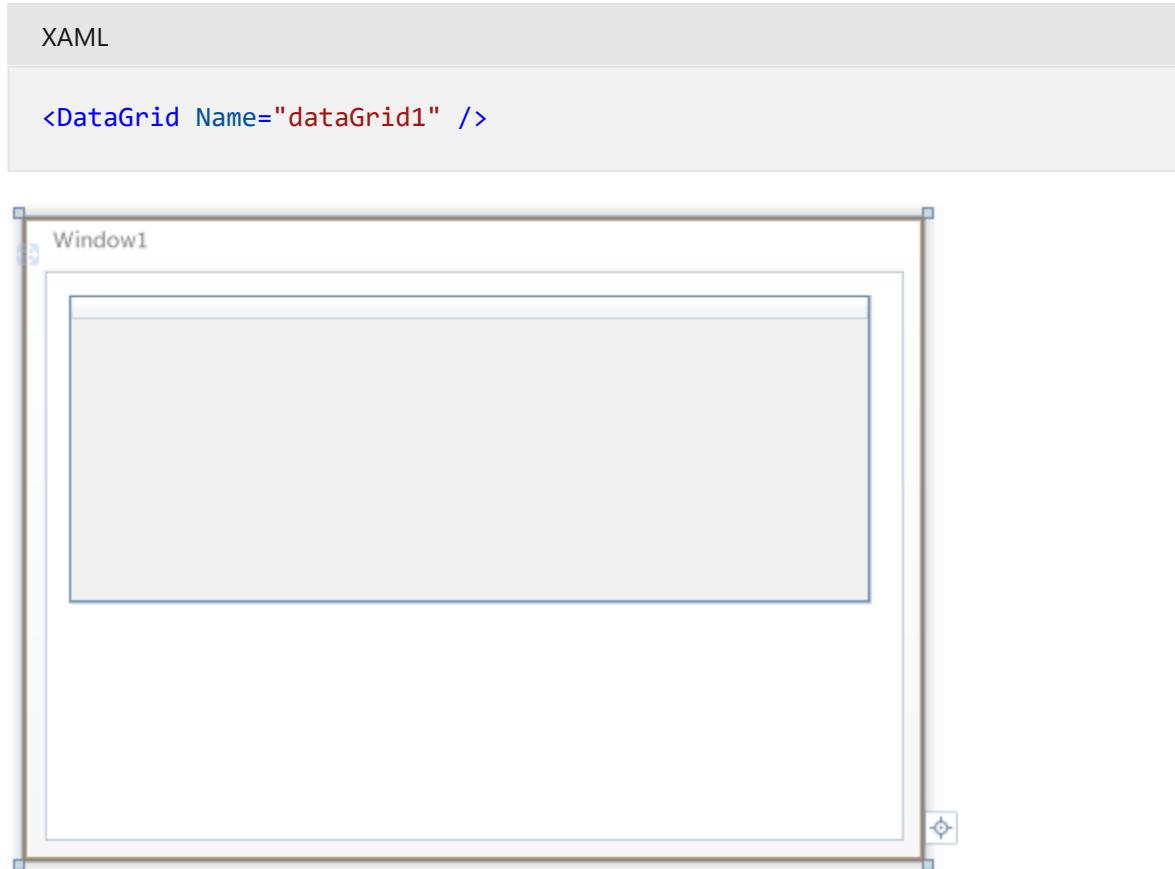
8. 单击“完成”。

Product 和 ProductCategory 实体显示在实体设计器中。



检索和呈现数据

1. 打开 MainWindow.xaml 文件。
2. 将 `Window` 上的 `Width` 属性设置为 450。
3. 在 XAML 编辑器中，在 `<Grid>` 和 `</Grid>` 标记之间添加以下 `DataGrid` 标记以添加名为 `dataGrid1` 的 `DataGrid`。



4. 选择 `Window`。
5. 使用“属性”窗口或 XAML 编辑器，为 `Loaded` 事件创建名为 `Window_Loaded` 的 `Window` 事件处理程序。有关详细信息，请参阅[如何：创建简单事件处理程序](#)。

以下内容显示了 MainWindow.xaml 的 XAML。

① 备注

如果使用的是 Visual Basic，则在 MainWindow.xaml 的第一行中，将 `x:Class="DataGridSQLExample.MainWindow"` 替换为 `x:Class="MainWindow"`。

XAML

```
<Window x:Class="DataGridSQLExample.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="450"
    Loaded="Window_Loaded">
    <Grid>
        <DataGrid Name="dataGrid1" />
    </Grid>
</Window>
```

6. 打开 [Window](#) 的代码隐藏文件 (`MainWindow.xaml.vb` 或 `MainWindow.xaml.cs`) 。
7. 添加以下代码以仅从联接表中检索特定值，并将 [DataGrid](#) 的 `ItemsSource` 属性设置为查询结果。

C#

```
using System.Data.Entity.Core.Objects;
using System.Linq;
using System.Windows;

namespace DataGridSQLExample
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        AdventureWorksLT2008Entities dataEntities = new
AdventureWorksLT2008Entities();

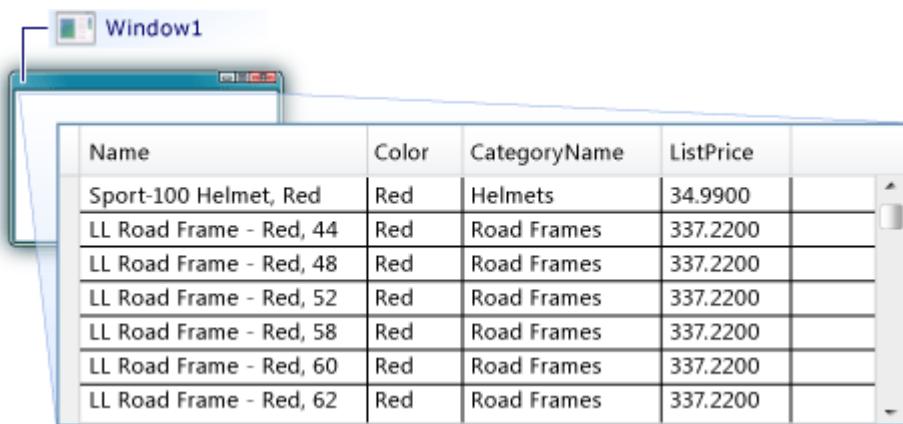
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            var query =
from product in dataEntities.Products
where product.Color == "Red"
orderby product.ListPrice
select new { product.Name, product.Color, CategoryName =
product.ProductCategory.Name, product.ListPrice };

            dataGrid1.ItemsSource = query.ToList();
        }
    }
}
```

8. 运行示例。

应该会看到一个显示数据的 [DataGrid](#)。



另请参阅

- [DataGrid](#)

DataGrid 控件中的调整大小选项

项目 • 2023/02/06

可以使用各种选项来控制 DataGrid 自己调整大小的方式。可以将 DataGrid 以及 DataGrid 中的各个行和列设置为根据其内容自动调整大小，也可以设置为特定值。默认情况下，DataGrid 会根据其内容的大小增大和收缩。

调整 DataGrid 大小

使用自动调整大小时的注意事项

默认情况下，DataGrid 的 Height 和 Width 属性设置为 Double.NaN（在 XAML 中为 “Auto”），DataGrid 将根据其内容的大小进行调整。

当放置在不限制其子项大小的容器（例如 Canvas 或 StackPanel）内时，DataGrid 将超出容器的可见边界，并且不会显示滚动条。此状况可能会影响可用性和性能。

绑定到数据集时，如果 DataGrid 的 Height 不受限制，它将继续为绑定数据集中的每个数据项添加一行。这可能会导致在添加行时 DataGrid 超出应用程序的可见边界。在这种情况下，DataGrid 将不会显示滚动条，因为它的 Height 将继续延伸以适应新行。

针对 DataGrid 中的每一行，会创建一个对象。如果你使用的是大型数据集并且允许 DataGrid 自动调整自身大小，创建大量对象则可能会影响应用程序的性能。

为避免在处理大型数据集时出现这些问题，建议专门设置 DataGrid 的 Height 或将它放置在将限制其 Height 的容器中，例如 Grid。当 Height 受到限制时，DataGrid 将仅创建适合其指定 Height 的行，并且将根据需要回收这些行以显示新数据。

设置 DataGrid 大小

可以将 DataGrid 设置为在指定边界内自动调整大小，也可以将 DataGrid 设置为特定大小。下表显示了用于控制 DataGrid 大小可以设置的属性。

属性	描述
Height	设置 DataGrid 的特定高度。
MaxHeight	设置 DataGrid 高度的上限。 DataGrid 将垂直延伸，直到达到此高度。
MinHeight	设置 DataGrid 高度的下限。 DataGrid 将垂直收缩，直到达到此高度。
Width	设置 DataGrid 的特定宽度。

属性	描述
MaxWidth	设置 DataGrid 宽度的上限。 DataGrid 将水平延伸，直到达到此宽度。
MinWidth	设置 DataGrid 宽度的下限。 DataGrid 将水平收缩，直到达到此宽度。

调整行和行标题的大小

DataGrid 行

默认情况下，DataGrid 行的 Height 属性设置为 Double.NaN（在 XAML 中为“Auto”），并且行高将根据其内容大小扩展。可以通过设置 DataGrid.RowHeight 属性来指定 DataGrid 中所有行的高度。用户可以拖动行标题分隔符来更改行高。

DataGrid 行标题

若要显示行标题，必须将 HeadersVisibility 属性设置为 DataGridHeadersVisibility.Row 或 DataGridHeadersVisibility.All。行标题默认将会显示，并且它们会根据其内容自动调整大小。可以通过设置 DataGrid.RowHeaderWidth 属性来赋予行标题特定宽度。

调整列和列标题的大小

DataGrid 列

DataGrid 使用 DataGridLength 和 DataGridLengthUnitType 结构的值来指定绝对或自动调整大小模式。

下表显示了 DataGridLengthUnitType 结构提供的值。

名称	描述
Auto	默认的自动调整大小模式根据单元格和列标题的内容来调整 DataGrid 列的大小。
SizeToCells	基于单元格的自动调整大小模式根据列中单元格（不包括列标题）的内容来调整 DataGrid 列的大小。
SizeToHeader	基于标题的自动调整大小模式仅根据列标题的内容来调整 DataGrid 列的大小。
Pixel	基于像素的调整大小模式根据所提供的数值来调整 DataGrid 列的大小。

名称	描述
Star	<p>比例缩放模式用于按加权比例分配可用空间。</p> <p>在 XAML 中，star 值表示为 n^*，其中 n 表示数值。1^* 等效于 *。例如，如果 DataGrid 中有两列宽度为 * 和 2^*，那么第一个列将获得一部分的可用空间，第二列将获得两部分的可用空间。</p>

[DataGridLengthConverter](#) 类可用于在数值或字符串值与 [DataGridLength](#) 值之间转换数据。

默认情况下，[DataGrid.ColumnWidth](#) 属性设置为 [SizeToHeader](#)，[DataGridColumn.Width](#) 属性设置为 [Auto](#)。当调整大小模式设置为 [Auto](#) 或 [SizeToCells](#) 时，列将扩展到其最宽可见内容的宽度。滚动时，如果滚动到查看范围中的内容超过了当前列的大小，那么这些大小调整模式将使列扩大。在此内容滚动到查看范围之外后，此列也不会收缩。

还可以将 [DataGrid](#) 中的列设置为仅在指定边界内自动调整大小，或者可以将列设置为特定大小。下表显示了用于控制列的大小可以设置的属性。

属性	描述
DataGrid.MaxColumnWidth	设置 DataGrid 中所有列的上界限。
DataGridColumn.MaxWidth	设置单个列的上界限。重写 DataGrid.MaxColumnWidth 。
DataGrid.MinColumnWidth	设置 DataGrid 中所有列的下界限。
DataGridColumn.MinWidth	设置单个列的下界限。重写 DataGrid.MinColumnWidth 。
DataGrid.ColumnWidth	设置 DataGrid 中所有列的特定宽度。
DataGridColumn.Width	设置单个列的特定宽度。重写 DataGrid.ColumnWidth 。

DataGrid 列标题

默认将显示 [DataGrid](#) 列标题。若要隐藏列标题，必须将 [HeadersVisibility](#) 属性设置为 [DataGridHeadersVisibility.Row](#) 或 [DataGridHeadersVisibility.None](#)。默认情况下，当显示列标题时，它们会根据其内容自动调整大小。可以通过设置 [DataGrid.ColumnHeaderHeight](#) 属性来赋予列标题特定高度。

使用鼠标调整大小

用户可以拖动行或列标题分隔符来调整 [DataGrid](#) 行和列的大小。[DataGrid](#) 还支持通过双击行或列标题分隔符来实现自动调整行和列的大小。若要防止用户调整特定列的大小，

对于各个列，请将 `DataGridColumn.CanUserResize` 属性设置为 `false`。若要防止用户调整所有列的大小，请将 `DataGrid.CanUserResizeColumns` 属性设置为 `false`。若要防止用户调整所有行的大小，请将 `DataGrid.CanUserResizeRows` 属性设置为 `false`。

另请参阅

- [DataGrid](#)
- [DataGridColumn](#)
- [DataGridLength](#)
- [DataGridLengthConverter](#)

DatePicker

项目 • 2023/02/06

[DatePicker](#) 控件允许用户通过将日期键入文本字段或使用下拉 [Calendar](#) 控件来选择日期。

下图显示了一个 [DatePicker](#)。



DatePicker 控件

[DatePicker](#) 控件的许多属性用于管理其内置的 [Calendar](#)，其功能与 [Calendar](#) 中的等效属性相同。具体来说，[DatePicker.IsTodayHighlighted](#)、[DatePicker.FirstDayOfWeek](#)、[DatePicker.BlackoutDates](#)、[DatePicker.DisplayDateStart](#)、[DatePicker.DisplayDateEnd](#)、[DatePicker.DisplayDate](#) 和 [DatePicker.SelectedDate](#) 属性的功能与其 [Calendar](#) 对应项相同。有关详细信息，请参阅 [Calendar](#)。

用户可以直接将日期键入文本字段，这将设置 [Text](#) 属性。如果 [DatePicker](#) 无法将输入的字符串转换为有效日期，则将引发 [DateValidationError](#) 事件。默认情况下，这会导致异常，但 [DateValidationError](#) 的事件处理程序可以将 [ThrowException](#) 属性设置为 `false`，并阻止引发异常。

另请参阅

- [控件](#)
- [样式设置和模板化](#)

DockPanel

项目 • 2023/02/06

[DockPanel](#) 元素用于将子内容放置在布局容器的边缘。

本节内容

[操作指南主题](#)

参考

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

相关章节

[布局](#)

[演练：我的第一个 WPF 桌面应用程序](#)

[ScrollViewer 概述](#)

DockPanel 帮助主题

项目 • 2023/02/06

本节中的主题介绍如何使用 DockPanel 元素在布局容器的边缘定位子元素。

本节内容

[获取或设置 Dock 值](#)

[创建 DockPanel](#)

[使用 DockPanel 元素对空间进行分区](#)

参考

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

相关章节

[布局](#)

[演练：我的第一个 WPF 桌面应用程序](#)

[ScrollViewer 概述](#)

如何：获取或设置 Dock 值

项目 • 2023/02/06

下面的示例显示如何为对象分配 Dock 值。本例使用 DockPanel 的 GetDock 和 SetDock 方法。

示例

本例创建元素 TextBlock 的实例 txt1，并使用 DockPanel 的 SetDock 方法分配 Top 的 Dock 值。然后，它使用 GetDock 方法将 Dock 属性的值附加到 TextBlock 元素的 Text。最后，该示例将 TextBlock 元素添加到父级 DockPanel、dp1。

C#

```
// Create the Panel DockPanel
dp1 = new DockPanel();

// Create a Text Control and then set its Dock property
txt1 = new TextBlock();
DockPanel.SetDock(txt1, System.Windows.Controls.Dock.Top);
txt1.Text = "The Dock Property is set to " + DockPanel.GetDock(txt1);
dp1.Children.Add(txt1);
mainWindow.Content = dp1;
mainWindow.Show();
```

另请参阅

- [DockPanel](#)
- [GetDock](#)
- [SetDock](#)
- [面板概述](#)

如何：创建 DockPanel

项目 • 2023/02/06

示例

以下示例通过代码来创建和使用 `DockPanel` 实例。该示例演示如何通过创建五个 `Rectangle` 元素和在父元素 `DockPanel` 内定位（停靠）它们来对空间分区。如果保留默认设置，最终的矩形将填充所有剩余的未分配空间。

C#

```
private void CreateAndShowMainWindow()
{
    // Create the application's main window
    mainWindow = new Window();

    // Create a DockPanel
    DockPanel myDockPanel = new DockPanel();

    // Add the first rectangle to the DockPanel
    Rectangle rect1 = new Rectangle();
    rect1.Stroke = Brushes.Black;
    rect1.Fill = Brushes.SkyBlue;
    rect1.Height = 25;
    DockPanel.SetDock(rect1, Dock.Top);
    myDockPanel.Children.Add(rect1);

    // Add the second rectangle to the DockPanel
    Rectangle rect2 = new Rectangle();
    rect2.Stroke = Brushes.Black;
    rect2.Fill = Brushes.SkyBlue;
    rect2.Height = 25;
    DockPanel.SetDock(rect2, Dock.Top);
    myDockPanel.Children.Add(rect2);

    // Add the third rectangle to the DockPanel
    Rectangle rect4 = new Rectangle();
    rect4.Stroke = Brushes.Black;
    rect4.Fill = Brushes.Khaki;
    rect4.Height = 25;
    DockPanel.SetDock(rect4, Dock.Bottom);
    myDockPanel.Children.Add(rect4);

    // Add the fourth rectangle to the DockPanel
    Rectangle rect3 = new Rectangle();
    rect3.Stroke = Brushes.Black;
    rect3.Fill = Brushes.PaleGreen;
    rect3.Width = 200;
    DockPanel.SetDock(rect3, Dock.Left);
```

```
myDockPanel.Children.Add(rect3);

// Add the final rectangle to the DockPanel
Rectangle rect5 = new Rectangle();
rect5.Stroke = Brushes.Black;
rect5.Fill = Brushes.White;
myDockPanel.Children.Add(rect5);

// Add the DockPanel to the Window as Content and show the Window
mainWindow.Content = myDockPanel;
mainWindow.Title = "DockPanel Sample";
mainWindow.Show();
}

}
```

另请参阅

- [DockPanel](#)
- [Dock](#)
- [面板概述](#)

如何：使用 DockPanel 元素对空间进行分区

项目 • 2023/02/06

下面的示例使用 [DockPanel](#) 元素创建一个简单的用户界面 (UI) 框架。 [DockPanel](#) 将可用空间分给其子元素。

示例

此示例使用 [Dock](#) 属性（该属性是附加属性）将两个相同的 [Border](#) 元素停靠在已分区空间的 [Top](#) 处。 第三个 [Border](#) 元素停靠至 [Left](#)，其宽度设置为 200 像素。 第四个 [Border](#) 停靠至屏幕的 [Bottom](#)。 最后一个 [Border](#) 元素将自动填充剩余空间。

C#

```
// Create the application's main window
mainWindow = new Window();
mainWindow.Title = "DockPanel Sample";

// Create the DockPanel
DockPanel myDockPanel = new DockPanel();
myDockPanel.LastChildFill = true;

// Define the child content
Border myBorder1 = new Border();
myBorder1.Height = 25;
myBorder1.Background = Brushes.SkyBlue;
myBorder1.BorderBrush = Brushes.Black;
myBorder1.BorderThickness = new Thickness(1);
DockPanel.SetDock(myBorder1, Dock.Top);
TextBlock myTextBlock1 = new TextBlock();
myTextBlock1.Foreground = Brushes.Black;
myTextBlock1.Text = "Dock = Top";
myBorder1.Child = myTextBlock1;

Border myBorder2 = new Border();
myBorder2.Height = 25;
myBorder2.Background = Brushes.SkyBlue;
myBorder2.BorderBrush = Brushes.Black;
myBorder2.BorderThickness = new Thickness(1);
DockPanel.SetDock(myBorder2, Dock.Top);
TextBlock myTextBlock2 = new TextBlock();
myTextBlock2.Foreground = Brushes.Black;
myTextBlock2.Text = "Dock = Top";
myBorder2.Child = myTextBlock2;
```

```

Border myBorder3 = new Border();
myBorder3.Height = 25;
myBorder3.Background = Brushes.LemonChiffon;
myBorder3.BorderBrush = Brushes.Black;
myBorder3.BorderThickness = new Thickness(1);
DockPanel1.SetDock(myBorder3, Dock.Bottom);
TextBlock myTextBlock3 = new TextBlock();
myTextBlock3.Foreground = Brushes.Black;
myTextBlock3.Text = "Dock = Bottom";
myBorder3.Child = myTextBlock3;

Border myBorder4 = new Border();
myBorder4.Width = 200;
myBorder4.Background = Brushes.PaleGreen;
myBorder4.BorderBrush = Brushes.Black;
myBorder4.BorderThickness = new Thickness(1);
DockPanel1.SetDock(myBorder4, Dock.Left);
TextBlock myTextBlock4 = new TextBlock();
myTextBlock4.Foreground = Brushes.Black;
myTextBlock4.Text = "Dock = Left";
myBorder4.Child = myTextBlock4;

Border myBorder5 = new Border();
myBorder5.Background = Brushes.White;
myBorder5.BorderBrush = Brushes.Black;
myBorder5.BorderThickness = new Thickness(1);
TextBlock myTextBlock5 = new TextBlock();
myTextBlock5.Foreground = Brushes.Black;
myTextBlock5.Text = "This content will Fill the remaining space";
myBorder5.Child = myTextBlock5;

// Add child elements to the DockPanel Children collection
myDockPanel.Children.Add(myBorder1);
myDockPanel.Children.Add(myBorder2);
myDockPanel.Children.Add(myBorder3);
myDockPanel.Children.Add(myBorder4);
myDockPanel.Children.Add(myBorder5);

// Add the parent Canvas as the Content of the Window Object
mainWindow.Content = myDockPanel;
mainWindow.Show ();

```

XAML

```

<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
WindowTitle="DockPanel Sample">
<DockPanel LastChildFill="True">
    <Border Height="25" Background="SkyBlue" BorderBrush="Black"
BorderThickness="1" DockPanel.Dock="Top">
        <TextBlock Foreground="Black">Dock = Top</TextBlock>
    </Border>
    <Border Height="25" Background="SkyBlue" BorderBrush="Black"
BorderThickness="1" DockPanel.Dock="Bottom">
        <TextBlock Foreground="Black">Dock = Bottom</TextBlock>
    </Border>
</DockPanel>

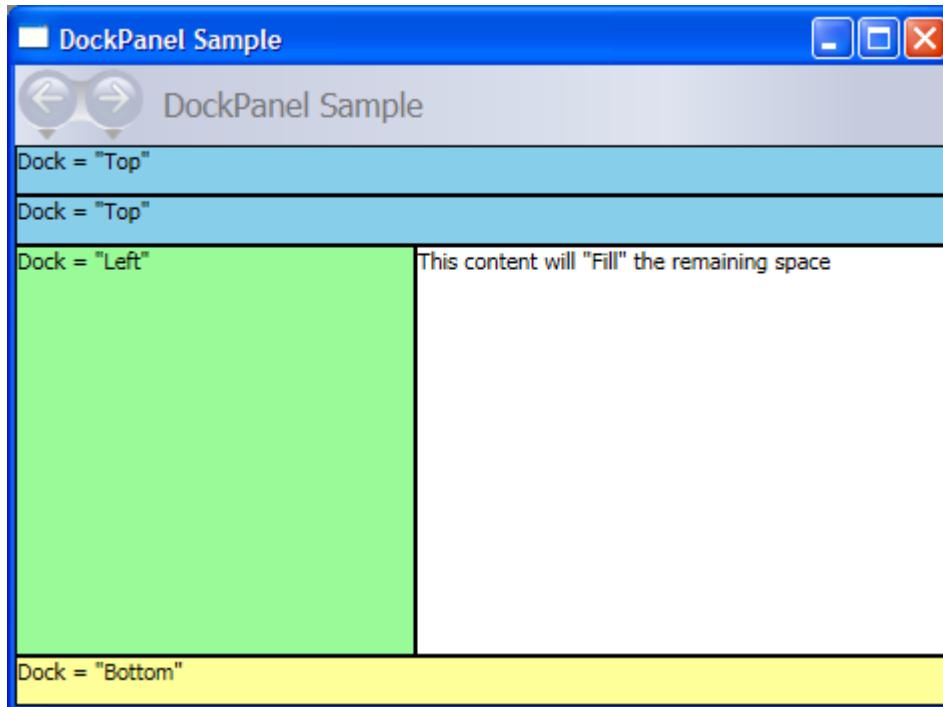
```

```
BorderThickness="1" DockPanel.Dock="Top">
    <TextBlock Foreground="Black">Dock = "Top"</TextBlock>
</Border>
<Border Height="25" Background="LemonChiffon" BorderBrush="Black"
BorderThickness="1" DockPanel.Dock="Bottom">
    <TextBlock Foreground="Black">Dock = "Bottom"</TextBlock>
</Border>
<Border Width="200" Background="PaleGreen" BorderBrush="Black"
BorderThickness="1" DockPanel.Dock="Left">
    <TextBlock Foreground="Black">Dock = "Left"</TextBlock>
</Border>
<Border Background="White" BorderBrush="Black" BorderThickness="1">
    <TextBlock Foreground="Black">This content will "Fill" the remaining
space</TextBlock>
</Border>
</DockPanel>
</Page>
```

① 备注

默认情况下，`DockPanel` 元素的最后一个子元素将填充剩余的未分配空间。如果不希望出现此行为，请设置 `LastChildFill="False"`。

已编译的应用程序将生成如下所示的新 UI。



另请参阅

- [DockPanel](#)
- [面板概述](#)

DocumentViewer

项目 • 2023/02/06

[DocumentViewer](#) 控件用于查看分页格式的 FixedDocument 内容（例如 XML 纸张规范 (XPS) 文档）。

参考

[DocumentViewer](#)

[FixedDocument](#)

请参阅

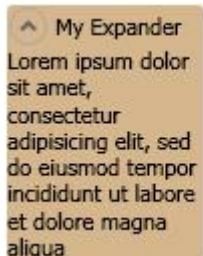
- [文档](#)
- [文档序列化和存储](#)
- [打印概述](#)

Expander

项目 • 2023/02/06

Expander 允许用户查看标题并展开该标题以查看更多详细信息，或将一个部分折叠为标题。

下图提供了此控件处于展开位置的示例。



本节内容

[Expander 概述](#)

[操作指南主题](#)

参考

[Expander](#)

相关章节

Expander 概述

项目 · 2023/02/06

通过 [Expander](#) 控件可以在类似于窗口并包括一个标头的可展开区域中提供内容。

创建简单的 Expander

以下示例演示如何创建一个简单的 [Expander](#) 控件。此示例创建与上图类似的 [Expander](#)。

XAML

```
<Expander Name="myExpander" Background="Tan"
    HorizontalAlignment="Left" Header="My Expander"
    ExpandDirection="Down" IsExpanded="True" Width="100">
    <TextBlock TextWrapping="Wrap">
        Lorem ipsum dolor sit amet, consectetur
        adipisicing elit, sed do eiusmod tempor incididunt ut
        labore et dolore magna aliqua
    </TextBlock>
</Expander>
```

[Expander](#) 的 [Content](#) 和 [Header](#) 还可包含复杂内容，例如 [RadioButton](#) 和 [Image](#) 对象。

设置展开内容区域的方向

通过使用 [ExpandDirection](#) 属性，可将一个 [Expander](#) 控件的内容区域设置为在四个方向的其中一个方向上展开（[Down](#)、[Up](#)、[Left](#) 或 [Right](#)）。当内容区域处于折叠状态时，仅显示 [ExpanderHeader](#) 及其切换按钮。显示方向箭头的 [Button](#) 控件可用作展开或折叠内容区域的切换按钮。展开时，[Expander](#) 将尝试在类似于窗口的区域中显示其所有内容。

在面板中控制 Expander 的大小

如果 [Expander](#) 控件位于继承自 [Panel](#)（例如 [StackPanel](#)）的布局控件内，当 [ExpandDirection](#) 属性设置为 [Down](#) 或 [Up](#) 时，不要在 [Expander](#) 上指定 [Height](#)。同样，当 [ExpandDirection](#) 属性设置为 [Left](#) 或 [Right](#) 时，不要在 [Expander](#) 上指定 [Width](#)。

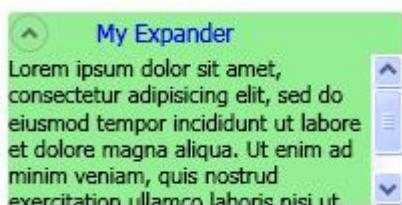
当在 [Expander](#) 控件上以显示展开内容的方向设置大小维度时，[Expander](#) 将控制内容使用的区域，并在内容周围显示一个边框。即使内容已折叠，也会显示该边框。若要设置展开内容区域的大小，请在 [Expander](#) 的内容上设置大小维度，或者如果需要滚动功能，请在包含内容的 [ScrollViewer](#) 上设置大小维度。

当 [Expander](#) 控件为 [DockPanel](#) 中的最后一个元素时，Windows Presentation Foundation (WPF) 会自动将 [Expander](#) 维度设置为与 [DockPanel](#) 的其余区域相等。若要防止此默认行为，将 [DockPanel](#) 对象上的 [LastChildFill](#) 属性设置为 `false`，或确保该 [Expander](#) 不是 [DockPanel](#) 中的最后一个元素。

创建可滚动内容

如果内容对于内容区域的大小而言过大，可在 [ScrollViewer](#) 中换行显示 [Expander](#) 的内容，以提供可滚动内容。[Expander](#) 控件不自动提供滚动功能。下图显示了包含 [ScrollViewer](#) 控件的 [Expander](#) 控件。

ScrollViewer 中的 Expander



将 [Expander](#) 控件置于 [ScrollViewer](#) 中时，将与 [Expander](#) 的内容展开方向对应的 [ScrollViewer](#) 维度属性设置为 [Expander](#) 内容区域的大小。例如，如果将 [Expander](#) 上的 [ExpandDirection](#) 属性设置为 [Down](#)（内容区域向下展开），则将 [ScrollViewer](#) 控件上的 [Height](#) 属性设置为内容区域的必需高度。如果设置的是内容本身的高度维度，[ScrollViewer](#) 不会识别此设置，因此也不会提供可滚动内容。

以下示例演示如何创建一个具有复杂内容并包含 [ScrollViewer](#) 控件的 [Expander](#) 控件。此示例将创建一个类似于本部分开头图示的 [Expander](#)。

C#

```
void MakeExpander()
{
    //Create containing stack panel and assign to Grid row/col
    StackPanel sp = new StackPanel();
    Grid.SetRow(sp, 0);
    Grid.SetColumn(sp, 1);
    sp.Background = Brushes.LightSalmon;

    //Create column title
    TextBlock colTitle = new TextBlock();
    colTitle.Text = "EXPANDER CREATED FROM CODE";
    colTitle.HorizontalAlignment= HorizontalAlignment.Center;
    colTitle.Margin.Bottom.Equals(20);
    sp.Children.Add(colTitle);

    //Create Expander object
}
```

```

Expander exp = new Expander();

//Create Bullet Panel for Expander Header
BulletDecorator bp = new BulletDecorator();
Image i = new Image();
BitmapImage bi= new BitmapImage();
bi.UriSource = new Uri(@"pack://application:,,,/images/icon.jpg");
i.Source = bi;
i.Width = 10;
bp.Bullet = i;
TextBlock tb = new TextBlock();
tb.Text = "My Expander";
tb.Margin = new Thickness(20,0,0,0);
bp.Child = tb;
exp.Header = bp;

//Create TextBlock with ScrollViewer for Expander Content
StackPanel spScroll = new StackPanel();
TextBlock tbc = new TextBlock();
tbc.Text =
    "Lorem ipsum dolor sit amet, consectetur adipisicing elit," +
    "sed do eiusmod tempor incididunt ut labore et dolore magna" +
    "aliqua. Ut enim ad minim veniam, quis nostrud exercitation" +
    "ullamco laboris nisi ut aliquip ex ea commodo consequat." +
    "Duis aute irure dolor in reprehenderit in voluptate velit" +
    "esse cillum dolore eu fugiat nulla pariatur. Excepteur sint" +
    "occaecat cupidatat non proident, sunt in culpa qui officia" +
    "deserunt mollit anim id est laborum.";
tbc.TextWrapping = TextWrapping.Wrap;

spScroll.Children.Add(tbc);
ScrollView scr = new ScrollView();
scr.Content = spScroll;
scr.Height = 50;
exp.Content = scr;

exp.Width=200;
exp.HorizontalContentAlignment= HorizontalAlignment.Stretch;
//Insert Expander into the StackPanel and add it to the
//Grid
sp.Children.Add(exp);
myGrid.Children.Add(sp);
}

```

XAML

```

<Expander Width="200" HorizontalContentAlignment="Stretch">
    <Expander.Header>
        <BulletDecorator>
            <BulletDecorator.Bullet>
                <Image Width="10" Source="images\icon.jpg"/>
            </BulletDecorator.Bullet>

```

```
<TextBlock Margin="20,0,0,0">My Expander</TextBlock>
</BulletDecorator>
</Expander.Header>
<Expander.Content>
<ScrollViewer Height="50">
<TextBlock TextWrapping="Wrap">
    Lorem ipsum dolor sit amet, consectetur adipisicing elit,
    sed do eiusmod tempor incididunt ut labore et dolore magna
    aliqua. Ut enim ad minim veniam, quis nostrud exercitation
    ullamco laboris nisi ut aliquip ex ea commodo consequat.
    Duis aute irure dolor in reprehenderit in voluptate velit
    esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
    occaecat cupidatat non proident, sunt in culpa qui officia
    deserunt mollit anim id est laborum.
</TextBlock>
</ScrollViewer>
</Expander.Content>
</Expander>
```

使用对齐属性

可通过在 [Expander](#) 控件上设置 [HorizontalContentAlignment](#) 和 [VerticalContentAlignment](#) 属性来对齐内容。当设置这些属性时，对齐将同时应用于标头和展开的内容。

另请参阅

- [Expander](#)
- [ExpandDirection](#)
- [操作指南主题](#)

Expander 帮助主题

项目 • 2023/02/06

本部分中的主题介绍如何使用 [Expander](#) 控件。

本节内容

[使用 ScrollViewer 创建扩展器](#)

参考

[Expander](#)

相关章节

[Expander 概述](#)

如何：使用 ScrollViewer 创建 Expander

项目 • 2023/02/06

此示例演示如何创建包含复杂内容（例如图像和文本）的 [Expander](#) 控件。该示例还将 [Expander](#) 的内容包含在 [ScrollViewer](#) 控件中。

示例

以下示例演示如何创建 [Expander](#)。该示例使用包含图像和文本的 [BulletDecorator](#) 控件来定义 [Header](#)。[ScrollViewer](#) 控件提供一种滚动展开内容的方法。

请注意，该示例在 [ScrollViewer](#) 上而不是在内容上设置 [Height](#) 属性。如果在内容上设置了 [Height](#)，则 [ScrollViewer](#) 不允许用户滚动内容。[Width](#) 属性在 [Expander](#) 控件上设置，此设置适用于 [Header](#) 和展开的内容。

XAML

```
<Expander Width="200" HorizontalContentAlignment="Stretch">
    <Expander.Header>
        <BulletDecorator>
            <BulletDecorator.Bullet>
                <Image Width="10" Source="images\icon.jpg"/>
            </BulletDecorator.Bullet>
            <TextBlock Margin="20,0,0,0">My Expander</TextBlock>
        </BulletDecorator>
    </Expander.Header>
    <Expander.Content>
        <ScrollViewer Height="50">
            <TextBlock TextWrapping="Wrap">
                Lorem ipsum dolor sit amet, consectetur adipisicing elit,
                sed do eiusmod tempor incididunt ut labore et dolore magna
                aliqua. Ut enim ad minim veniam, quis nostrud exercitation
                ullamco laboris nisi ut aliquip ex ea commodo consequat.
                Duis aute irure dolor in reprehenderit in voluptate velit
                esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
                occaecat cupidatat non proident, sunt in culpa qui officia
                deserunt mollit anim id est laborum.
            </TextBlock>
        </ScrollViewer>
    </Expander.Content>
</Expander>
```

C#

```
//Create Expander object
Expander exp = new Expander();
```

```

//Create Bullet Panel for Expander Header
BulletDecorator bp = new BulletDecorator();
Image i = new Image();
BitmapImage bi= new BitmapImage();
bi.UriSource = new Uri(@"pack://application:,,/images/icon.jpg");
i.Source = bi;
i.Width = 10;
bp.Bullet = i;
TextBlock tb = new TextBlock();
tb.Text = "My Expander";
tb.Margin = new Thickness(20,0,0,0);
bp.Child = tb;
exp.Header = bp;

//Create TextBlock with ScrollViewer for Expander Content
StackPanel spScroll = new StackPanel();
TextBlock tbc = new TextBlock();
tbc.Text =
    "Lorem ipsum dolor sit amet, consectetur adipisicing elit," +
    "sed do eiusmod tempor incididunt ut labore et dolore magna" +
    "aliqua. Ut enim ad minim veniam, quis nostrud exercitation" +
    "ullamco laboris nisi ut aliquip ex ea commodo consequat." +
    "Duis aute irure dolor in reprehenderit in voluptate velit" +
    "esse cillum dolore eu fugiat nulla pariatur. Excepteur sint" +
    "occaecat cupidatat non proident, sunt in culpa qui officia" +
    "deserunt mollit anim id est laborum.";
tbc.TextWrapping = TextWrapping.Wrap;

spScroll.Children.Add(tbc);
ScrollViewer scr = new ScrollViewer();
scr.Content = spScroll;
scr.Height = 50;
exp.Content = scr;

exp.Width=200;
exp.HorizontalContentAlignment= HorizontalAlignment.Stretch;

```

另请参阅

- [Expander](#)
- [Expander 概述](#)
- [操作指南主题](#)

FlowDocumentPageViewer

项目 • 2023/02/06

FlowDocumentPageViewer 控件用于按页面查看 FlowDocument 内容。不同于 FlowDocumentScrollView，后者在滚动查看器中显示 FlowDocument 内容。

另请参阅

- [FlowDocument](#)
- [流文档概述](#)
- [操作指南主题](#)
- [WPF 中的文档](#)

FlowDocumentReader

项目 • 2023/02/06

该 [FlowDocumentReader](#) 控件用于查看 [FlowDocument](#) 内容。 它支持多种查看模式。

另请参阅

- [FlowDocumentReader](#)
- [FlowDocumentPageViewer](#)
- [FlowDocumentScrollView](#)
- [WPF 中的文档](#)
- [流文档概述](#)

FlowDocumentScrollView

项目 • 2023/02/06

FlowDocumentScrollView 控件用于查看滚动容器中的 FlowDocument 内容。 这与 FlowDocumentPageViewer 不同，它查看每个页面上的内容。

另请参阅

- [FlowDocumentReader](#)
- [FlowDocumentPageViewer](#)
- [FlowDocumentScrollView](#)
- [FlowDocument](#)
- [WPF 中的文档](#)
- [流文档概述](#)

Frame

项目 • 2023/02/06

Frame 控件支持在内容中进行导航。 Frame 可以由 Window、NavigationWindow、Page、UserControl、FlowDocument 等根元素托管，也可以作为根元素的内容树中的一个孤岛。

参考

[Frame](#)

相关章节

[导航概述](#)

网格

项目 • 2023/02/06

[Grid](#) 元素用于在行和列中精确定位内容。

本节内容

[操作指南主题](#)

参考

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

相关章节

[布局](#)

[演练：我的第一个 WPF 桌面应用程序](#)

[ScrollViewer 概述](#)

Grid 帮助主题

项目 • 2023/02/06

本部分中的主题介绍如何使用 [Grid](#) 元素定位元素。

本节内容

[使用 Grid 生成标准 UI 对话框](#)

[创建复杂网格](#)

[创建 Grid 元素](#)

[创建和使用 GridLengthConverter 对象](#)

[使用 ColumnDefinitionsCollection 和 RowDefinitionsCollection 操作列和行](#)

[定位 Grid 的子元素](#)

[在 Grid 之间共享重设大小属性](#)

参考

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

相关章节

[布局](#)

[演练：我的第一个 WPF 桌面应用程序](#)

[ScrollViewer 概述](#)

如何：使用 Grid 来构建标准的 UI 对话框

项目 • 2023/02/06

此示例演示如何使用 [Grid](#) 元素创建标准用户界面 (UI) 对话框。

示例

以下示例在 Windows 操作系统中创建一个类似于“运行”对话框的对话框。

该示例创建一个 [Grid](#) 并使用 [ColumnDefinition](#) 和 [RowDefinition](#) 类来定义五列和四行。

然后，该示例添加并放置 [Image](#)、[RunIcon.png](#)，以表示在对话框中找到的图像。该图像放置在 [Grid](#) 的第一列和第一行（左上角）。

接下来，该示例将 [TextBlock](#) 元素添加到第一列，该元素跨越第一行的其余列。它将另一个 [TextBlock](#) 元素添加到第一列的第二行，以表示“打开”文本框。后面是 [TextBlock](#)，表示数据输入区域。

最后，该示例将三个 [Button](#) 元素添加到最后一行，它们代表“确认”、“取消”和“浏览”事件。

C#

```
// Create the Grid.
grid1 = new Grid();
grid1.Background = Brushes.Gainsboro;
grid1.HorizontalAlignment = HorizontalAlignment.Left;
grid1.VerticalAlignment = VerticalAlignment.Top;
grid1.ShowGridLines = true;
grid1.Width = 425;
grid1.Height = 165;

// Define the Columns.
colDef1 = new ColumnDefinition();
colDef1.Width = new GridLength(1, GridUnitType.Auto);
colDef2 = new ColumnDefinition();
colDef2.Width = new GridLength(1, GridUnitType.Star);
colDef3 = new ColumnDefinition();
colDef3.Width = new GridLength(1, GridUnitType.Star);
colDef4 = new ColumnDefinition();
colDef4.Width = new GridLength(1, GridUnitType.Star);
colDef5 = new ColumnDefinition();
colDef5.Width = new GridLength(1, GridUnitType.Star);
grid1.ColumnDefinitions.Add(colDef1);
grid1.ColumnDefinitions.Add(colDef2);
grid1.ColumnDefinitions.Add(colDef3);
grid1.ColumnDefinitions.Add(colDef4);
```

```
grid1.ColumnDefinitions.Add(colDef5);

// Define the Rows.
rowDef1 = new RowDefinition();
rowDef1.Height = new GridLength(1, GridUnitType.Auto);
rowDef2 = new RowDefinition();
rowDef2.Height = new GridLength(1, GridUnitType.Auto);
rowDef3 = new RowDefinition();
rowDef3.Height = new GridLength(1, GridUnitType.Star);
rowDef4 = new RowDefinition();
rowDef4.Height = new GridLength(1, GridUnitType.Auto);
grid1.RowDefinitions.Add(rowDef1);
grid1.RowDefinitions.Add(rowDef2);
grid1.RowDefinitions.Add(rowDef3);
grid1.RowDefinitions.Add(rowDef4);

// Add the Image.
img1 = new Image();
img1.Source = new System.Windows.Media.Imaging.BitmapImage(new Uri("runicon.png", UriKind.Relative));
Grid.SetRow(img1, 0);
Grid.SetColumn(img1, 0);

// Add the main application dialog.
txt1 = new TextBlock();
txt1.Text = "Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.";
txt1.TextWrapping = TextWrapping.Wrap;
Grid.SetColumnSpan(txt1, 4);
Grid.SetRow(txt1, 0);
Grid.SetColumn(txt1, 1);

// Add the second text cell to the Grid.
txt2 = new TextBlock();
txt2.Text = "Open:";
Grid.SetRow(txt2, 1);
Grid.SetColumn(txt2, 0);

// Add the TextBox control.
tb1 = new TextBox();
Grid.SetRow(tb1, 1);
Grid.SetColumn(tb1, 1);
Grid.SetColumnSpan(tb1, 5);

// Add the buttons.
button1 = new Button();
button2 = new Button();
button3 = new Button();
button1.Content = "OK";
button2.Content = "Cancel";
button3.Content = "Browse ...";
Grid.SetRow(button1, 3);
Grid.SetColumn(button1, 2);
button1.Margin = new Thickness(10, 0, 10, 15);
button2.Margin = new Thickness(10, 0, 10, 15);
```

```
button3.Margin = new Thickness(10, 0, 10, 15);
Grid.SetRow(button2, 3);
Grid.SetColumn(button2, 3);
Grid.SetRow(button3, 3);
Grid.SetColumn(button3, 4);

grid1.Children.Add(img1);
grid1.Children.Add(txt1);
grid1.Children.Add(txt2);
grid1.Children.Add(tb1);
grid1.Children.Add(button1);
grid1.Children.Add(button2);
grid1.Children.Add(button3);

mainWindow.Content = grid1;
```

另请参阅

- [Grid](#)
- [GridUnitType](#)
- [面板概述](#)
- [操作指南主题](#)

如何创建复杂网格

项目 • 2022/09/27

此示例说明如何使用 Grid 控件创建类似于月历的布局。

示例

以下示例使用 `RowDefinition` 和 `ColumnDefinition` 类定义八行和八列。它使用 `Grid.ColumnSpan` 和 `Grid.RowSpan` 附加属性以及 `Rectangle` 元素，这些元素填充各种列和行的背景。这种设计是可能实现的，因为 `Grid` 的每个单元格中可以存在多个元素，这是 `Grid` 和 `Table` 之间的主要区别。

该示例使用垂直渐变来 Fill 列和行，以改善日历的视觉呈现和可读性。带样式的 TextBlock 元素表示日期和星期几。通过使用在应用程序样式中定义的 Margin 属性和对齐属性在其单元格中对 TextBlock 元素进行绝对定位。

XAML

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      WindowTitle="Complex Grid Example">
<Border BorderBrush="Black">

<Grid ShowGridLines="false" Background="White">
    <Grid.Resources>
        <Style TargetType="{x:Type ColumnDefinition}">
            <Setter Property="Width" Value="30"/>
        </Style>
        <Style TargetType="{x:Type Rectangle}">
            <Setter Property="RadiusX" Value="6"/>
            <Setter Property="RadiusY" Value="6"/>
        </Style>
        <Style x:Key="DayOfWeek">
            <Setter Property="Grid.Row" Value="1"/></Setter>
            <Setter Property="TextBlock.Margin" Value="5"/></Setter>
        </Style>
        <Style x:Key="OneDate">
            <Setter Property="TextBlock.Margin" Value="5"/></Setter>
        </Style>
    </Grid.Resources>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="30" />
        <RowDefinition Height="30" />
    </Grid.RowDefinitions>
    <Rectangle RadiusX="6" RadiusY="6" Width="30" Height="30" Margin="5" />
    <TextBlock Text="Monday" Style="{StaticResource DayOfWeek}" Margin="5" />
    <TextBlock Text="Tuesday" Style="{StaticResource DayOfWeek}" Margin="5" />
    <TextBlock Text="Wednesday" Style="{StaticResource DayOfWeek}" Margin="5" />
    <TextBlock Text="Thursday" Style="{StaticResource DayOfWeek}" Margin="5" />
    <TextBlock Text="Friday" Style="{StaticResource DayOfWeek}" Margin="5" />
    <TextBlock Text="Saturday" Style="{StaticResource DayOfWeek}" Margin="5" />
    <TextBlock Text="Sunday" Style="{StaticResource DayOfWeek}" Margin="5" />
    <TextBlock Text="One Date" Style="{StaticResource OneDate}" Margin="5" />

```

```

        <!-- This column will receive all remaining width -->
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <!-- This row will receive all remaining Height -->
        <RowDefinition/>
    </Grid.RowDefinitions>

    <!-- These Rectangles constitute the backgrounds of the various Rows
and Columns -->

    <Rectangle Grid.ColumnSpan="7" Fill="#73B2F5"/>
    <Rectangle Grid.Row="1" Grid.RowSpan="6" Fill="#73B2F5"/>
    <Rectangle Grid.Column="6" Grid.Row="1" Grid.RowSpan="6"
Fill="#73B2F5"/>
        <Rectangle Grid.Column="1" Grid.Row="1" Grid.ColumnSpan="5"
Grid.RowSpan="6"
Fill="#efefef"/>

    <!-- Month row -->
    <TextBlock Grid.ColumnSpan="7" Margin="0,5,0,5"
HorizontalAlignment="Center">
        January 2004
    </TextBlock>

    <!-- Draws a separator under the days-of-the-week row -->

    <Rectangle Grid.Row="1" Grid.ColumnSpan="7"
Fill="Black" RadiusX="1" RadiusY="1" Height="2" Margin="0,20,0,0"/>

    <!-- Day-of-the-week row -->
    <TextBlock Grid.Column="0" Style="{StaticResource
DayOfWeek}">Sun</TextBlock>
    <TextBlock Grid.Column="1" Style="{StaticResource
DayOfWeek}">Mon</TextBlock>
    <TextBlock Grid.Column="2" Style="{StaticResource
DayOfWeek}">Tue</TextBlock>
    <TextBlock Grid.Column="3" Style="{StaticResource
DayOfWeek}">Wed</TextBlock>
    <TextBlock Grid.Column="4" Style="{StaticResource
DayOfWeek}">Thu</TextBlock>
    <TextBlock Grid.Column="5" Style="{StaticResource
DayOfWeek}">Fri</TextBlock>
    <TextBlock Grid.Column="6" Style="{StaticResource
DayOfWeek}">Sat</TextBlock>

    <!-- Dates go here -->

```



```
OneDate}>28</TextBlock>
    <TextBlock Grid.Column="4" Grid.Row="6" Style="{StaticResource
OneDate}>29</TextBlock>
    <TextBlock Grid.Column="5" Grid.Row="6" Style="{StaticResource
OneDate}>30</TextBlock>
    <TextBlock Grid.Column="6" Grid.Row="6" Style="{StaticResource
OneDate}>31</TextBlock>
</Grid>
</Border>
</Page>
```

下图显示了生成的控件，一个可自定义的日历：



另请参阅

- [System.Windows.Controls.Grid](#)
- [System.Windows.Documents.TableCell](#)
- [使用纯色和渐变进行绘制概述](#)
- [面板概述](#)
- [表概述](#)

如何：创建网格元素

项目 • 2023/02/06

示例

以下示例演示如何使用 Extensible Application Markup Language (XAML) 或代码创建和使用 `Grid` 实例。此示例使用三个 `ColumnDefinition` 对象和三个 `RowDefinition` 对象来创建一个具有九个单元格的网格，例如在工作表中。每个单元格都包含一个表示数据的 `TextBlock` 元素，顶部行包含一个应用了 `ColumnSpan` 属性的 `TextBlock`。为了显示每个单元格的边界，启用了 `ShowGridLines` 属性。

C#

```
// Create the application's main window
mainWindow = new Window();
mainWindow.Title = "Grid Sample";

// Create the Grid
Grid myGrid = new Grid();
myGrid.Width = 250;
myGrid.Height = 100;
myGrid.HorizontalAlignment = HorizontalAlignment.Left;
myGrid.VerticalAlignment = VerticalAlignment.Top;
myGrid.ShowGridLines = true;

// Define the Columns
ColumnDefinition colDef1 = new ColumnDefinition();
ColumnDefinition colDef2 = new ColumnDefinition();
ColumnDefinition colDef3 = new ColumnDefinition();
myGrid.ColumnDefinitions.Add(colDef1);
myGrid.ColumnDefinitions.Add(colDef2);
myGrid.ColumnDefinitions.Add(colDef3);

// Define the Rows
RowDefinition rowDef1 = new RowDefinition();
RowDefinition rowDef2 = new RowDefinition();
RowDefinition rowDef3 = new RowDefinition();
RowDefinition rowDef4 = new RowDefinition();
myGrid.RowDefinitions.Add(rowDef1);
myGrid.RowDefinitions.Add(rowDef2);
myGrid.RowDefinitions.Add(rowDef3);
myGrid.RowDefinitions.Add(rowDef4);

// Add the first text cell to the Grid
TextBlock txt1 = new TextBlock();
txt1.Text = "2005 Products Shipped";
txt1.FontSize = 20;
txt1.FontWeight = FontWeights.Bold;
```

```
Grid.SetColumnSpan(txt1, 3);
Grid.SetRow(txt1, 0);

// Add the second text cell to the Grid
TextBlock txt2 = new TextBlock();
txt2.Text = "Quarter 1";
txt2.FontSize = 12;
txt2.FontWeight = FontWeights.Bold;
Grid.SetRow(txt2, 1);
Grid.SetColumn(txt2, 0);

// Add the third text cell to the Grid
TextBlock txt3 = new TextBlock();
txt3.Text = "Quarter 2";
txt3.FontSize = 12;
txt3.FontWeight = FontWeights.Bold;
Grid.SetRow(txt3, 1);
Grid.SetColumn(txt3, 1);

// Add the fourth text cell to the Grid
TextBlock txt4 = new TextBlock();
txt4.Text = "Quarter 3";
txt4.FontSize = 12;
txt4.FontWeight = FontWeights.Bold;
Grid.SetRow(txt4, 1);
Grid.SetColumn(txt4, 2);

// Add the sixth text cell to the Grid
TextBlock txt5 = new TextBlock();
Double db1 = new Double();
db1 = 50000;
txt5.Text = db1.ToString();
Grid.SetRow(txt5, 2);
Grid.SetColumn(txt5, 0);

// Add the seventh text cell to the Grid
TextBlock txt6 = new TextBlock();
Double db2 = new Double();
db2 = 100000;
txt6.Text = db2.ToString();
Grid.SetRow(txt6, 2);
Grid.SetColumn(txt6, 1);

// Add the final text cell to the Grid
TextBlock txt7 = new TextBlock();
Double db3 = new Double();
db3 = 150000;
txt7.Text = db3.ToString();
Grid.SetRow(txt7, 2);
Grid.SetColumn(txt7, 2);

// Total all Data and Span Three Columns
TextBlock txt8 = new TextBlock();
txt8.FontSize = 16;
txt8.FontWeight = FontWeights.Bold;
```

```

txt8.Text = "Total Units: " + (db1 + db2 + db3).ToString();
Grid.SetRow(txt8, 3);
Grid.SetColumnSpan(txt8, 3);

// Add the TextBlock elements to the Grid Children collection
myGrid.Children.Add(txt1);
myGrid.Children.Add(txt2);
myGrid.Children.Add(txt3);
myGrid.Children.Add(txt4);
myGrid.Children.Add(txt5);
myGrid.Children.Add(txt6);
myGrid.Children.Add(txt7);
myGrid.Children.Add(txt8);

// Add the Grid as the Content of the Parent Window Object
mainWindow.Content = myGrid;
mainWindow.Show ();

```

XAML

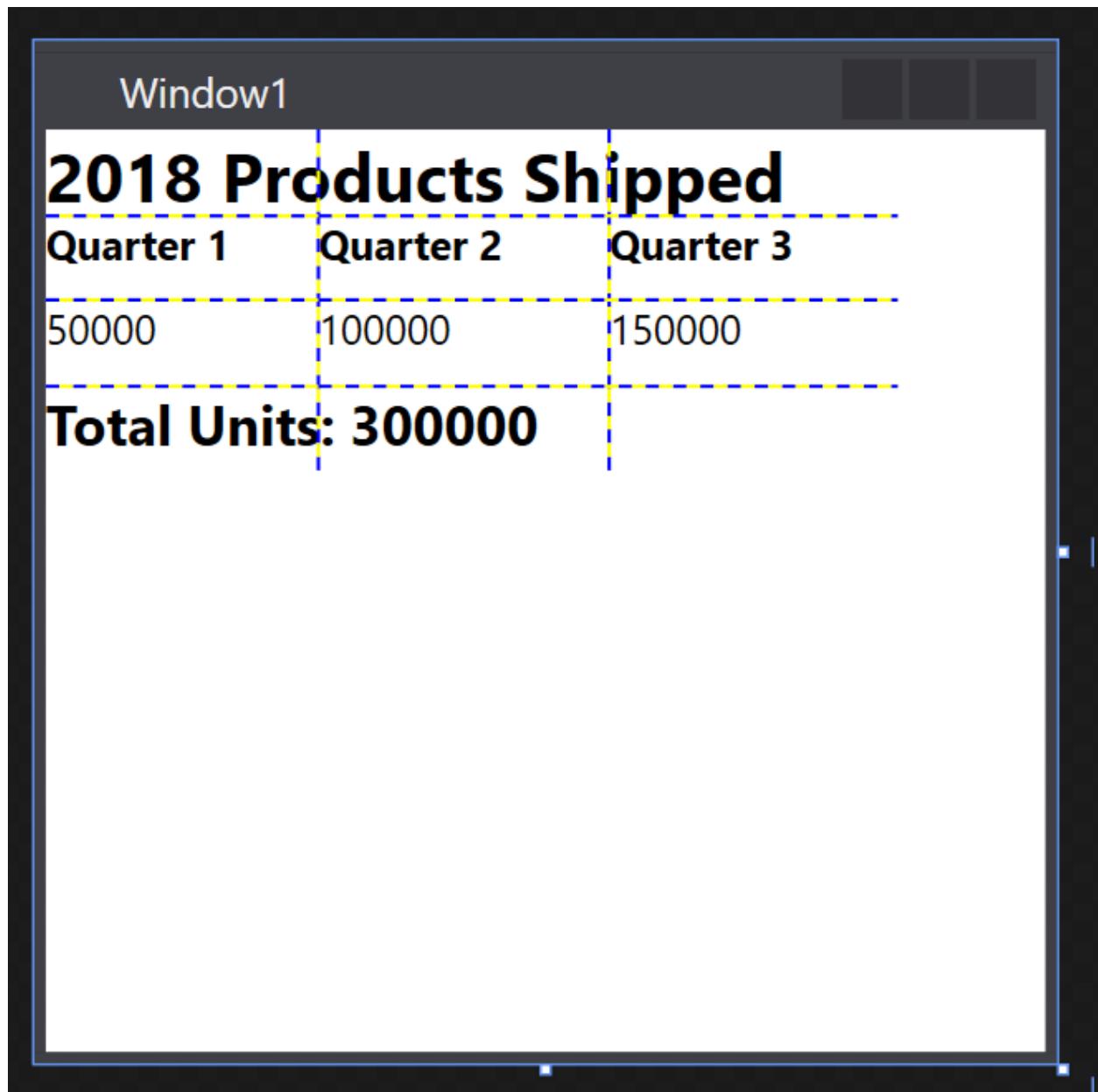
```

<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
WindowTitle="Grid Sample">
    <Grid VerticalAlignment="Top" HorizontalAlignment="Left"
ShowGridLines="True" Width="250" Height="100">
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
            <ColumnDefinition />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
        </Grid.RowDefinitions>

        <TextBlock FontSize="20" FontWeight="Bold" Grid.ColumnSpan="3"
Grid.Row="0">2005 Products Shipped</TextBlock>
        <TextBlock FontSize="12" FontWeight="Bold" Grid.Row="1"
Grid.Column="0">Quarter 1</TextBlock>
        <TextBlock FontSize="12" FontWeight="Bold" Grid.Row="1"
Grid.Column="1">Quarter 2</TextBlock>
        <TextBlock FontSize="12" FontWeight="Bold" Grid.Row="1"
Grid.Column="2">Quarter 3</TextBlock>
        <TextBlock Grid.Row="2" Grid.Column="0">50000</TextBlock>
        <TextBlock Grid.Row="2" Grid.Column="1">100000</TextBlock>
        <TextBlock Grid.Row="2" Grid.Column="2">150000</TextBlock>
        <TextBlock FontSize="16" FontWeight="Bold" Grid.ColumnSpan="3"
Grid.Row="3">Total Units: 300000</TextBlock>
    </Grid>
</Page>

```

任一方法都会生成一个看起来非常相似的用户界面，如下所示。



另请参阅

- Grid
- 面板概述

如何：创建和使用 GridLengthConverter 对象

项目 • 2023/02/06

示例

下面的示例演示如何创建并使用 [GridLengthConverter](#) 的实例。该示例定义了一个名为 `changeCol` 的自定义方法，该方法将 [ListBoxItem](#) 传递给 [GridLengthConverter](#)，从而将 [ListBoxItem](#) 的 [Content](#) 转换为 [GridLength](#) 的实例。然后，此转换值作为 [ColumnDefinition](#) 元素的 [Width](#) 属性的值传回。

此示例还定义称为 `changeColVal` 的另一个自定义方法。此自定义方法将 [Slider](#) 的 [Value](#) 转换为 [String](#)，然后将该值作为元素的 [Width](#) 传递回 [ColumnDefinition](#)。

请注意，单独的 Extensible Application Markup Language (XAML) 文件可定义 [ListBoxItem](#) 内容。

C#

```
private void changeColVal(object sender, RoutedEventArgs e)
{
    txt1.Text = "Current Grid Column is " + hs1.Value.ToString();
}

private void changeCol(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    GridLengthConverter myGridLengthConverter = new GridLengthConverter();
    if (hs1.Value == 0)
    {
        GridLength gl1 =
(GridLength)myGridLengthConverter.ConvertFromString(li.Content.ToString());
        col1.Width = gl1;
    }
    else if (hs1.Value == 1)
    {
        GridLength gl2 =
(GridLength)myGridLengthConverter.ConvertFromString(li.Content.ToString());
        col2.Width = gl2;
    }
    else if (hs1.Value == 2)
    {
        GridLength gl3 =
(GridLength)myGridLengthConverter.ConvertFromString(li.Content.ToString());
        col3.Width = gl3;
    }
}
```

```
    }  
}
```

另请参阅

- [GridLengthConverter](#)
- [GridLength](#)

如何：使用 ColumnDefinitionsCollection 和 RowDefinitionsCollection 来操作列和行

项目 • 2023/02/06

此示例演示如何使用 [ColumnDefinitionCollection](#) 和 [RowDefinitionCollection](#) 类中的方法对行或列的内容执行添加、清除或计数等操作。例如，可以 [Add](#)、[Clear](#) 或 [Count](#) 包含在 [ColumnDefinition](#) 或 [RowDefinition](#) 中的项。

示例

以下示例使用 `grid1` 的 [Name](#) 创建 [Grid](#) 元素。[Grid](#) 包含 [StackPanel](#)，后者保存了 [Button](#) 元素，其中每个元素由不同的集合方法控制。单击 [Button](#) 时，它会激活代码隐藏文件中的方法调用。

XAML

```
<DockPanel Margin="10,0,0,0">

    <TextBlock FontSize="20" FontWeight="Bold" DockPanel.Dock="Top">Grid
    Column and Row Collections</TextBlock>
    <TextBlock DockPanel.Dock="Top">Click any of the buttons below to invoke
    the associated methods and properties of ColumnDefinition and RowDefinition
    Collections.</TextBlock>
    <Grid DockPanel.Dock="Top" HorizontalAlignment="Left" Name="grid1"
    ShowGridLines="true" Width="625" Height="400" Background="#b0e0e6">
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition/>
            <RowDefinition/>
        </Grid.RowDefinitions>
    </Grid>

    <StackPanel HorizontalAlignment="Left" Orientation="Horizontal"
    Width="625" DockPanel.Dock="Top">
        <Button Width="125" Click="addCol">Add Column</Button>
        <Button Width="125" Click="addRow">Add Row</Button>
        <Button Width="125" Click="clearCol">Clear All Columns</Button>
        <Button Width="125" Click="clearRow">Clear All Rows</Button>
        <Button Width="125" Click="removeCol">Remove One Column</Button>
    </StackPanel>
```

```

<StackPanel HorizontalAlignment="Left" Orientation="Horizontal"
Width="625" DockPanel.Dock="Top">
    <Button Width="125" Click="removeRow">Remove One Row</Button>
    <Button Width="125" Click="colCount">How Many Columns?</Button>
    <Button Width="125" Click="rowCount">How Many Rows?</Button>
    <Button Width="125" Click="rem5Col">Remove 5 Columns</Button>
    <Button Width="125" Click="rem5Row">Remove 5 Rows</Button>
</StackPanel>
<StackPanel HorizontalAlignment="Left" Orientation="Horizontal"
Width="625" DockPanel.Dock="Top">
    <Button Width="125" Click="containsRow">Contains Row?</Button>
    <Button Width="125" Click="containsCol">Contains Column?</Button>
    <Button Width="125" Click="insertRowAt">Insert Row</Button>
    <Button Width="125" Click="insertColAt">Insert Column</Button>
    <Button Width="125" Click="colReadOnly">IsReadOnly?</Button>
</StackPanel>
<TextBlock DockPanel.Dock="Top" Name="tp1"/>
<TextBlock DockPanel.Dock="Top" Name="tp2"/>
<TextBlock DockPanel.Dock="Top" Name="tp3"/>
<TextBlock DockPanel.Dock="Top" Name="tp4"/>
<TextBlock DockPanel.Dock="Top" Name="tp5"/>
</DockPanel>

```

此示例定义一系列自定义方法，每个方法对应于 Extensible Application Markup Language (XAML) 文件中的 `Click` 事件。可以通过多种方法更改 `Grid` 中的列数和行数，这些方法包括添加或删除行和列，以及计算行和列的总数。若要防止 `ArgumentOutOfRangeException` 和 `ArgumentException` 异常，可以使用 `RemoveRange` 方法提供的错误检查功能。

C#

```

using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;

namespace columndefinitions_grid
{
    public partial class Window1 : Window
    {
        RowDefinition rowDef1;
        ColumnDefinition colDef1;

        private void addCol(object sender, RoutedEventArgs e)
        {
            colDef1 = new ColumnDefinition();
            grid1.ColumnDefinitions.Add(colDef1);
        }

        private void addRow(object sender, RoutedEventArgs e)
        {
            rowDef1 = new RowDefinition();

```

```

        grid1.RowDefinitions.Add(rowDef1);
    }

    private void clearCol(object sender, RoutedEventArgs e)
    {
        grid1.ColumnDefinitions.Clear();
    }

    private void clearRow(object sender, RoutedEventArgs e)
    {
        grid1.RowDefinitions.Clear();
    }

    private void removeCol(object sender, RoutedEventArgs e)
    {
        if (grid1.ColumnDefinitions.Count <= 0)
        {
            tp1.Text = "No More Columns to Remove!";
        }
        else
        {
            grid1.ColumnDefinitions.RemoveAt(0);
        }
    }

    private void removeRow(object sender, RoutedEventArgs e)
    {
        if (grid1.RowDefinitions.Count <= 0)
        {
            tp1.Text = "No More Rows to Remove!";
        }
        else
        {
            grid1.RowDefinitions.RemoveAt(0);
        }
    }

    private void colCount(object sender, RoutedEventArgs e)
    {
        tp2.Text = "The current number of Columns is: " +
grid1.ColumnDefinitions.Count;
    }

    private void rowCount(object sender, RoutedEventArgs e)
    {
        tp2.Text = "The current number of Rows is: " +
grid1.RowDefinitions.Count;
    }

    private void rem5Col(object sender, RoutedEventArgs e)
    {
        if (grid1.ColumnDefinitions.Count < 5)
        {
            tp1.Text = "There aren't five Columns to Remove!";
        }
    }

```

```

        else
    {
        grid1.ColumnDefinitions.RemoveRange(0,5);
    }
}

private void rem5Row(object sender, RoutedEventArgs e)
{
    if (grid1.RowDefinitions.Count < 5)
    {
        tp1.Text = "There aren't five Rows to Remove!";
    }
    else
    {
        grid1.RowDefinitions.RemoveRange(0, 5);
    }
}

private void containsRow(object sender, RoutedEventArgs e)
{
    if (grid1.RowDefinitions.Contains(rowDef1))
    {
        tp2.Text = "Grid Contains RowDefinition rowDef1";
    }
    else
    {
        tp2.Text = "Grid Does Not Contain RowDefinition rowDef1";
    }
}

private void containsCol(object sender, RoutedEventArgs e)
{
    if (grid1.ColumnDefinitions.Contains(colDef1))
    {
        tp3.Text = "Grid Contains ColumnDefinition colDef1";
    }
    else
    {
        tp3.Text = "Grid Does Not Contain ColumnDefinition colDef1";
    }
}

private void colReadOnly(object sender, RoutedEventArgs e)
{
    tp4.Text = "RowDefinitionsCollection IsReadOnly?: " +
grid1.RowDefinitions.IsReadOnly.ToString();
    tp5.Text = "ColumnDefinitionsCollection IsReadOnly?: " +
grid1.ColumnDefinitions.IsReadOnly.ToString();
}

private void insertRowAt(object sender, RoutedEventArgs e)
{
    rowDef1 = new RowDefinition();
    grid1.RowDefinitions.Insert(grid1.RowDefinitions.Count,
rowDef1);
}

```

```
        tp1.Text = "RowDefinition added at index position " +
grid1.RowDefinitions.IndexOf(rowDef1).ToString();
    }

    private void insertColAt(object sender, RoutedEventArgs e)
{
    colDef1 = new ColumnDefinition();
    grid1.ColumnDefinitions.Insert(grid1.ColumnDefinitions.Count,
colDef1);
    tp2.Text = "ColumnDefinition added at index position " +
grid1.ColumnDefinitions.IndexOf(colDef1).ToString();
}
}
```

另请参阅

- [Grid](#)
- [ColumnDefinitionCollection](#)
- [RowDefinitionCollection](#)

如何：定位网格的子元素

项目 • 2023/02/06

此示例演示如何使用在 Grid 上定义的 get 和 set 方法来定位子元素。

示例

以下示例定义具有三列和三行的父 Grid 元素 (grid1)。 子 Rectangle 元素 (rect1) 被添加到列位置为零、行位置为零的 Grid。 Button 元素表示可调用以在 Grid 中重新定位 Rectangle 元素的方法。 当用户单击某个按钮时，将激活相关方法。

XAML

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="grid_getset_methods.Window1"
    Title="Grid Methods Sample">
    <Border BorderBrush="Black" Background="White" BorderThickness="2">
        <DockPanel VerticalAlignment="Top" HorizontalAlignment="Left"
Margin="10">
            <TextBlock FontSize="20" FontWeight="Bold" DockPanel.Dock="Top">Grid
Methods Sample</TextBlock>
            <TextBlock DockPanel.Dock="Top">Click the buttons on the left to
reposition the Rectangle below using methods defined on Grid.</TextBlock>
            <Grid Margin="0,10,15,0" DockPanel.Dock="Left">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition/>
                    <ColumnDefinition/>
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition/>
                </Grid.RowDefinitions>
                <!-- <Snippet1> -->
                <StackPanel Grid.Column="0" Grid.Row="0"
HorizontalAlignment="Left" Orientation="Vertical">
                    <Button Click="setCol0">Move Rectangle to Column 0</Button>
                    <Button Click="setCol1">Move Rectangle to Column 1</Button>
                    <Button Click="setCol2" Margin="0,0,0,10">Move Rectangle to
Column 2</Button>
                    <Button Click="setRow0">Move Rectangle to Row 0</Button>
                    <Button Click="setRow1">Move Rectangle to Row 1</Button>
                    <Button Click="setRow2" Margin="0,0,0,10">Move Rectangle to Row
2</Button>
                    <Button Click="setColspan">Span All Columns</Button>
                    <Button Click="setRowspan">Span All Rows</Button>
                    <Button Click="clearAll">Clear All</Button>
                </StackPanel>
            </Grid>
        </DockPanel>
    </Border>
</Window>
```

```

        <Grid DockPanel.Dock="Top" Margin="0,10,15,0"
HorizontalAlignment="Left" Name="grid1" ShowGridLines="True" Width="400"
Height="400" Background="LightSteelBlue">
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>

        <Rectangle Name="rect1" Fill="Silver" Grid.Column="0" Grid.Row="0"/>
        <TextBlock FontSize="15" HorizontalAlignment="Right"
VerticalAlignment="Bottom" Grid.Column="0" Grid.Row="0" Margin="5">Column 0,
Row 0</TextBlock>
        <TextBlock FontSize="15" HorizontalAlignment="Right"
VerticalAlignment="Bottom" Grid.Column="1" Grid.Row="0" Margin="5">Column 1,
Row 0</TextBlock>
        <TextBlock FontSize="15" HorizontalAlignment="Right"
VerticalAlignment="Bottom" Grid.Column="2" Grid.Row="0" Margin="5">Column 2,
Row 0</TextBlock>
        <TextBlock FontSize="15" HorizontalAlignment="Right"
VerticalAlignment="Bottom" Grid.Column="0" Grid.Row="1" Margin="5">Column 0,
Row 1</TextBlock>
        <TextBlock FontSize="15" HorizontalAlignment="Right"
VerticalAlignment="Bottom" Grid.Column="1" Grid.Row="1" Margin="5">Column 1,
Row 1</TextBlock>
        <TextBlock FontSize="15" HorizontalAlignment="Right"
VerticalAlignment="Bottom" Grid.Column="2" Grid.Row="1" Margin="5">Column 2,
Row 1</TextBlock>
        <TextBlock FontSize="15" HorizontalAlignment="Right"
VerticalAlignment="Bottom" Grid.Column="0" Grid.Row="2" Margin="5">Column 0,
Row 2</TextBlock>
        <TextBlock FontSize="15" HorizontalAlignment="Right"
VerticalAlignment="Bottom" Grid.Column="1" Grid.Row="2" Margin="5">Column 1,
Row 2</TextBlock>
        <TextBlock FontSize="15" HorizontalAlignment="Right"
VerticalAlignment="Bottom" Grid.Column="2" Grid.Row="2" Margin="5">Column 2,
Row 2</TextBlock>
    </Grid>
    <!-- </Snippet1> -->

        <TextBlock DockPanel.Dock="Top" Name="txt1"/>
        <TextBlock DockPanel.Dock="Top" Name="txt2"/>
        <TextBlock DockPanel.Dock="Top" Name="txt3"/>
        <TextBlock DockPanel.Dock="Top" Name="txt4"/>
    </DockPanel>
    </Border>
</Window>

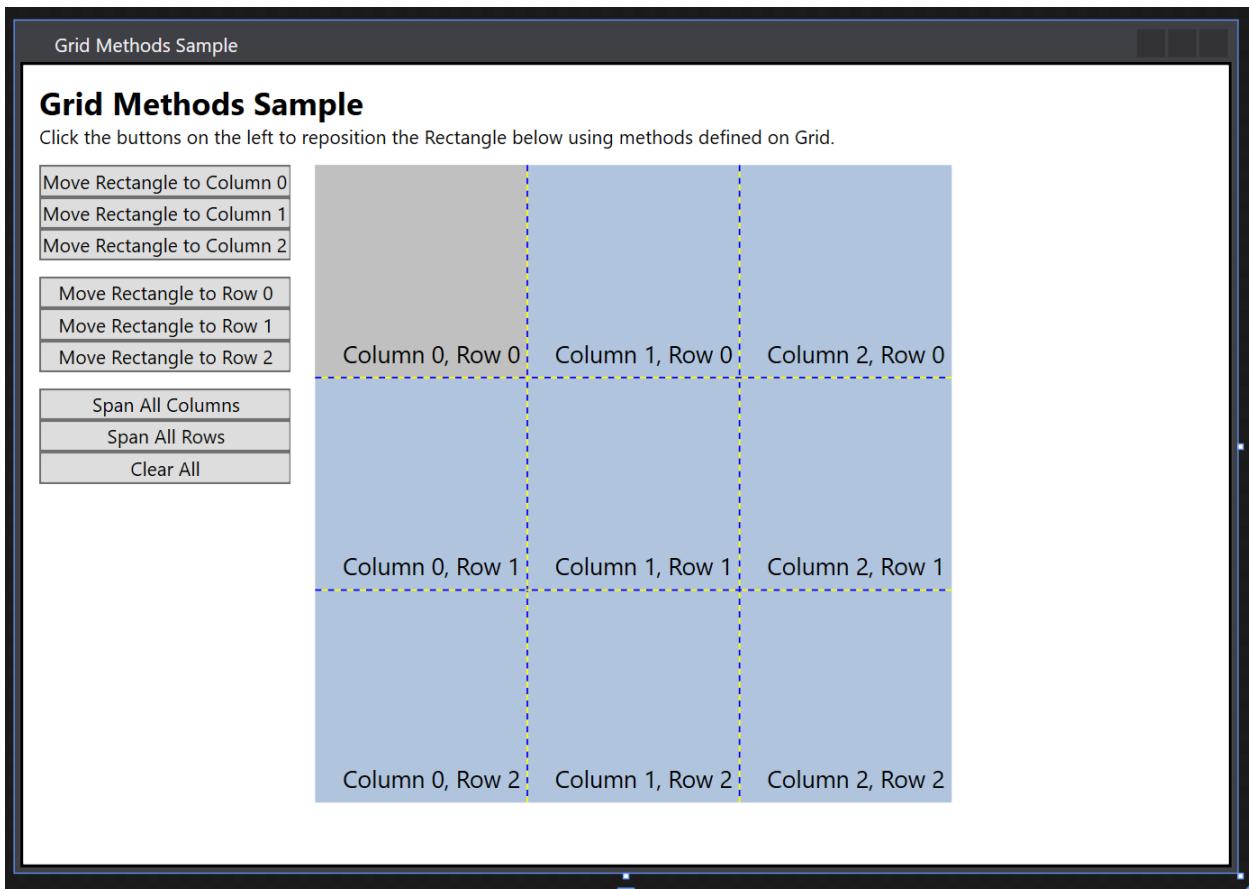
```

以下代码隐藏示例处理按钮 Click 事件引发的方法。该示例将这些方法调用写入 TextBlock 元素，这些元素使用相关的 get 方法以字符串形式输出新属性值。

C#

```
private void setCol0(object sender, RoutedEventArgs e)
{
    Grid.SetColumn(rect1, 0);
    txt1.Text = "Rectangle is in Column " +
Grid.GetColumn(rect1).ToString();
}
private void setCol1(object sender, RoutedEventArgs e)
{
    Grid.SetColumn(rect1, 1);
    txt1.Text = "Rectangle is in Column " +
Grid.GetColumn(rect1).ToString();
}
private void setCol2(object sender, RoutedEventArgs e)
{
    Grid.SetColumn(rect1, 2);
    txt1.Text = "Rectangle is in Column " +
Grid.GetColumn(rect1).ToString();
}
private void setRow0(object sender, RoutedEventArgs e)
{
    Grid.SetRow(rect1, 0);
    txt2.Text = "Rectangle is in Row " + Grid.GetRow(rect1).ToString();
}
private void setRow1(object sender, RoutedEventArgs e)
{
    Grid.SetRow(rect1, 1);
    txt2.Text = "Rectangle is in Row " + Grid.GetRow(rect1).ToString();
}
private void setRow2(object sender, RoutedEventArgs e)
{
    Grid.SetRow(rect1, 2);
    txt2.Text = "Rectangle is in Row " + Grid.GetRow(rect1).ToString();
}
private void setColspan(object sender, RoutedEventArgs e)
{
    Grid.SetColumnSpan(rect1, 3);
    txt3.Text = "ColumnSpan is set to " +
Grid.GetColumnSpan(rect1).ToString();
}
private void setRowspan(object sender, RoutedEventArgs e)
{
    Grid.SetRowSpan(rect1, 3);
    txt4.Text = "RowSpan is set to " + Grid.GetRowSpan(rect1).ToString();
}
```

下面是完成的结果！



另请参阅

- [Grid](#)
- [面板概述](#)

如何：在网格之间共享大小调整属性

项目 • 2023/02/06

此示例演示如何在 Grid 元素之间共享列和行的大小调整数据，以使大小调整保持一致。

示例

下面的示例引入了两个 Grid 元素作为父 DockPanel 的子元素。 Grid 的 IsSharedSizeScope 附加属性在父级 DockPanel 上定义。

该示例通过使用两个 Button 元素处理属性值；每个元素表示其中一个布尔属性值。当将 IsSharedSizeScope 属性值设置为 true 时，无论行或列的内容如何，SharedSizeGroup 的每个列或行成员都将共享大小调整信息。

XAML

```
<DockPanel Name="dp1" Grid.IsSharedSizeScope="False" VerticalAlignment="Top"
HorizontalAlignment="Left" Margin="10">
```

...

XAML

```
<StackPanel Orientation="Horizontal" DockPanel.Dock="Top">
    <Button Click="setTrue" Margin="0,0,10,10">Set
    IsSharedSizeScope="True" </Button>
    <Button Click="setFalse" Margin="0,0,10,10">Set
    IsSharedSizeScope="False" </Button>
</StackPanel>

<StackPanel Orientation="Horizontal" DockPanel.Dock="Top">

    <Grid ShowGridLines="True" Margin="0,0,10,0">
        <Grid.ColumnDefinitions>
            <ColumnDefinition SharedSizeGroup="FirstColumn"/>
            <ColumnDefinition SharedSizeGroup="SecondColumn"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" SharedSizeGroup="FirstRow"/>
        </Grid.RowDefinitions>

            <Rectangle Fill="Silver" Grid.Column="0" Grid.Row="0" Width="200"
Height="100"/>
            <Rectangle Fill="Blue" Grid.Column="1" Grid.Row="0" Width="150"
Height="100"/>

            <TextBlock Grid.Column="0" Grid.Row="0" FontWeight="Bold">First
```

```

Column</TextBlock>
    <TextBlock Grid.Column="1" Grid.Row="0" FontWeight="Bold">Second
Column</TextBlock>
</Grid>

<Grid ShowGridLines="True">
    <Grid.ColumnDefinitions>
        <ColumnDefinition SharedSizeGroup="FirstColumn"/>
        <ColumnDefinition SharedSizeGroup="SecondColumn"/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" SharedSizeGroup="FirstRow"/>
    </Grid.RowDefinitions>

        <Rectangle Fill="Silver" Grid.Column="0" Grid.Row="0"/>
        <Rectangle Fill="Blue" Grid.Column="1" Grid.Row="0"/>

        <TextBlock Grid.Column="0" Grid.Row="0" FontWeight="Bold">First
Column</TextBlock>
        <TextBlock Grid.Column="1" Grid.Row="0" FontWeight="Bold">Second
Column</TextBlock>
</Grid>

</StackPanel>

<TextBlock Margin="10" DockPanel.Dock="Top" Name="txt1"/>

```

以下代码隐藏示例处理按钮 Click 事件引发的方法。该示例将这些方法调用的结果写入 TextBlock 元素，这些元素使用相关的 get 方法以字符串形式输出新属性值。

C#

```

private void setTrue(object sender, System.Windows.RoutedEventArgs e)
{
    Grid.SetIsSharedSizeScope(dp1, true);
    txt1.Text = "IsSharedSizeScope Property is set to " +
    Grid.GetIsSharedSizeScope(dp1).ToString();
}

private void setFalse(object sender, System.Windows.RoutedEventArgs e)
{
    Grid.SetIsSharedSizeScope(dp1, false);
    txt1.Text = "IsSharedSizeScope Property is set to " +
    Grid.GetIsSharedSizeScope(dp1).ToString();
}

```

另请参阅

- [Grid](#)
- [IsSharedSizeScope](#)

- 面板概述

GridSplitter

项目 • 2023/02/06

GridSplitter 重新分布 Grid 控件的列间距或行间距。

本节内容

[操作指南主题](#)

参考

[GridSplitter](#)

相关章节

GridSplitter 帮助主题

项目 • 2023/02/06

本部分中的主题介绍如何使用 [GridSplitter](#) 控件。

本节内容

[使用 GridSplitter 重设行大小](#)

[使用 GridSplitter 重设列大小](#)

[确保 GridSplitter 可见](#)

参考

[GridSplitter](#)

[Grid](#)

相关章节

如何：使用 GridSplitter 调整行的大小

项目 • 2023/02/06

此示例演示如何使用水平 [GridSplitter](#)，在不更改 [Grid](#) 大小的情况下重新分配 [Grid](#) 中两行之间的空间。

示例

如何创建覆盖行边缘的 GridSplitter

若要指定重设 [Grid](#) 中相邻行大小的 [GridSplitter](#)，请将 [Row](#) 附加属性设置为要重设大小的行之一。如果 [Grid](#) 超过一列，请将 [ColumnSpan](#) 附加属性设置为指定列数。然后，将 [VerticalAlignment](#) 设置为 [Top](#) 或 [Bottom](#)（设置的对齐方式取决于要重设大小的两个行）。最后，将 [HorizontalAlignment](#) 属性设置为 [Stretch](#)。

下面的示例演示如何定义可重设相邻行大小的 [GridSplitter](#)。

XAML

```
<GridSplitter Grid.Row="1"
    Grid.ColumnSpan="3"
    HorizontalAlignment="Stretch"
    VerticalAlignment="Top"
    Background="Black"
    ShowsPreview="true"
    ResizeDirection="Rows"
    Height="5"/>
```

未占据自身行的 [GridSplitter](#) 可能会被 [Grid](#) 中的其他控件遮盖。有关如何避免此问题的详细信息，请参阅[确保 GridSplitter 可见](#)。

如何创建占据一行的 GridSplitter

若要指定在 [Grid](#) 中占据一行的 [GridSplitter](#)，请将 [Row](#) 附加属性设置为要重设大小的行之一。如果 [Grid](#) 超过一列，请将 [ColumnSpan](#) 附加属性设置为列数。然后，将 [VerticalAlignment](#) 设置为 [Center](#)，将 [HorizontalAlignment](#) 属性设置为 [Stretch](#)，并将包含 [GridSplitter](#) 的行的 [Height](#) 设置为 [Auto](#)。

下面的示例演示如何定义一个占据一行并可重设行任意一侧大小的水平 [GridSplitter](#)。

XAML

```
<Grid.RowDefinitions>
    <RowDefinition Height="50*" />
    <RowDefinition Height="Auto" />
```

```
<RowDefinition Height="50*" />
</Grid.RowDefinitions>
```

XAML

```
<StackPanel Grid.Row="0" Grid.Column="1" Background="Tan"/>
<GridSplitter Grid.Row="1"
    HorizontalAlignment="Stretch"
    VerticalAlignment="Center"
    Background="Black"
    ShowsPreview="True"
    Height="5"
/>
<StackPanel Grid.Row="2" Grid.Column="0" Background="Brown"/>
```

另请参阅

- [GridSplitter](#)
- [操作指南主题](#)

如何：使用 GridSplitter 调整列的大小

项目 • 2023/02/06

此示例演示如何创建垂直 GridSplitter，以便在不更改 Grid 大小的情况下重新分配 Grid 中两列之间的空间。

示例

如何创建覆盖列边缘的 GridSplitter

若要指定重设 Grid 中相邻列大小的 GridSplitter，请将 Column 附加属性设置为要重设大小的列之一。如果 Grid 超过一行，请将 RowSpan 附加属性设置为行数。然后将 HorizontalAlignment 属性设置为 Left 或 Right（设置的对齐方式取决于要重设大小的两列）。最后，将 VerticalAlignment 属性设置为 Stretch。

XAML

```
<GridSplitter Grid.Column="1"
    Grid.RowSpan="3"
    HorizontalAlignment="Left"
    VerticalAlignment="Stretch"
    Background="Black"
    ShowsPreview="true"
    Width="5"/>
```

没有自己的列的 GridSplitter 可能会被 Grid 中的其他控件遮挡。有关如何避免此问题的详细信息，请参阅[确保 GridSplitter 可见](#)。

如何创建占据一列的 GridSplitter

若要指定在 Grid 中占据一列的 GridSplitter，请将 Column 附加属性设置为要重设大小的列之一。如果 Grid 超过一行，请将 RowSpan 附加属性设置为行数。然后将 HorizontalAlignment 设置为 Center，将 VerticalAlignment 属性设置为 Stretch，并将包含 GridSplitter 的列的 Width 设置为 Auto。

下面的示例演示如何定义一个占据一列并可重设列任意一侧大小的垂直 GridSplitter。

XAML

```
<Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition/>
</Grid.ColumnDefinitions>
```

XAML

```
<GridSplitter Grid.Column="1"
    HorizontalAlignment="Center"
    VerticalAlignment="Stretch"
    Background="Black"
    ShowsPreview="True"
    Width="5"
    />
```

另请参阅

- [GridSplitter](#)
- [操作指南主题](#)

如何：确保 GridSplitter 可见

项目 • 2023/02/06

此示例演示如何确保 [GridSplitter](#) 控件不会被 [Grid](#) 中的其他控件隐藏。

示例

[Grid](#) 控件的 [Children](#) 按标记或代码中定义的顺序呈现。如果不将 [GridSplitter](#) 控件定义为 [Children](#) 集合中的最后一个元素，或者为其他控件指定了更高的 [ZIndexProperty](#)，那么其他控件可能会隐藏该控件。

为防止隐藏 [GridSplitter](#) 控件，请执行下列任一操作。

- 确保 [GridSplitter](#) 控件是添加到 [Grid](#) 的最后一个 [Children](#)。以下示例将 [GridSplitter](#) 显示为 [Grid](#) 的 [Children](#) 集合中的最后一个元素。

XAML

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Button Grid.Column="0"/>
    <GridSplitter Grid.Column ="0" Background="Blue"/>
</Grid>
```

- 将 [GridSplitter](#) 的 [ZIndexProperty](#) 设置为高于可能会隐藏它的控件。以下示例为 [GridSplitter](#) 控件提供高于 [Button](#) 控件的 [ZIndexProperty](#)。

XAML

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <GridSplitter Grid.Column="0" Background="Blue"
                  Panel.ZIndex="1"/>
    <Button Grid.Column="0"/>
</Grid>
```

- 对可能会隐藏 [GridSplitter](#) 的控件设置边距，以便显示 [GridSplitter](#)。以下示例对可能会覆盖和隐藏 [GridSplitter](#) 的控件设置边距。

XAML

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <GridSplitter Grid.Column ="0" Background="Blue"/>
    <Button Grid.Column="0" Margin="0,0,5,0"/>
</Grid>
```

另请参阅

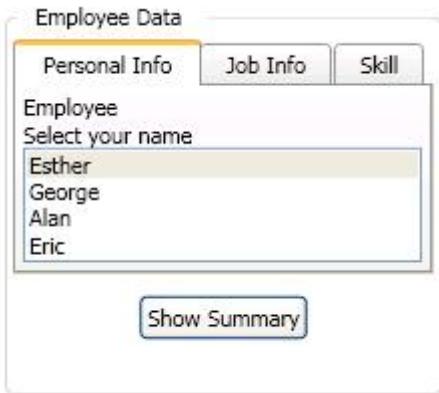
- [GridSplitter](#)
- [操作指南主题](#)

GroupBox

项目 • 2023/02/06

GroupBox 控件是一个 [HeaderedContentControl](#)，它为图形用户界面 (GUI) 内容提供标题容器。

下图显示了包含 [TabControl](#) 的 [GroupBox](#) 和包含在 [StackPanel](#) 中的 [Button](#)。



本节内容

[定义 GroupBox 模板](#)

参考

[GroupBox](#)

相关章节

如何：定义 GroupBox 模板

项目 • 2022/09/27

此示例演示如何为 [GroupBox](#) 控件创建模板。

示例

下面的示例通过使用布局的 [Grid](#) 控件来定义 [GroupBox](#) 控件模板。该模板使用 [BorderGapMaskConverter](#) 来定义 [GroupBox](#) 的边框，以便边框不会遮挡 [Header](#) 内容。

XAML

```
<!-- ===== GroupBox Template Example ===== -->
<BorderGapMaskConverter x:Key="BorderGapMaskConverter"/>
<Style x:Key="{x:Type GroupBox}"
       TargetType="{x:Type GroupBox}">
    <Setter Property="BorderBrush"
           Value="Gray"/>
    <Setter Property="Foreground"
           Value="White"/>
    <Setter Property="BorderThickness"
           Value="1"/>
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type GroupBox}">
                <Grid>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="4"/>
                        <ColumnDefinition Width="Auto"/>
                        <ColumnDefinition Width="*"/>
                        <ColumnDefinition Width="4"/>
                    </Grid.ColumnDefinitions>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto"/>
                        <RowDefinition Height="Auto"/>
                        <RowDefinition Height="*"/>
                        <RowDefinition Height="4"/>
                    </Grid.RowDefinitions>
                    <Border CornerRadius="4"
                           Grid.Row="1"
                           Grid.RowSpan="3"
                           Grid.Column="0"
                           Grid.ColumnSpan="4"
                           BorderThickness="{TemplateBinding BorderThickness}"
                           BorderBrush="Transparent"
                           Background="{TemplateBinding Background}"/>
                <!-- ContentPresenter for the header -->
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

```

<Border x:Name="Header"
        Padding="6,0,6,0"
        Grid.Row="0"
        Grid.RowSpan="2"
        Grid.Column="1">
    <ContentPresenter ContentSource="Header"
                      RecognizesAccessKey="True" />
</Border>
<!-- Primary content for GroupBox -->
<ContentPresenter Grid.Row="2"
                  Grid.Column="1"
                  Grid.ColumnSpan="2"
                  Margin="{TemplateBinding Padding}"/>
<Border CornerRadius="0"
        Grid.Row="1"
        Grid.RowSpan="3"
        Grid.ColumnSpan="4"
        BorderThickness="{TemplateBinding BorderThickness}"
        BorderBrush="{TemplateBinding BorderBrush}">
    <Border.OpacityMask>
        <MultiBinding Converter=
                        "{StaticResource BorderGapMaskConverter}"
                        ConverterParameter="6">
            <Binding ElementName="Header"
                    Path="ActualWidth"/>
            <Binding RelativeSource="{RelativeSource Self}"
                    Path="ActualWidth"/>
            <Binding RelativeSource="{RelativeSource Self}"
                    Path="ActualHeight"/>
        </MultiBinding>
    </Border.OpacityMask>
    </Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

另请参阅

- [GroupBox](#)
- [如何：创建 GroupBox](#)

映像

项目 • 2023/02/06

[Image](#) 元素用于显示 Windows Presentation Foundation (WPF) 应用程序中的位图图像。

本节内容

[操作指南主题](#)

参考

[Image](#)

[BitmapImage](#)

[BitmapSource](#)

请参阅

- [图像处理概述](#)
- [操作指南主题](#)

图像帮助主题

项目 • 2023/02/06

本部分中的主题介绍如何使用 [Image](#) 元素。

本节内容

[使用 Image 元素](#)

[将图像转换成灰度图像](#)

[裁剪图像](#)

[旋转图像](#)

参考

[Image](#)

[BitmapImage](#)

[BitmapSource](#)

请参阅

- [图像处理概述](#)
- [操作指南主题](#)

如何：使用 Image 元素

项目 • 2023/02/06

此示例演示如何通过使用 [Image](#) 元素在应用程序中包含图像。

定义图像

下面的示例演示如何呈现宽为 200 像素的图像。在此 Extensible Application Markup Language (XAML) 示例中，同时使用特性语法和属性标记语法来定义图像。有关特性语法和属性语法的详细信息，请参阅[依赖属性概述](#)。[BitmapImage](#) 用于定义图像的源数据，并且针对属性标记语法示例显式定义。此外，[BitmapImage](#) 中的 [DecodePixelWidth](#) 的宽度设置为与 [Image](#) 中的 [Width](#) 的宽度相同。这样做是为了确保呈现图像所使用的内存量最小。

① 备注

通常，如果你需要指定呈现图像的大小，只需指定 [Width](#) 或 [Height](#)，但请勿同时指定两者。如果仅指定一个，则会保持图像的纵横比。否则，图像可能会出现意外的拉伸或扭曲。若要控制图像的拉伸行为，请使用 [Stretch](#) 和 [StretchDirection](#) 属性。

② 备注

当使用 [Width](#) 或 [Height](#) 指定图像大小时，还应该将 [DecodePixelWidth](#) 或 [DecodePixelHeight](#) 设置为相同大小。

[Stretch](#) 属性确定如何拉伸图像源来填充图像元素。有关详细信息，请参见 [Stretch 枚举](#)。

XAML

```
<!-- Simple image rendering. However, rendering an image this way may not
     result in the best use of application memory. See markup below which
     creates the same end result but using less memory. -->
<Image Width="200"
Source="C:\Documents and Settings\All Users\Documents\My Pictures\Sample
Pictures\Water Lilies.jpg"/>

<Image Width="200">
  <Image.Source>
    <!-- To save significant application memory, set the DecodePixelWidth or
        DecodePixelHeight of the BitmapImage value of the image source to the
        desired
```

```
height and width of the rendered image. If you don't do this, the
application will
    cache the image as though it were rendered as its normal size rather
than just
        the size that is displayed. -->
    <!-- Note: In order to preserve aspect ratio, only set either
DecodePixelWidth
        or DecodePixelHeight but not both. -->
<BitmapImage DecodePixelWidth="200"
    UriSource="C:\Documents and Settings\All Users\Documents\My
Pictures\Sample Pictures\Water Lilies.jpg" />
</Image.Source>
</Image>
```

呈现图像

以下示例演示如何使用代码呈现宽度为 200 像素的图像。

① 备注

必须在 BeginInit 和 EndInit 块中完成 BitmapImage 属性的设置。

C#

```
// Create Image Element
Image myImage = new Image();
myImage.Width = 200;

// Create source
BitmapImage myBitmapImage = new BitmapImage();

// BitmapImage.UriSource must be in a BeginInit/EndInit block
myBitmapImage.BeginInit();
myBitmapImage.UriSource = new Uri(@"C:\Documents and Settings\All
Users\Documents\My Pictures\Sample Pictures\Water Lilies.jpg");

// To save significant application memory, set the DecodePixelWidth or
// DecodePixelHeight of the BitmapImage value of the image source to the
desired
// height or width of the rendered image. If you don't do this, the
application will
// cache the image as though it were rendered as its normal size rather than
just
// the size that is displayed.
// Note: In order to preserve aspect ratio, set DecodePixelWidth
// or DecodePixelHeight but not both.
myBitmapImage.DecodePixelWidth = 200;
myBitmapImage.EndInit();
```

```
//set image source  
myImage.Source = myBitmapImage;
```

另请参阅

- [图像处理概述](#)

如何：将图像转换为灰度

项目 • 2023/02/06

此示例演示如何使用 [FormatConvertedBitmap](#) 将图像转换为灰度。

示例

XAML

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

    <Page.Resources>

        <!-- This resource defines a BitmapImage with a source and a
            DecodePixelWidth of 200. This property is set to the same value
            as the desired width of the image to save on memory use. This
            BitmapImage is used as the base for the other BitmapSource
            resources. -->
        <BitmapImage x:Key="masterImage" DecodePixelWidth="200"
                     UriSource="C:\Documents and Settings\All Users\Documents\My
                     Pictures\Sample Pictures\Forest.jpg"/>

        <!-- This FormatConvertedBitmap uses the BitmapImage defined above and
            changes the format to Gray32Float (grayscale). -->
        <FormatConvertedBitmap x:Key="convertFormatImage"
                              Source="{StaticResource masterImage}"
                              DestinationFormat="Gray32Float" />

    </Page.Resources>

    <StackPanel>

        <!-- Apply the "convertFormatImage" resource defined above to this
            image. -->
        <Image Width="200" Source="{StaticResource convertFormatImage}" />
    </StackPanel>
</Page>
```

C#

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Media.Imaging;

namespace SDKSample
{
```

```
public partial class FormatConvertedBitmapExample : Page
{
    public FormatConvertedBitmapExample()
    {

        // //// Create a BitmapImage and set it's DecodePixelWidth to 200.
        Use     //////
            // //// this BitmapImage as a source for other BitmapSource
        objects.   /////

        BitmapImage myBitmapImage = new BitmapImage();

        // BitmapSource objects like BitmapImage can only have their
        properties
            // changed within a BeginInit/EndInit block.
        myBitmapImage.BeginInit();
        myBitmapImage.UriSource = new
        Uri(@"sampleImages/WaterLilies.jpg", UriKind.Relative);

            // To save significant application memory, set the
        DecodePixelWidth or
            // DecodePixelHeight of the BitmapImage value of the image
        source to the desired
            // height or width of the rendered image. If you don't do this,
        the application will
            // cache the image as though it were rendered as its normal size
        rather than just
            // the size that is displayed.
            // Note: In order to preserve aspect ratio, set DecodePixelWidth
            // or DecodePixelHeight but not both.
        myBitmapImage.DecodePixelWidth = 200;
        myBitmapImage.EndInit();

        /////////// Convert the BitmapSource to a new format ///////////
        // Use the BitmapImage created above as the source for a new
        BitmapSource object
            // which is set to a gray scale format using the
        FormatConvertedBitmap BitmapSource.
            // Note: New BitmapSource does not cache. It is always pulled
        when required.

        FormatConvertedBitmap newFormatedBitmapSource = new
        FormatConvertedBitmap();

        // BitmapSource objects like FormatConvertedBitmap can only have
        their properties
            // changed within a BeginInit/EndInit block.
        newFormatedBitmapSource.BeginInit();

            // Use the BitmapSource object defined above as the source for
        this new
            // BitmapSource (chain the BitmapSource objects together).
        newFormatedBitmapSource.Source = myBitmapImage;

            // Set the new format to Gray32Float (grayscale).
```

```
newFormatedBitmapSource.DestinationFormat =
PixelFormats.Gray32Float;
newFormatedBitmapSource.EndInit();

// Create Image Element
Image myImage = new Image();
myImage.Width = 200;
//set image source
myImage.Source = newFormatedBitmapSource;

// Add Image to the UI
StackPanel myStackPanel = new StackPanel();
myStackPanel.Children.Add(myImage);
this.Content = myStackPanel;
}
}

}
```

另请参阅

- [使用 Image 元素](#)
- [裁剪图像](#)
- [旋转图像](#)

如何：裁剪图像

项目 • 2022/09/27

此示例演示如何使用 [CroppedBitmap](#) 裁剪图像。

[CroppedBitmap](#) 主要用于对图像的裁剪版本进行编码以保存到文件中。 若要裁剪图像以进行显示，请参阅[如何：创建剪裁区域](#)主题。

示例

以下 Extensible Application Markup Language (XAML) 定义以下示例中使用的资源。

XAML

```
<Page.Resources>
    <!-- Define some image resources, for use as the image element source. --
    >
    <BitmapImage x:Key="masterImage" UriSource="/sampleImages/gecko.jpg" />
    <CroppedBitmap x:Key="croppedImage"
        Source="{StaticResource masterImage}" SourceRect="30 20 105 50"/>
</Page.Resources>
```

以下示例使用 [CroppedBitmap](#) 作为源创建一个图像。

XAML

```
<!-- Use the cropped image resource as the images source -->
<Image Width="200" Source="{StaticResource croppedImage}"
    Margin="5" Grid.Column="0" Grid.Row="1" />
```

C#

```
// Create an Image element.
Image croppedImage = new Image();
croppedImage.Width = 200;
croppedImage.Margin = new Thickness(5);

// Create a CroppedBitmap based off of a xaml defined resource.
CroppedBitmap cb = new CroppedBitmap(
    (BitmapSource)this.Resources["masterImage"],
    new Int32Rect(30, 20, 105, 50));           //select region rect
croppedImage.Source = cb;                   //set image source to cropped
```

[CroppedBitmap](#) 也可以用作另一个 [CroppedBitmap](#) 的源，链接裁剪。 注意，[SourceRect](#) 使用相对于源裁剪位图的值，而不是初始图像。

XAML

```
<!-- Chain a cropped bitmap off a previously defined cropped image -->
<Image Width="200" Grid.Column="0" Grid.Row="3" Margin="5">
    <Image.Source>
        <CroppedBitmap Source="{StaticResource croppedImage}"
            SourceRect="30 0 75 50"/>
    </Image.Source>
</Image>
```

C#

```
// Create an Image element.
Image chainImage = new Image();
chainImage.Width = 200;
chainImage.Margin = new Thickness(5);

// Create the cropped image based on previous CroppedBitmap.
CroppedBitmap chained = new CroppedBitmap(cb,
    new Int32Rect(30, 0, (int)cb.Width-30, (int)cb.Height));
// Set the image's source.
chainImage.Source = chained;
```

另请参阅

- [如何：创建剪裁区域](#)

如何：旋转图像

项目 • 2023/02/06

此示例演示如何使用 [BitmapImage](#) 的 [Rotation](#) 属性将图像旋转 90 度。

示例

XAML

```
<Image Width="150" Margin="5" Grid.Column="0" Grid.Row="1">
    <Image.Source>
        <BitmapImage UriSource="/sampleImages/watermelon.jpg"
Rotation="Rotate90" />
    </Image.Source>
</Image>
```

C#

```
//Create Image element
Image rotated270 = new Image();
rotated270.Width = 150;

//Create source
BitmapImage bi = new BitmapImage();
//BitmapImage properties must be in a BeginInit/EndInit block
bi.BeginInit();
bi.UriSource = new
Uri(@"pack://application:,,/sampleImages/watermelon.jpg");
//Set image rotation
bi.Rotation = Rotation.Rotate270;
bi.EndInit();
//set image source
rotated270.Source = bi;
```

Label

项目 • 2022/09/27

[Label](#) 控件通常提供用户界面中的信息 (UI)。在过去，一个 [Label](#) 只包含文本，但由于随 Windows Presentation Foundation(WPF) 一起提供的 [Label](#) 是一个 [ContentControl](#)，因此它可以包含文本或 [UIElement](#)。

[Label](#) 为访问键提供功能和视觉方面的支持。它通常用于启用对控件的快速键盘访问，如 [TextBox](#)。若要将 [Label](#) 分配到 [Control](#)，请设置 [Label.Target](#) 属性为当用户按下访问键时应获取焦点的控件。

下图显示面向 [ComboBox](#) 的 [Label](#)“主题”。当用户按 [T](#) 时，[ComboBox](#) 会接收焦点。有关详细信息，请参见[如何：设置 Label 的目标属性](#)。



本节内容

[如何：创建具有访问键和文本换行的控件](#)

参考

[Label](#)

如何：创建具有访问键和文本换行的控件

项目 • 2022/09/27

此示例演示如何创建具有访问键且支持文本换行的控件。该示例使用 [Label](#) 控件来阐述这些概念。

示例

将文本换行添加到标签

[Label](#) 控件不支持文本换行。如果需要一个多次换行的标签，可以嵌套其他支持文本换行的元素，并将该元素放在标签内。下面的示例演示如何使用 [TextBlock](#) 创建一个进行多次文本换行的标签。

XAML

```
<Label Width="200" HorizontalAlignment="Left">
    <TextBlock TextWrapping="WrapWithOverflow">
        A long piece of text that requires text wrapping
        goes here.
    </TextBlock>
</Label>
```

将访问键和文本换行添加到标签

如果需要具有访问键（助记键）的 [Label](#)，请使用 [Label](#) 内的 [AccessText](#) 元素。

控件，例如 [Label](#)、[Button](#)、[RadioButton](#)、[CheckBox](#)、[MenuItem](#)、[TabItem](#)、[Expander](#) 以及 [GroupBox](#) 具有默认控件模板。这些模板包含一个 [ContentPresenter](#)。可在 [ContentPresenter](#) 上设置的属性之一是 `RecognizesAccessKey="true"`，可使用该属性为控件指定访问键。

以下示例演示如何创建具有访问键且支持文本换行的 [Label](#)。若要实现文本换行，本示例将设置 [TextWrapping](#) 属性并使用下划线字符指定访问键。（紧跟下划线字符后面的字符就是访问键。）

XAML

```
<TextBox Name="textBox1" Width="50" Height="20"/>
<Label Width="200" HorizontalAlignment="Left"
    Target="{Binding ElementName(textBox1)}">
    <AccessText TextWrapping="WrapWithOverflow">
        _Another long piece of text that requires text wrapping
        goes here.
    </AccessText>
</Label>
```

```
</AccessText>  
</Label>
```

另请参阅

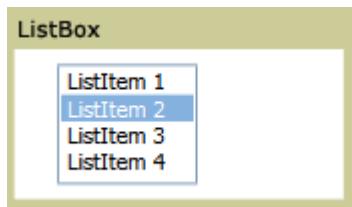
- 如何：设置 Label 的目标属性

ListBox

项目 • 2023/02/06

[ListBox](#) 控件为用户提供可选项列表。

下图说明一个典型的 [ListBox](#)。



典型的 [ListBox](#)

本节内容

[操作指南主题](#)

参考

[ListBox](#)

[ListBoxItem](#)

相关章节

ListBox 帮助主题

项目 • 2022/09/27

本节中的主题介绍如何使用 [ListBox](#) 控件显示可选择的项列表。

本节内容

- [将 ListBox 绑定到数据](#)
- [获取 ListBoxItem](#)
- [如何：向 ItemsControl 添加数据](#)
- [提升 ListBox 的滚动性能](#)

参考

[ListBox](#)

[ListBoxItem](#)

相关章节

如何：将 ListBox 绑定到数据

项目 • 2023/02/06

应用程序开发人员可以创建 [ListBox](#) 控件，而无需分别指定每个 [ListBoxItem](#) 的内容。 可以使用数据绑定将数据绑定到各个项。

下面的示例演示如何创建一个 [ListBox](#)，它通过数据绑定到名为 *Colors* 的数据源来填充 [ListBoxItem](#) 元素。 在这种情况下，没有必要使用 [ListBoxItem](#) 标记来指定每个项的内容。

示例

XAML

```
<Canvas.Resources>
    <src:myColors x:Key="Colors"/>
</Canvas.Resources>
```

XAML

```
<ListBox Name="myListBox" HorizontalAlignment="Left"
SelectionMode="Extended"
Width="265" Height="55" Background="HoneyDew"
SelectionChanged="myListBox_SelectionChanged"
ItemsSource="{Binding Source={StaticResource Colors}}"
IsSynchronizedWithCurrentItem="true">
</ListBox>
```

另请参阅

- [ListBox](#)
- [ListBoxItem](#)
- [控件](#)

如何：获取 ListBoxItem

项目 • 2023/02/06

如果需要在 [ListBox](#) 中的特定索引处获取特定 [ListBoxItem](#)，可以使用 [ItemContainerGenerator](#)。

示例

以下示例显示了 [ListBox](#) 及其项。

XAML

```
<ListBox Margin="10,0,0,5" Name="lb" VerticalAlignment="Top" Grid.Column="0"
Grid.Row="2">
    <ListBoxItem>Item 0</ListBoxItem>
    <ListBoxItem>Item 1</ListBoxItem>
    <ListBoxItem>Item 2</ListBoxItem>
    <ListBoxItem>Item 3</ListBoxItem>
</ListBox>
```

以下示例演示如何通过在 [ItemContainerGenerator](#) 的 [ContainerFromIndex](#) 属性中指定项的索引来检索该项。

C#

```
private void GetIndex0(object sender, RoutedEventArgs e)
{
    ListBoxItem lbi = (ListBoxItem)
        (lb.ItemContainerGenerator.ContainerFromIndex(0));
    Item.Content = "The contents of the item at index 0 are: " +
        (lbi.Content.ToString()) + ".";
}
```

检索到列表框项后，可以显示该项的内容，如下例所示。

C#

```
Item.Content = "The contents of the item at index 0 are: " +
    (lbi.Content.ToString()) + ".";
```

如何：提高 ListBox 的滚动性能

项目 • 2023/02/06

如果 [ListBox](#) 包含许多项，则当用户使用鼠标滚轮或拖动滚动条的滚动块滚动 [ListBox](#) 时，用户界面响应可能会很慢。可以通过将

`VirtualizingStackPanel.VirtualizationMode` 附加属性设置为 `VirtualizationMode.Recycling` 来提高用户滚动时 [ListBox](#) 的性能。

示例

描述

以下示例创建 [ListBox](#) 并将 `VirtualizingStackPanel.VirtualizationMode` 附加属性设置为 `VirtualizationMode.Recycling` 以提高滚动期间的性能。

代码

XAML

```
<StackPanel>

    <StackPanel.Resources>
        <src:LotsOfItems x:Key="data"/>
    </StackPanel.Resources>

    <ListBox Height="150" ItemsSource="{StaticResource data}"
        VirtualizingStackPanel.VirtualizationMode="Recycling" />

</StackPanel>
```

以下示例显示了上一个示例所使用的数据。

C#

```
public class LotsOfItems : ObservableCollection<String>
{
    public LotsOfItems()
    {
        for (int i = 0; i < 1000; ++i)
        {
            Add("item " + i.ToString());
        }
    }
}
```

```
    }  
}
```

ListView

项目 • 2023/02/06

[ListView](#) 控件提供了用于在不同布局或视图中显示一组数据项的基础结构。

下图显示了一个 [ListView](#)。

First Name	Last Name	Employee No.
Jesper	Aaberg	12345
Dominik	Paiha	98765
Yale	Li	23875
Muru	Subramani	49392

本节内容

[概述](#)

[操作指南主题](#)

参考

[ListView](#)

[ListViewItem](#)

[GridView](#)

相关章节

[数据绑定概述](#)

[数据模板化概述](#)

ListView 概述

项目 • 2023/02/06

本节中的主题将介绍如何使用 [ListView](#) 控件。

本节内容

[ListView 概述](#)

[GridView 概述](#)

[GridView 列标题的样式和模板概述](#)

参考

[ListView](#)

[GridView](#)

相关章节

[操作指南主题](#)

ListView 概述

项目 • 2023/02/06

[ListView](#) 控件提供了用于在不同布局或视图中显示一组数据项的基础结构。例如，用户可能需要在表格中显示数据项，并同时对表格的列进行排序。

① 备注

你可在[代码参考](#)部分中找到本文中引用的类型。

什么是 ListView？

[ListView](#) 控件是派生自 [ListBox](#) 的 [ItemsControl](#) 控件。通常，该控件的项为数据集合的成员，并且表示为 [ListViewItem](#) 对象。[ListViewItem](#) 是一个 [ContentControl](#) 且只能包含单个子元素。但是，该子元素可以是任何视觉元素。

为 ListView 定义视图模式

若要为 [ListView](#) 控件的内容指定视图模式，请设置 [View](#) 属性。Windows Presentation Foundation (WPF) 提供的一个视图模式为 [GridView](#)，可在具有可自定义列的表格中显示数据项集合。

下面的示例演示如何为显示员工信息的 [ListView](#) 控件定义 [GridView](#)。

XAML

```
<ListView ItemsSource="{Binding Source={StaticResource EmployeeInfoDataSource}}">

    <ListView.View>

        <GridView AllowsColumnReorder="true" ColumnHeaderToolTip="Employee Information">

            <GridViewColumn DisplayMemberBinding="{Binding Path=FirstName}" Header="First Name" Width="100"/>

            <GridViewColumn DisplayMemberBinding="{Binding Path=LastName}" Width="100">
                <GridViewColumnHeader>Last Name
                <GridViewColumnHeader.ContextMenu>
                    <ContextMenu MenuItem.Click="LastNameCM_Click" Name="LastNameCM">
                
```

```

        <MenuItem Header="Ascending" />
        <MenuItem Header="Descending" />
    </ContextMenu>
</GridViewColumnHeader>
</GridViewColumnHeader>
</GridViewColumn>

<GridViewColumn DisplayMemberBinding="{Binding Path=EmployeeNumber}" Header="Employee No." Width="100"/>
</GridView>

</ListView.View>
</ListView>

```

下图演示上一个示例的数据显示方式。

First Name	Last Name	Employee No.
Jesper	Aaberg	12345
Dominik	Paiha	98765
Yale	Li	23875
Muru	Subramani	49392

通过定义继承自 [ViewBase](#) 类的类，可创建自定义视图模式。[ViewBase](#) 类提供了创建自定义视图时所需的基础结构。有关如何创建自定义视图的详细信息，请参阅[为 ListView 创建自定义视图模式](#)。

将数据绑定到 ListView

使用 [Items](#) 和 [ItemsSource](#) 属性为 [ListView](#) 控件指定项。下面的示例将 [ItemsSource](#) 属性设置为名为 [EmployeeInfoDataSource](#) 的数据集合。

XAML

```
<ListView ItemsSource="{Binding Source={StaticResource EmployeeInfoDataSource}}">
```

在 [GridView](#) 中，[GridViewColumn](#) 对象绑定到指定的数据字段。以下示例通过为 [DisplayMemberBinding](#) 属性指定 [Binding](#)，将 [GridViewColumn](#) 对象绑定到数据字段。

C#

```
GridViewColumn gvc1 = new GridViewColumn();
gvc1.DisplayMemberBinding = new Binding("FirstName");
gvc1.Header = "FirstName";
gvc1.Width = 100;
```

XAML

```
<GridViewColumn DisplayMemberBinding="{Binding Path=FirstName}"  
Header="First Name" Width="100"/>
```

还可将 [Binding](#) 指定为 [DataTemplate](#) 定义的一部分，该定义可用于对列中的单元格设置样式。在下面的示例中，用 [ResourceKey](#) 标识的 [DataTemplate](#) 为 [GridViewColumn](#) 设置 [Binding](#)。请注意，此示例未定义 [DisplayMemberBinding](#)，因为这样做会替代由 [DataTemplate](#) 指定的绑定。

XAML

```
<DataTemplate x:Key="myCellTemplateMonth">  
    <DockPanel>  
        <TextBlock Foreground="DarkBlue" HorizontalAlignment="Center">  
            <TextBlock.Text>  
                <Binding Path="Month"/>  
            </TextBlock.Text>  
        </TextBlock>  
    </DockPanel>  
</DataTemplate>
```

XAML

```
<GridViewColumn Header="Month" Width="80"  
    CellTemplate="{StaticResource myCellTemplateMonth}"/>
```

为实现 GridView 的 ListView 设置样式

[ListView](#) 控件包含 [ListViewItem](#) 对象，这些对象表示显示的数据项。可使用以下属性定义数据项的内容和样式：

- 在 [ListView](#) 控件上，使用 [ItemTemplate](#)、[ItemTemplateSelector](#) 和 [ItemContainerStyle](#) 属性。
- 在 [ListViewItem](#) 控件上，使用 [ContentTemplate](#) 和 [ContentTemplateSelector](#) 属性。

为避免 [GridView](#) 中的单元格之间出现对齐问题，不要使用 [ItemContainerStyle](#) 设置属性，或添加对 [ListView](#) 中项的宽度有影响的内容。例如，当在 [ItemContainerStyle](#) 中设置 [Margin](#) 属性时，可能会出现对齐问题。若要指定属性或定义对 [GridView](#) 中项的宽度有影响的内容，请使用 [GridView](#) 类及其相关类（如 [GridViewColumn](#)）的属性。

有关如何使用 及其支持类的详细信息，请参阅 [GridView](#)[GridView](#) 概述。

如果为 [ListView](#) 控件定义 [ItemContainerStyle](#)，并同时定义 [ItemTemplate](#)，必须在样式中包含 [ContentPresenter](#) 才能使 [ItemTemplate](#) 正常工作。

对于使用 [GridView](#) 显示的 [ListView](#) 内容，请不要使用 [HorizontalContentAlignment](#) 和 [VerticalContentAlignment](#) 属性。 若要指定 [GridView](#) 的列中内容的对齐方式，请定义 [CellTemplate](#)。

共享同一视图模式

两个 [ListView](#) 控件无法同时共享同一视图模式。 如果尝试将同一视图模式用于多个 [ListView](#) 控件，则会发生异常。

若要指定可同时被由多个 [ListView](#) 使用的视图模式，请使用模板或样式。

创建自定义视图模式

诸如 [GridView](#) 的自定义视图派生自 [ViewBase](#) 抽象类，该类提供了一些工具，用于显示表示为 [ListViewItem](#) 对象的数据项。

代码参考

本文引用了以下对象：

- [EmployeeInfoDataSource](#) 数据收集。如果使用的是 Visual Basic .NET，[Window](#) 元素的声明方式将与你在示例代码中看到的略微不同：

XAML

```
<Window x:Class="SDKSample.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Loaded="OnLoad">
    xmlns:ds="clr-namespace:SDKSample"
    <Window.Resources>
        <ObjectDataProvider x:Key="EmployeeInfoDataSource" ObjectType="
{x:Type ds:myEmployees}" />
    </Window.Resources>
```

- [EmployeeInfo](#) 类，用作 [EmployeeInfoDataSource](#) 数据集合的类型。

C#

```
public class EmployeeInfo
{
    private string _firstName;
    private string _lastName;
```

```
private string _employeeNumber;

public string FirstName
{
    get {return _firstName;}
    set {_firstName = value;}
}

public string LastName
{
    get {return _lastName;}
    set {_lastName = value;}
}

public string EmployeeNumber
{
    get {return _employeeNumber;}
    set {_employeeNumber = value;}
}

public EmployeeInfo(string firstname, string lastname, string empnumber)
{
    _firstName = firstname;
    _lastName = lastname;
    _employeeNumber = empnumber;
}
```

另请参阅

- [GridView](#)
- [ListView](#)
- [ListViewItem](#)
- [Binding](#)
- [GridView 概述](#)
- [操作指南主题](#)
- [控件](#)

GridView 概述

项目 • 2023/02/06

GridView 视图模式是 ListView 控件的视图模式之一。通过 GridView 类及其支持类，你以及你的用户可以查看以表格形式呈现的项集合，该表格通常使用按钮作为交互式列标题。本主题介绍 GridView 类并概述其用途。

什么是 GridView 视图？

GridView 视图模式通过将数据字段绑定到列以及显示用于标识字段的列标题来显示数据项列表。默认的 GridView 样式将按钮作为列标题实现。通过对列标题使用按钮，可以实现重要的用户交互功能；例如，用户可以单击列标题来根据特定列的内容对 GridView 数据进行排序。

① 备注

GridView 用于列标题的按钮控件派生自 ButtonBase。

下图展示了 ListView 内容的 GridView 视图。

Name	Time	Artist	Disk
Song1	3:54	Singer1	Disk1
Song2	4:31	Singer2	Disk3
Recommended			
Song3	5:06	Singer3	Disk1
Strongly Recommended			
Song4	4:18	Singer3	Disk2
Song5	6:15	Singer1	Disk3
Strongly Recommended			

GridView 列由 GridViewColumn 对象表示，这些对象可根据其内容自动调整大小。也可以选择将 GridViewColumn 显式设置为一个特定宽度。可通过拖动列标题之间的手柄重设列的大小。还可动态地添加、移除、替换列以及对列重新排序，因为此功能内置于 GridView 中。但是，GridView 无法直接更新它显示的数据。

下面的示例演示如何定义显示员工数据的 GridView。在此示例中，ListView 将 EmployeeInfoDataSource 定义为 ItemsSource。DisplayMemberBinding 的属性定义将 GridViewColumn 内容绑定到 EmployeeInfoDataSource 数据类别。

XAML

```
<ListView ItemsSource="{Binding Source={StaticResource EmployeeInfoDataSource}}">
```

```

<ListView.View>

    <GridView AllowsColumnReorder="true" ColumnHeaderToolTip="Employee
Information">

        <GridViewColumn DisplayMemberBinding="{Binding Path=FirstName}"
Header="First Name" Width="100"/>

        <GridViewColumn DisplayMemberBinding="{Binding Path=LastName}"
Width="100">
            <GridViewColumnHeader>Last Name
                <GridViewColumnHeader.ContextMenu>
                    <ContextMenu MenuItem.Click="LastNameCM_Click"
Name="LastNameCM">
                        <MenuItem Header="Ascending" />
                        <MenuItem Header="Descending" />
                    </ContextMenu>
                </GridViewColumnHeader.ContextMenu>
            </GridViewColumnHeader>
        </GridViewColumn>

        <GridViewColumn DisplayMemberBinding="{Binding
Path=EmployeeNumber}" Header="Employee No." Width="100"/>
    </GridView>

    </ListView.View>
</ListView>

```

下图显示了上一示例创建的表格。 GridView 控件显示 ItemsSource 对象中的数据：

First Name	Last Name	Employee No.
Jesper	Aaberg	12345
Dominik	Paiha	98765
Yale	Li	23875
Muru	Subramani	49392

GridView 布局和样式

GridViewColumn 的列单元格和列标题具有相同宽度。 默认情况下，每一列都根据其内容来调整宽度大小。 也可以选择将列设置为固定宽度。

相关的内容显示在水平行中。 例如，在上图中，每个员工的姓氏、名字和 ID 号显示为一组，因为它们显示在一个水平行中。

在 GridView 中定义列并设置其样式

定义要显示在 GridViewColumn 中的数据字段时，请使用 DisplayMemberBinding、CellTemplate 或 CellTemplateSelector 属性。 DisplayMemberBinding 属性优先于任何模板属性。

若要指定 [GridView](#) 的列中内容的对齐方式，请定义 [CellTemplate](#)。对于使用 [GridView](#) 显示的 [ListView](#) 内容，请不要使用 [HorizontalContentAlignment](#) 和 [VerticalContentAlignment](#) 属性。

若要指定列标题的模板和样式属性，请使用 [GridView](#)、[GridViewColumn](#) 和 [GridViewColumnHeader](#) 类。有关详细信息，请参阅 [GridView 列标题的样式和模板概述](#)。

将可视元素添加到 GridView

若要将可视元素（例如 [CheckBox](#) 和 [Button](#) 控件）添加到 [GridView](#) 视图模式，请使用模板或样式。

如果将可视元素显式定义为数据项，则它只能在 [GridView](#) 中出现一次。之所以存在此限制，是因为元素只能有一个父级，因此，只能在可视化树中出现一次。

设置 GridView 中的行的样式

使用 [GridViewRowPresenter](#) 和 [GridViewHeaderRowPresenter](#) 类格式化并显示 [GridView](#) 的行。有关如何设置 [GridView](#) 视图模式中行的样式的示例，请参阅[为实现 GridView 的 ListView 中的行设置样式](#)。

使用 ItemContainerStyle 时的对齐问题

为避免列标题和单元格之间出现对齐问题，不要在 [ItemContainerStyle](#) 中设置影响项宽度的属性或者指定这样的模板。例如，请勿设置 [Margin](#) 属性或指定将 [CheckBox](#) 添加到在 [ListView](#) 控件上定义的 [ItemContainerStyle](#) 的 [ControlTemplate](#)。而应直接在定义 [GridView](#) 视图模式的类上指定影响列宽度的属性和模板。

例如，若要向 [GridView](#) 视图模式中的行添加 [CheckBox](#)，请将 [CheckBox](#) 添加到 [DataTemplate](#)，然后将 [CellTemplate](#) 属性设置为该 [DataTemplate](#)。

与 GridView 的用户交互

当在应用程序中使用 [GridView](#) 时，用户可与 [GridView](#) 交互并修改它的格式。例如，用户可以对列重新排序、重设列的大小、选择表中的项以及滚动查看内容。还可定义当用户单击列标题按钮时响应的事件处理程序。事件处理程序可以执行一些操作，例如，根据一列的内容对 [GridView](#) 中显示的数据进行排序。

下表更详细地讨论了使用 [GridView](#) 进行用户交互的能力：

- **使用拖放方法对列重新排序。**

用户可以通过在列标题上按鼠标左键，然后将该列拖动到新的位置，对 [GridView](#) 中的列重新排序。当用户拖动列标题时，将显示标题的浮动版本以及显示列的插入位置的黑色实线。

如果想要修改标题的浮动版本的默认样式，请为 [GridViewColumnHeader](#) 类型指定一个 [ControlTemplate](#)，当 [Role](#) 属性设置为 [Floating](#) 时会触发该类型。有关详细信息，请参阅[为拖动的 GridView 列标题创建样式](#)。

- **根据列的内容重设其大小。**

用户可双击列标题右侧的手柄来根据列的内容重设其大小。

 **备注**

将 [Width](#) 属性设置为 `Double.NaN` 可以产生同样的效果。

- **选择行项目。**

用户可以选择 [GridView](#) 中的一个或多个项。

如果想要更改选定项的 [Style](#)，请参阅[使用触发器为 ListView 中的选定项设置样式](#)。

- **滚动查看最初未显示在屏幕上的内容。**

如果 [GridView](#) 的大小不足以显示所有项，用户可使用 [ScrollViewer](#) 控件提供的滚动条水平或垂直滚动查看。如果内容在某个特定方向全部可见，则 [ScrollBar](#) 将隐藏。列标题不会随着垂直滚动条滚动，但可水平滚动。

- **通过单击列标题按钮与列交互。**

如果提供了排序算法，则当用户单击列标题按钮时，可以对列中显示的数据进行排序。

为提供类似排序算法的功能，可处理列标题按钮的 [Click](#) 事件。若要处理单个列标题的 [Click](#) 事件，请在 [GridViewColumnHeader](#) 上设置事件处理程序。若要设置处理所有列标题的 [Click](#) 事件的事件处理程序，请在 [ListView](#) 控件上设置处理程序。

获取其他自定义视图

从 [ViewBase](#) 抽象类派生的 [GridView](#) 类仅仅是 [ListView](#) 类的可能视图模式之一。可通过从 [ViewBase](#) 类派生来为 [ListView](#) 创建其他自定义视图。有关自定义视图模式的示例，请参阅[为 ListView 创建自定义视图模式](#)。

GridView 支持类

以下类支持 [GridView](#) 视图模式。

- [GridViewColumn](#)
- [GridViewColumnHeader](#)
- [GridViewRowPresenter](#)
- [GridViewHeaderRowPresenter](#)
- [GridViewColumnCollection](#)
- [GridViewColumnHeaderRole](#)

另请参阅

- [ListView](#)
- [ListViewItem](#)
- [GridViewColumn](#)
- [GridViewColumnHeader](#)
- [GridViewRowPresenter](#)
- [GridViewHeaderRowPresenter](#)
- [ViewBase](#)
- [ListView 概述](#)
- [在标题获得单击时对 GridView 列进行排序](#)
- [操作指南主题](#)

GridView 列标题的样式和模板概述

项目 · 2023/02/06

本概述讨论在 [GridView](#) 控件的视图模式下用于自定义列标题的属性的优先顺序 [ListView](#)。

自定义 GridView 中的列标题

定义 中列标题 [GridView](#) 的内容、布局和样式的属性可以在许多相关类上找到。其中一些属性具有类似或相同的功能。

下表中的行显示执行相同函数的属性组。可以使用这些属性自定义 中的列标题 [GridView](#)。相关属性的优先顺序为从右到左的顺序，其中最右边列中的属性具有最高优先级。例如，如果在 对象[ContentTemplate](#)上设置了 [GridViewColumnHeader](#)，[HeaderTemplateSelector](#)[GridViewColumn](#)并且在关联的 上设置了 ，[ContentTemplate](#)则优先。在这种情况下， [HeaderTemplateSelector](#) 不起作用。

GridView 中列标题的相关属性

	GridView	GridViewColumn	GridViewColumnHeader
上下文菜单属性	ColumnHeaderContextMenu	不适用	ContextMenu
ToolTip	ColumnHeaderToolTip	不适用	ToolTip
属性			
标头模板属性	ColumnHeaderTemplate ^{1/} ColumnHeaderTemplateSelector	HeaderTemplate ^{1/} HeaderTemplateSelector	ContentTemplate ^{1/} ContentTemplateSelector
样式属性	ColumnHeaderContainerStyle	HeaderContainerStyle	Style

¹对于 "标头模板属性"，如果同时设置模板和模板选择器属性，则模板属性优先。例如，如果同时设置 和 [ContentTemplateContentTemplateSelector](#) 属性， [ContentTemplate](#) 则属性优先。

另请参阅

- 操作指南主题
- ListView 概述
- GridView 概述

ListView 帮助主题

项目 • 2023/02/06

本节中的主题介绍如何使用 [ListView](#) 控件显示一组数据项。

本节内容

[在标题获得单击时对 GridView 列进行排序](#)

[创建 ListView 的自定义视图模式](#)

[使用模板来设置使用 GridView 的 ListView 的样式](#)

[创建拖动的 GridView 列标头的样式](#)

[使用 GridView 显示 ListView 内容](#)

[使用触发器在 ListView 中设置选定项的样式](#)

[使用 CheckBox 创建 ListViewItem](#)

[使用 GridViewRowPresenter 显示数据](#)

[在实现 GridView 的 ListView 中对项进行分组](#)

[在实现 GridView 的 ListView 中设置行样式](#)

[更改 ListView 中列的水平对齐方式](#)

[处理 ListView 中每一项的 MouseDoubleClick 事件](#)

参考

[ListView](#)

[ListViewItem](#)

[GridView](#)

相关章节

[ListView 概述](#)

如何：在单击标题时对 GridView 列进行排序

项目 • 2022/09/27

此示例演示如何创建一个 [ListView](#) 控件，该控件可实现 [GridView](#) 视图模式，并在用户单击列标题时对数据内容进行排序。

示例

下面的示例定义了一个包含三个列的 [GridView](#)，这些列绑定到 [DateTime](#) 结构的 [Year](#)、[Month](#) 和 [Day](#) 属性。

XAML

```
<GridView>
    <GridViewColumn DisplayMemberBinding="{Binding Path=Year}"
        Header="Year"
        Width="100"/>
    <GridViewColumn DisplayMemberBinding="{Binding Path=Month}"
        Header="Month"
        Width="100"/>
    <GridViewColumn DisplayMemberBinding="{Binding Path=Day}"
        Header="Day"
        Width="100"/>
</GridView>
```

下面的示例演示定义为 [DateTime](#) 对象的 [ArrayList](#) 的数据项。[ArrayList](#) 被定义为 [ListView](#) 控件的 [ItemsSource](#)。

XAML

```
<ListView.ItemsSource>
    <s:ArrayList>
        <p:DateTime>1993/1/1 12:22:02</p:DateTime>
        <p:DateTime>1993/1/2 13:2:01</p:DateTime>
        <p:DateTime>1997/1/3 2:1:6</p:DateTime>
        <p:DateTime>1997/1/4 13:6:55</p:DateTime>
        <p:DateTime>1999/2/1 12:22:02</p:DateTime>
        <p:DateTime>1998/2/2 13:2:01</p:DateTime>
        <p:DateTime>2000/2/3 2:1:6</p:DateTime>
        <p:DateTime>2002/2/4 13:6:55</p:DateTime>
        <p:DateTime>2001/3/1 12:22:02</p:DateTime>
        <p:DateTime>2006/3/2 13:2:01</p:DateTime>
        <p:DateTime>2004/3/3 2:1:6</p:DateTime>
        <p:DateTime>2004/3/4 13:6:55</p:DateTime>
```

```
</s:ArrayList>
</ListView.ItemsSource>
```

XAML 标记中的 `s` 和 `p` 标识符引用在 XAML 页面元数据中定义的命名空间映射。下面的示例显示了元数据定义。

XAML

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="ListViewSort.Window1"
    xmlns:s="clr-namespace:System.Collections;assembly=mscorlib"
    xmlns:p="clr-namespace:System;assembly=mscorlib">
```

为了根据列的内容对数据进行排序，该示例定义了一个事件处理程序，用于处理当按下列标题按钮时发生的 `Click` 事件。以下示例显示如何为 `GridViewColumnHeader` 控件指定事件处理程序。

XAML

```
<ListView x:Name='lv' Height="150" HorizontalAlignment="Center"
    VerticalAlignment="Center"
    GridViewColumnHeader.Click="GridViewColumnHeaderClickedHandler"
    >
```

该示例定义了事件处理程序，以便每次按下列标题按钮时，排序方向会在升序和降序之间发生变化。下面的示例显示了事件处理程序。

C#

```
public partial class Window1 : Window
{
    public Window1()
    {
        InitializeComponent();
    }

    GridViewColumnHeader _lastHeaderClicked = null;
    ListSortDirection _lastDirection = ListSortDirection.Ascending;

    void GridViewColumnHeaderClickedHandler(object sender,
                                              RoutedEventArgs e)
    {
        var headerClicked = e.OriginalSource as GridViewColumnHeader;
        ListSortDirection direction;

        if (headerClicked != null)
        {
```

```

    if (headerClicked.Role != GridViewColumnHeaderRole.Padding)
    {
        if (headerClicked != _lastHeaderClicked)
        {
            direction = ListSortDirection.Ascending;
        }
        else
        {
            if (_lastDirection == ListSortDirection.Ascending)
            {
                direction = ListSortDirection.Descending;
            }
            else
            {
                direction = ListSortDirection.Ascending;
            }
        }

        var columnBinding =
headerClicked.Column.DisplayMemberBinding as Binding;
        var sortBy = columnBinding?.Path.Path ??
headerClicked.Column.Header as string;

        Sort(sortBy, direction);

        if (direction == ListSortDirection.Ascending)
        {
            headerClicked.Column.HeaderTemplate =
                Resources["HeaderTemplateArrowUp"] as DataTemplate;
        }
        else
        {
            headerClicked.Column.HeaderTemplate =
                Resources["HeaderTemplateArrowDown"] as DataTemplate;
        }

        // Remove arrow from previously sorted header
        if (_lastHeaderClicked != null && _lastHeaderClicked !=
headerClicked)
        {
            _lastHeaderClicked.Column.HeaderTemplate = null;
        }

        _lastHeaderClicked = headerClicked;
        _lastDirection = direction;
    }
}
}

```

下面的示例演示由事件处理程序调用以供对数据进行排序的排序算法。通过创建新的 `SortDescription` 结构执行排序。

C#

```
private void Sort(string sortBy, ListSortDirection direction)
{
    ICollectionView dataView =
        CollectionViewSource.GetDefaultView(lv.ItemsSource);

    dataView.SortDescriptions.Clear();
    SortDescription sd = new SortDescription(sortBy, direction);
    dataView.SortDescriptions.Add(sd);
    dataView.Refresh();
}
```

另请参阅

- [ListView](#)
- [GridView](#)
- [ListView 概述](#)
- [GridView 概述](#)
- [操作指南主题](#)

如何：为 ListView 创建自定义视图模式

项目 • 2023/02/06

此示例演示如何为 View 控件创建自定义 ListView 模式。

示例

为 ListView 控件创建自定义视图时，必须使用 ViewBase 类。下面的示例显示了一个派生自 ViewBase 类的名为 PlainView 的视图模式。

C#

```
public class PlainView : ViewBase
{
    public static readonly DependencyProperty ItemContainerStyleProperty =
        ItemsControl.ItemContainerStyleProperty.AddOwner(typeof(PlainView));

    public Style ItemContainerStyle
    {
        get { return (Style)GetValue(ItemContainerStyleProperty); }
        set { SetValue(ItemContainerStyleProperty, value); }
    }

    public static readonly DependencyProperty ItemTemplateProperty =
        ItemsControl.ItemTemplateProperty.AddOwner(typeof(PlainView));

    public DataTemplate ItemTemplate
    {
        get { return (DataTemplate)GetValue(ItemTemplateProperty); }
        set { SetValue(ItemTemplateProperty, value); }
    }

    public static readonly DependencyProperty ItemWidthProperty =
        WrapPanel.ItemWidthProperty.AddOwner(typeof(PlainView));

    public double ItemWidth
    {
        get { return (double)GetValue(ItemWidthProperty); }
        set { SetValue(ItemWidthProperty, value); }
    }

    public static readonly DependencyProperty ItemHeightProperty =
        WrapPanel.ItemHeightProperty.AddOwner(typeof(PlainView));

    public double ItemHeight
    {
        get { return (double)GetValue(ItemHeightProperty); }
    }
}
```

```

        set { SetValue(ItemHeightProperty, value); }

    }

    protected override object DefaultStyleKey
    {
        get
        {
            return new ComponentResourceKey(GetType(), "myPlainViewDSK");
        }
    }
}

```

要将样式应用于自定义视图，请使用 [Style](#) 类。下面的示例为 `PlainView` 视图模式定义 [Style](#)。在前面的示例中，此样式被设置为为 `PlainView` 定义的 `DefaultStyleKey` 属性的值。

XAML

```

<Style x:Key="{ComponentResourceKey
    TypeInTargetAssembly={x:Type l:PlainView},
    ResourceId=myPlainViewDSK}"
    TargetType="{x:Type ListView}"
    BasedOn="{StaticResource {x:Type ListBox}}"
    >
    <Setter Property="HorizontalContentAlignment"
        Value="Center"/>
    <Setter Property="ItemContainerStyle"
        Value="{Binding (ListView.View).ItemContainerStyle,
        RelativeSource={RelativeSource Self}}"/>
    <Setter Property="ItemTemplate"
        Value="{Binding (ListView.View).ItemTemplate,
        RelativeSource={RelativeSource Self}}"/>
    <Setter Property="ItemsPanel">
        <Setter.Value>
            <ItemsPanelTemplate>
                <WrapPanel Width="{Binding (FrameworkElement.ActualWidth),
                    RelativeSource
                    AncestorType=ScrollContentPresenter}">
                    ItemWidth="{Binding (ListView.View).ItemWidth,
                    RelativeSource={RelativeSource AncestorType=ListView}}"
                    MinWidth="{Binding (ListView.View).ItemWidth,
                    RelativeSource={RelativeSource AncestorType=ListView}}"
                    ItemHeight="{Binding (ListView.View).ItemHeight,
                    RelativeSource={RelativeSource AncestorType=ListView}}"/>
            </ItemsPanelTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

要在自定义视图模式下定义数据布局，请定义 [DataTemplate](#) 对象。下面的示例定义了一个 [DataTemplate](#)，可用于在 `PlainView` 视图模式下显示数据。

XAML

```
<DataTemplate x:Key="centralTile">
    <StackPanel Height="100" Width="90">
        <Grid Width="70" Height="70" HorizontalAlignment="Center">
            <Image Source="{Binding XPath=@Image}" Margin="6,6,6,9"/>
        </Grid>
        <TextBlock Text="{Binding XPath=@Name}" FontSize="13"
            HorizontalAlignment="Center" Margin="0,0,0,1" />
        <TextBlock Text="{Binding XPath=@Type}" FontSize="9"
            HorizontalAlignment="Center" Margin="0,0,0,1" />
    </StackPanel>
</DataTemplate>
```

下面的示例显示如何为使用前面示例中定义的 `DataTemplate` 的 `PlainView` 视图模式定义一个 `ResourceKey`。

XAML

```
<l:PlainView x:Key="tileView"
    ItemTemplate="{StaticResource centralTile}"
    ItemWidth="100"/>
```

如果将 `View` 属性设置为资源键，则 `ListView` 控件可以使用自定义视图。下面的示例演示如何将 `PlainView` 指定为 `ListView` 的视图模式。

C#

```
//Set the ListView View property to the tileView custom view
lv.View = lv.FindResource("tileView") as ViewBase;
```

有关完整的示例，请参阅[具有多个视图的 ListView \(C#\)](#) 或[具有多个视图的 ListView \(Visual Basic\)](#)。

另请参阅

- [ListView](#)
- [GridView](#)
- [操作指南主题](#)
- [ListView 概述](#)
- [GridView 概述](#)

如何：使用模板创建使用 GridView 的 ListView 的样式

项目 • 2023/02/06

此示例演示如何使用 DataTemplate 和 Style 对象来指定使用 GridView 视图模式的 ListView 控件的外观。

示例

下面的示例显示了自定义 GridViewColumn 列标题外观的 Style 和 DataTemplate 对象。

XAML

```
<Style x:Key="myHeaderStyle" TargetType="{x:Type GridViewColumnHeader}">
    <Setter Property="Background" Value="LightBlue"/>
</Style>
```

XAML

```
<DataTemplate x:Key="myHeaderTemplate">
    <DockPanel>
        <CheckBox/>
        <TextBlock FontSize="16" Foreground="DarkBlue">
            <TextBlock.Text>
                <Binding/>
            </TextBlock.Text>
        </TextBlock>
    </DockPanel>
</DataTemplate>
```

下面的示例演示如何使用这些 Style 和 DataTemplate 对象来设置 GridViewColumn 的 HeaderContainerStyle 和 HeaderTemplate 属性。 DisplayMemberBinding 属性定义列单元格的内容。

XAML

```
<GridViewColumn Header="Month" Width="80"
    HeaderContainerStyle="{StaticResource myHeaderStyle}"
    HeaderTemplate="{StaticResource myHeaderTemplate}"
    DisplayMemberBinding="{Binding Path=Month}"/>
```

HeaderContainerStyle 和 HeaderTemplate 只是可用于自定义 GridView 控件列标题外观的几个属性中的两个。有关详细信息，请参阅 [GridView 列标题的样式和模板概述](#)。

下面的示例演示如何定义自定义 `GridViewColumn` 中单元格的外观的 `DataTemplate`。

XAML

```
<DataTemplate x:Key="myCellTemplateMonth">
    <DockPanel>
        <TextBlock Foreground="DarkBlue" HorizontalAlignment="Center">
            <TextBlock.Text>
                <Binding Path="Month"/>
            </TextBlock.Text>
        </TextBlock>
    </DockPanel>
</DataTemplate>
```

下面的示例演示如何使用此 `DataTemplate` 来定义 `GridViewColumn` 单元格的内容。 使用此模板代替前面 `GridViewColumn` 示例中显示的 `DisplayMemberBinding` 属性。

XAML

```
<GridViewColumn Header="Month" Width="80"
    CellTemplate="{StaticResource myCellTemplateMonth}" />
```

另请参阅

- [ListView](#)
- [GridView](#)
- [GridView 概述](#)
- [操作指南主题](#)
- [ListView 概述](#)

如何：为拖动的 GridView 列标题创建样式

项目 • 2023/02/06

此示例演示如何在用户更改列的位置时更改已拖动的 [GridViewColumnHeader](#) 的外观。

示例

将列标题拖动到使用 [GridView](#) 作为其视图模式的 [ListView](#) 中的另一个位置时，该列将移动到新位置。 拖动列标题时，除了原始标题之外，还会显示该标题的浮动副本。

[GridView](#) 中的列标题由 [GridViewColumnHeader](#) 对象表示。

若要自定义浮动标题和原始标题的外观，可以设置 [Triggers](#) 以修改 [GridViewColumnHeaderStyle](#)。 当 [IsPressed](#) 属性值为 `true`，且 [Role](#) 属性值为 [Floating](#) 时，会应用这些 [Triggers](#)。

若用户在鼠标停在 [GridViewColumnHeader](#) 上时按住鼠标按钮，[IsPressed](#) 属性值将更改为 `true`。 同样，当用户开始拖动操作时，[Role](#) 属性更改为 [Floating](#)。

下面的示例演示如何设置 [Triggers](#)，以在用户将列拖动到新位置时更改原始标题和浮动标题的 [Foreground](#) 和 [Background](#) 颜色。

XAML

```
<ControlTemplate TargetType="{x:Type GridViewColumnHeader}">
```

XAML

```
<ControlTemplate.Triggers>
```

XAML

```
<Trigger Property="IsPressed"
      Value="true">
  <Setter TargetName="HighlightBorder"
         Property="Visibility"
         Value="Hidden"/>
  <Setter TargetName="PART_HeaderGripper"
         Property="Visibility"
         Value="Hidden"/>
  <Setter Property="Background"
         Value="SkyBlue"/>
  <Setter Property="Foreground"
```

```
    Value="Yellow"/>  
</Trigger>
```

XAML

```
<Trigger Property="Role"  
        Value="Floating">  
    <Setter TargetName="PART_HeaderGripper"  
            Property="Visibility"  
            Value="Collapsed"/>  
    <Setter Property="Background"  
            Value="Yellow"/>  
    <Setter Property="Foreground"  
            Value="SkyBlue"/>  
</Trigger>
```

XAML

```
</ControlTemplate.Triggers>
```

XAML

```
</ControlTemplate>
```

另请参阅

- [GridViewColumnHeader](#)
- [GridViewColumnHeaderRole](#)
- [ListView](#)
- [GridView](#)
- [操作指南主题](#)
- [ListView 概述](#)
- [GridView 概述](#)

如何：使用 GridView 显示 ListView 内容

项目 • 2023/02/06

此示例演示如何为 ListView 控件定义 GridView 视图模式。

示例

可以通过指定 `GridViewColumn` 对象来定义 `GridView` 的视图模式。以下示例演示如何定义绑定到为 `ListView` 控件指定的数据内容的 `GridViewColumn` 对象。此 `GridView` 示例指定三个 `GridViewColumn` 对象，它们映射到设置为 `ListView` 控件的 `ItemsSource` 的 `EmployeeInfoDataSource` 的 `FirstName`、`LastName` 和 `EmployeeNumber` 字段。

XAML

```
<ListView ItemsSource="{Binding Source={StaticResource EmployeeInfoDataSource}}">

    <ListView.View>

        <GridView AllowsColumnReorder="true" ColumnHeaderToolTip="Employee Information">

            <GridViewColumn DisplayMemberBinding="{Binding Path=FirstName}" Header="First Name" Width="100"/>

            <GridViewColumn DisplayMemberBinding="{Binding Path=LastName}" Width="100">
                <GridViewColumnHeader>Last Name
                    <GridViewColumnHeader.ContextMenu>
                        <ContextMenu MenuItem.Click="LastNameCM_Click" Name="LastNameCM">
                            <MenuItem Header="Ascending" />
                            <MenuItem Header="Descending" />
                        </ContextMenu>
                    </GridViewColumnHeader.ContextMenu>
                </GridViewColumnHeader>
            </GridViewColumn>

            <GridViewColumn DisplayMemberBinding="{Binding Path=EmployeeNumber}" Header="Employee No." Width="100"/>
        </GridView>

    </ListView.View>
</ListView>
```

下图显示了此示例的显示方式。

First Name	Last Name	Employee No.
Jesper	Aaberg	12345
Dominik	Paiha	98765
Yale	Li	23875
Muru	Subramani	49392

另请参阅

- [ListView](#)
- [GridView](#)
- [ListView 概述](#)
- [GridView 概述](#)
- [操作指南主题](#)

如何：使用触发器为 ListView 中的选定项设置样式

项目 • 2023/02/06

此示例演示如何为 [ListViewItem](#) 控件定义 [Triggers](#)，以便当 [ListViewItem](#) 的属性值发生更改时，[ListViewItem](#) 的 [Style](#) 也会相应更改。

示例

如果想要更改 [ListViewItem](#) 的 [Style](#) 以响应属性更改，则为 [Style](#) 更改定义 [Triggers](#)。

下面的示例定义了一个 [Trigger](#)，它可将 [Foreground](#) 属性设置为 [Blue](#)，并在 [IsMouseOver](#) 属性更改为 [true](#) 时，更改 [Cursor](#) 以显示 [Hand](#)。

XAML

```
<Style x:Key="MyContainer" TargetType="{x:Type ListViewItem}">

    <Setter Property="Margin" Value="0,1,0,0"/>
    <Setter Property="Height" Value="21"/>

    <Style.Triggers>
```

XAML

```
<Trigger Property="IsMouseOver" Value="true">
    <Setter Property="Foreground" Value="Blue" />
    <Setter Property="Cursor" Value="Hand"/>
</Trigger>
```

XAML

```
</Style.Triggers>
</Style>
```

下面的示例定义了一个 [MultiTrigger](#)，它可在 [ListViewItem](#) 为选定项目具有键盘焦点时，将 [ListViewItem](#) 的 [Foreground](#) 属性设置为 [Yellow](#)。

XAML

```
<Style x:Key="MyContainer" TargetType="{x:Type ListViewItem}">

    <Setter Property="Margin" Value="0,1,0,0"/>
```

```
<Setter Property="Height" Value="21"/>
```

```
<Style.Triggers>
```

XAML

```
<MultiTrigger>
  <MultiTrigger.Conditions>
    <Condition Property="IsSelected" Value="true" />
    <Condition Property="Selector.IsSelectionActive" Value="true" />
  </MultiTrigger.Conditions>
  <Setter Property="Foreground" Value="Yellow" />
</MultiTrigger>
```

XAML

```
</Style.Triggers>
</Style>
```

另请参阅

- [Control](#)
- [ListView](#)
- [GridView](#)
- [操作指南主题](#)
- [ListView 概述](#)
- [GridView 概述](#)

如何：使用 CheckBox 创建 ListViewItem

项目 • 2023/02/06

此示例显示如何在使用 [GridView](#) 的 [ListView](#) 控件中显示 [CheckBox](#) 控件的列。

示例

要在 [ListView](#) 中创建包含 [CheckBox](#) 控件的列，请创建包含 [CheckBox](#) 的 [DataTemplate](#)。然后将 [GridViewColumn](#) 的 [CellTemplate](#) 设置为 [DataTemplate](#)。

下面的示例显示了包含 [CheckBox](#) 的 [DataTemplate](#)。该示例将 [CheckBox](#) 的 [IsChecked](#) 属性绑定到包含它的 [ListViewItem](#) 的 [IsSelected](#) 属性值。因此，选择包含 [CheckBox](#) 的 [ListViewItem](#) 时，会勾选 [CheckBox](#)。

XAML

```
<DataTemplate x:Key="FirstCell">
    <StackPanel Orientation="Horizontal">
        <CheckBox IsChecked="{Binding Path=IsSelected,
            RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
ListViewItem}}}" />
    </StackPanel>
</DataTemplate>
```

下面的示例演示如何创建 [CheckBox](#) 控件列。为了生成列，本例会将 [GridViewColumn](#) 的 [CellTemplate](#) 属性设置为 [DataTemplate](#)。

XAML

```
<GridViewColumn CellTemplate="{StaticResource FirstCell}"
    Width="30"/>
```

另请参阅

- [Control](#)
- [ListView](#)
- [GridView](#)
- [ListView 概述](#)
- [操作指南主题](#)
- [GridView 概述](#)

如何：使用 GridViewRowPresenter 来显示数据

项目 • 2023/02/06

此示例演示如何使用 [GridViewRowPresenter](#) 和 [GridViewHeaderRowPresenter](#) 对象在列中显示数据。

示例

以下示例演示如何使用 [GridViewRowPresenter](#) 和 [GridViewHeaderRowPresenter](#) 对象，指定显示 [DateTime](#) 对象的 [DayOfWeek](#) 和 [Year](#) 的 [GridViewColumnCollection](#)。该示例还为 [GridViewColumn](#) 的 [Header](#) 定义了 [Style](#)。

XAML

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x='http://schemas.microsoft.com/winfx/2006/xaml'
    xmlns:sys="clr-namespace:System;assembly=mscorlib">

    <Window.Resources>

        <Style x:Key="MyHeaderStyle" TargetType="{x:Type GridViewColumnHeader}">
            <Setter Property="Background" Value="BurlyWood"/>
        </Style>

        <GridViewColumnCollection x:Key="gvcc">
            <GridViewColumn Header="Year"
                DisplayMemberBinding="{Binding Year}"
                Width="80"/>
            <GridViewColumn Header="Day"
                DisplayMemberBinding="{Binding DayOfWeek}"
                Width="80" />
        </GridViewColumnCollection>
    </Window.Resources>

    <StackPanel>
        <GridViewHeaderRowPresenter Name="hrp" Columns="{StaticResource gvcc}"
            ColumnHeaderContainerStyle=
            "{StaticResource MyHeaderStyle}" />

        <GridViewRowPresenter Columns="{StaticResource gvcc}" >
            <GridViewRowPresenter.Content>
                <sys:DateTime>2005/2/1</sys:DateTime>
            </GridViewRowPresenter.Content>
        </GridViewRowPresenter>
    </StackPanel>
</Window>
```

```
<GridViewRowPresenter Columns="{StaticResource gvcc}" >
    <GridViewRowPresenter.Content>
        <sys:DateTime>2006/10/12</sys:DateTime>
    </GridViewRowPresenter.Content>
</GridViewRowPresenter>
</StackPanel>

</Window>
```

另请参阅

- [GridViewHeaderRowPresenter](#)
- [GridViewRowPresenter](#)
- [GridViewColumnCollection](#)
- [GridView 概述](#)

如何：对实现 GridView 的 ListView 中的项进行分组

项目 • 2023/02/06

此示例演示如何在 ListView 控件的 GridView 视图模式中显示项组。

示例

若要在 ListView 中显示项组，请定义 CollectionViewSource。以下示例演示根据 Catalog 数据字段的值对数据项进行分组的 CollectionViewSource。

XAML

```
<CollectionViewSource x:Key='src'
    Source="{Binding Source={StaticResource MyData},
        XPath=Item}">
    <CollectionViewSource.GroupDescriptions>
        <PropertyGroupDescription PropertyName="@Catalog" />
    </CollectionViewSource.GroupDescriptions>
</CollectionViewSource>
```

以下示例将 ListView 的 ItemsSource 设置为上一示例定义的 CollectionViewSource。该示例还定义了一个实现 Expander 控件的 GroupStyle。

XAML

```
<ListView ItemsSource='{Binding Source={StaticResource src}}'
    BorderThickness="0">
    <ListView.GroupStyle>
        <GroupStyle>
            <GroupStyle.ContainerStyle>
                <Style TargetType="{x:Type GroupItem}">
                    <Setter Property="Margin" Value="0,0,0,5"/>
                    <Setter Property="Template">
                        <Setter.Value>
                            <ControlTemplate TargetType="{x:Type GroupItem}">
                                <Expander IsExpanded="True" BorderBrush="#FFA4B97F"
                                    BorderThickness="0,0,0,1">
                                    <Expander.Header>
                                        <DockPanel>
                                            <TextBlock FontWeight="Bold" Text="{Binding
Path=Name}" Margin="5,0,0,0" Width="100"/>
                                            <TextBlock FontWeight="Bold"
                                                Text="{Binding Path=ItemCount}"/>
                                        </DockPanel>
                                </Expander>
                            </ControlTemplate>
                        </Setter.Value>
                    </Setter>
                </Style>
            </GroupStyle.ContainerStyle>
        </GroupStyle>
    </ListView.GroupStyle>
</ListView>
```

```
</Expander.Header>
<Expander.Content>
    <ItemsPresenter />
</Expander.Content>
</Expander>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</GroupStyle.ContainerStyle>
</GroupStyle>
</ListView.GroupStyle>
```

XAML

```
</ListView>
```

另请参阅

- [ListView](#)
- [GridView](#)
- [操作指南主题](#)
- [ListView 概述](#)
- [GridView 概述](#)

如何：为实现 GridView 的 ListView 中的行设置样式

项目 • 2023/02/06

此示例演示如何在使用 [GridViewView](#) 模式的 [ListView](#) 控件中设置行的样式。

示例

可以通过在 [ListView](#) 控件上设置 [ItemContainerStyle](#) 来设置 [ListView](#) 控件中的行的样式。 设置代表 [ListViewItem](#) 对象的项的样式。[ItemContainerStyle](#) 引用用于显示行内容的 [ControlTemplate](#) 对象。

从中提取了以下示例的完整示例显示了存储在 XML 数据库中的歌曲信息的集合。 数据库中的每首歌都有一个评级字段，此字段的值指定了歌曲信息行的显示方式。

下面的示例演示了如何为表示歌曲集合中的歌曲的 [ListViewItem](#) 对象定义 [ItemContainerStyle](#)。[ItemContainerStyle](#) 引用了指定如何显示一行歌曲信息的 [ControlTemplate](#) 对象。

XAML

```
<ListView.ItemContainerStyle>
<Style TargetType="{x:Type ListViewItem}" >
    <Setter Property="Template"
        Value="{StaticResource Default}"/>
    <Style.Triggers>
        <DataTrigger Binding="{Binding XPath=@Rating}" Value="5">
            <Setter Property="Template"
                Value="{StaticResource StronglyRecommended}"/>
        </DataTrigger>
        <DataTrigger Binding="{Binding XPath=@Rating}" Value="4">
            <Setter Property="Template"
                Value="{StaticResource Recommended}"/>
        </DataTrigger>
    </Style.Triggers>
</Style>
</ListView.ItemContainerStyle>
```

下面的示例演示了将文本字符串 "Strongly Recommended" 添加到行中的 [ControlTemplate](#)。 此模板在 [ItemContainerStyle](#) 中引用，并且在歌曲的评级值为 5 (五) 时显示。[ControlTemplate](#) 包括一个 [GridViewRowPresenter](#) 对象，该对象将行的内容按 [GridView](#) 视图模式所定义的列的形式进行布局。

XAML

```
<ControlTemplate x:Key="StronglyRecommended"
    TargetType='{x:Type ListViewItem}'>
<StackPanel Background="Beige">
    <GridViewRowPresenter Content="{TemplateBinding Content}"
        Columns="{TemplateBinding GridView.ColumnCollection}" />
    <TextBlock Background="LightBlue" Text="Strongly Recommended" />
</StackPanel>
</ControlTemplate>
```

下面的示例定义了 [GridView](#)。

XAML

```
<ListView.View>
    <GridView ColumnHeaderContainerStyle="{StaticResource MyHeaderStyle}">
        <GridViewColumn Header="Name"
            DisplayMemberBinding="{Binding XPath=@Name}"
            Width="100"/>
        <GridViewColumn Header="Time"
            DisplayMemberBinding="{Binding XPath=@Time}"
            Width="80"/>
        <GridViewColumn Header="Artist"
            DisplayMemberBinding="{Binding XPath=@Artist}"
            Width="80" />
        <GridViewColumn Header="Disk"
            DisplayMemberBinding="{Binding XPath=@Disk}"
            Width="100"/>
    </GridView>
</ListView.View>
```

另请参阅

- [ListView](#)
- [GridView](#)
- [操作指南主题](#)
- [ListView 概述](#)
- [样式设置和模板化](#)

如何：更改 ListView 中列的水平对齐方式

项目 • 2023/02/06

默认情况下，[ListViewItem](#) 中每列的内容均为左对齐。你可以通过以下方法更改每列的对齐方式：提供一个 [DataTemplate](#)，并在 [DataTemplate](#) 内的元素上设置 [HorizontalAlignment](#) 属性。本主题演示 [ListView](#) 默认的内容对齐方式以及更改 [ListView](#) 中某一列的对齐方式的方法。

示例

在以下示例中，[Title](#) 和 [ISBN](#) 列中的数据均为左对齐。

XAML

```
<!--XmlDataProvider is defined in a ResourceDictionary,  
such as Window.Resources-->  
<XmlDataProvider x:Key="InventoryData" XPath="Books">  
  <x:XData>  
    <Books xmlns="">  
      <Book ISBN="0-7356-0562-9" Stock="in" Number="9">  
        <Title>XML in Action</Title>  
        <Summary>XML Web Technology</Summary>  
      </Book>  
      <Book ISBN="0-7356-1370-2" Stock="in" Number="8">  
        <Title>Programming Microsoft Windows With C#</Title>  
        <Summary>C# Programming using the .NET Framework</Summary>  
      </Book>  
      <Book ISBN="0-7356-1288-9" Stock="out" Number="7">  
        <Title>Inside C#</Title>  
        <Summary>C# Language Programming</Summary>  
      </Book>  
      <Book ISBN="0-7356-1377-X" Stock="in" Number="5">  
        <Title>Introducing Microsoft .NET</Title>  
        <Summary>Overview of .NET Technology</Summary>  
      </Book>  
      <Book ISBN="0-7356-1448-2" Stock="out" Number="4">  
        <Title>Microsoft C# Language Specifications</Title>  
        <Summary>The C# language definition</Summary>  
      </Book>  
    </Books>  
  </x:XData>  
</XmlDataProvider>
```

XAML

```
<ListView ItemsSource="{Binding Source={StaticResource InventoryData},  
XPath=Book}">
```

```
<ListView.View>
    <GridView>
        <GridViewColumn Width="300" Header="Title"
            DisplayMemberBinding="{Binding XPath=Title}"/>
        <GridViewColumn Width="150" Header="ISBN"
            DisplayMemberBinding="{Binding XPath=@ISBN}"/>
    </GridView>
</ListView.View>
</ListView>
```

若要更改 ISBN 列的对齐方式，你需要将每个 ListViewItem 的 HorizontalContentAlignment 属性指定为 Stretch，这样每个 ListViewItem 中的元素可跨各列的整个宽度分布。 ListView 绑定到数据源，因此你需要创建设置 HorizontalContentAlignment 的样式。接下来，需要使用 DataTemplate 来显示内容，而不是使用 DisplayMemberBinding 属性。 若要显示每个模板的 ISBN，DataTemplate 只能包含一个 TextBlock（它将 HorizontalAlignment 属性设置为 Right）。

以下示例定义使 ISBN 列右对齐所必需的样式和 DataTemplate，并将 GridViewColumn 更改为引用 DataTemplate。

XAML

```
<!--The Style and DataTemplate are defined in a ResourceDictionary,
such as Window.Resources-->
<Style TargetType="ListViewItem">
    <Setter Property="HorizontalContentAlignment" Value="Stretch"/>
</Style>

<DataTemplate x:Key="ISBNTemplate">
    <TextBlock HorizontalAlignment="Right"
        Text="{Binding XPath=@ISBN}"/>
</DataTemplate>
```

XAML

```
<ListView ItemsSource="{Binding Source={StaticResource InventoryData},
XPath=Book}">
    <ListView.View>
        <GridView>
            <GridViewColumn Width="300" Header="Title"
                DisplayMemberBinding="{Binding XPath=Title}"/>
            <GridViewColumn Width="150" Header="ISBN"
                CellTemplate="{StaticResource ISBNTemplate}"/>
        </GridView>
    </ListView.View>
</ListView>
```

另请参阅

- [数据绑定概述](#)
- [数据模板化概述](#)
- [使用 XMLDataProvider 和 XPath 查询绑定到 XML 数据](#)
- [ListView 概述](#)

如何：处理 ListView 中每一项的 MouseDoubleClick 事件

项目 • 2023/02/06

要为 `ListView` 中的项处理事件，需要为每个 `ListViewItem` 添加一个事件处理程序。当 `ListView` 绑定到数据源时，无需显式创建 `ListViewItem`，但可以通过将 `EventSetter` 添加到 `ListViewItem` 的样式来处理每个项的事件。

示例

以下示例创建一个数据绑定 `ListView`，并创建一个 `Style` 以向每个 `ListViewItem` 添加一个事件处理程序。

XAML

```
<!--XmlDataProvider is defined in a ResourceDictionary,
such as Window.Resources-->
<XmlDataProvider x:Key="InventoryData" XPath="Books">
    <x:XData>
        <Books xmlns="">
            <Book ISBN="0-7356-0562-9" Stock="in" Number="9">
                <Title>XML in Action</Title>
                <Summary>XML Web Technology</Summary>
            </Book>
            <Book ISBN="0-7356-1370-2" Stock="in" Number="8">
                <Title>Programming Microsoft Windows With C#</Title>
                <Summary>C# Programming using the .NET Framework</Summary>
            </Book>
            <Book ISBN="0-7356-1288-9" Stock="out" Number="7">
                <Title>Inside C#</Title>
                <Summary>C# Language Programming</Summary>
            </Book>
            <Book ISBN="0-7356-1377-X" Stock="in" Number="5">
                <Title>Introducing Microsoft .NET</Title>
                <Summary>Overview of .NET Technology</Summary>
            </Book>
            <Book ISBN="0-7356-1448-2" Stock="out" Number="4">
                <Title>Microsoft C# Language Specifications</Title>
                <Summary>The C# language definition</Summary>
            </Book>
        </Books>
    </x:XData>
</XmlDataProvider>
```

XAML

```
<!--The Style is defined in a ResourceDictionary,  
such as Window.Resources-->  
<Style TargetType="ListViewItem">  
    <EventSetter Event="MouseDoubleClick"  
Handler="ListViewItem_MouseDoubleClick"/>  
</Style>
```

XAML

```
<ListView ItemsSource="{Binding Source={StaticResource InventoryData},  
XPath=Book}">  
    <ListView.View>  
        <GridView>  
            <GridViewColumn Width="300" Header="Title"  
                DisplayMemberBinding="{Binding XPath=Title}"/>  
            <GridViewColumn Width="150" Header="ISBN"  
                DisplayMemberBinding="{Binding XPath=@ISBN}"/>  
        </GridView>  
    </ListView.View>  
</ListView>
```

下面的示例处理 [MouseDoubleClick](#) 事件。

C#

```
void ListViewItem_MouseDoubleClick(object sender, MouseButtonEventArgs e)  
{  
  
    XmlElement book = ((ListViewItem) sender).Content as XmlElement;  
  
    if (book == null)  
    {  
        return;  
    }  
  
    if (book.GetAttribute("Stock") == "out")  
    {  
        MessageBox.Show("Time to order more copies of " +  
book["Title"].InnerText);  
    }  
    else  
    {  
        MessageBox.Show(book["Title"].InnerText + " is available.");  
    }  
}
```

① 备注

尽管将 ListView 绑定到数据源最常见，但可以使用样式将事件处理程序添加到非数据绑定 ListView 中的每个 ListViewItem，无论是否显式创建 ListViewItem。有关显式和隐式创建的 ListViewItem 控件的详细信息，请参阅 [ItemsControl](#)。

另请参阅

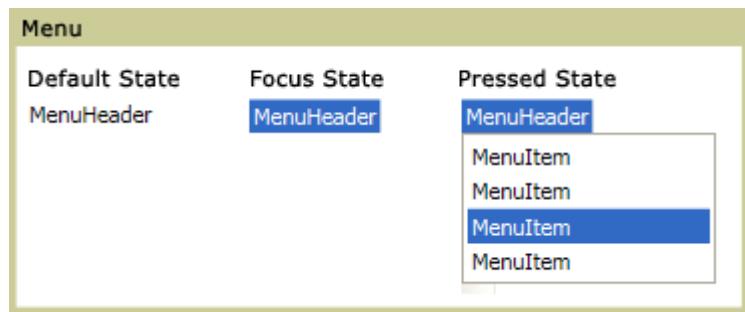
- [XmlElement](#)
- [数据绑定概述](#)
- [样式设置和模板化](#)
- [使用 XMLDataProvider 和 XPath 查询绑定到 XML 数据](#)
- [ListView 概述](#)

菜单

项目 · 2023/02/06

[Menu](#) 是一个控件，可用于对与命令或事件处理程序关联的元素以分层方式进行组织。每个 [MenuItem](#) 控件。每个 [MenuItem](#) 都可以调用命令或调用 [Click](#) 事件处理程序。[MenuItem](#) 还可以将多个 [MenuItem](#) 元素作为子元素，形成子菜单。

下图显示了 [Menu](#) 控件的三种不同状态。默认状态是没有任何设备（例如鼠标指针）停留在 [Menu](#) 上。当鼠标指针悬停在 [Menu](#) 上时将进入焦点状态，当鼠标按钮在 [Menu](#) 上单击时将进入按下状态。



处于不同状态的 [Menu](#)

本节内容

[菜单概述](#)

参考

[Menu](#)

[MenuItem](#)

[MenuBase](#)

[ContextMenu](#)

相关章节

菜单概述

项目 • 2023/02/06

使用 [Menu](#) 类，可以按分层顺序组织与命令和事件处理程序相关联的元素。每个 [Menu](#) 元素都包含一组 [MenuItem](#) 元素。

Menu 控件

[Menu](#) 控件显示用于为应用程序指定命令或选项的项列表。通常，单击 [MenuItem](#) 即可打开子菜单或导致应用程序执行某个命令。

创建菜单

以下示例创建一个 [Menu](#) 来操作 [TextBox](#) 中的文本。 [Menu](#) 包含 [MenuItem](#) 对象，这些对象使用 [Command](#)、[IsCheckable](#) 和 [Header](#) 属性以及 [Checked](#)、[Unchecked](#) 和 [Click](#) 事件。

XAML

```
<Menu>
    <MenuItem Header="_Edit">
        <MenuItem Command="ApplicationCommands.Copy"/>
        <MenuItem Command="ApplicationCommands.Cut"/>
        <MenuItem Command="ApplicationCommands.Paste"/>
    </MenuItem>
    <MenuItem Header="_Font">
        <MenuItem Header="_Bold" IsCheckable="True"
            Checked="Bold_Checked"
            Unchecked="Bold_Unchecked"/>
        <MenuItem Header="_Italic" IsCheckable="True"
            Checked="Italic_Checked"
            Unchecked="Italic_Unchecked"/>
        <Separator/>
        <MenuItem Header="I_ncrease Font Size"
            Click="IncreaseFont_Click"/>
        <MenuItem Header="_Decrease Font Size"
            Click="DecreaseFont_Click"/>
    </MenuItem>
</Menu>
<TextBox Name="textBox1" TextWrapping="Wrap"
    Margin="2">
    The quick brown fox jumps over the lazy dog.
</TextBox>
```

C#

```

private void Bold_Checked(object sender, RoutedEventArgs e)
{
    textBox1.FontWeight = FontWeights.Bold;
}

private void Bold_Unchecked(object sender, RoutedEventArgs e)
{
    textBox1.FontWeight = FontWeights.Normal;
}

private void Italic_Checked(object sender, RoutedEventArgs e)
{
    textBox1.FontStyle = FontStyles.Italic;
}

private void Italic_Unchecked(object sender, RoutedEventArgs e)
{
    textBox1.FontStyle = FontStyles.Normal;
}

private void IncreaseFont_Click(object sender, RoutedEventArgs e)
{
    if (textBox1.FontSize < 18)
    {
        textBox1.FontSize += 2;
    }
}

private void DecreaseFont_Click(object sender, RoutedEventArgs e)
{
    if (textBox1.FontSize > 10)
    {
        textBox1.FontSize -= 2;
    }
}

```

VB

```

Private Sub Bold_Checked(ByVal sender As Object, ByVal e As RoutedEventArgs)
    textBox1.FontWeight = FontWeights.Bold
End Sub

Private Sub Bold_Unchecked(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    textBox1.FontWeight = FontWeights.Normal
End Sub

Private Sub Italic_Checked(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    textBox1.FontStyle = FontStyles.Italic
End Sub

```

```

Private Sub Italic_Unchecked(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    textBox1.FontStyle = FontStyles.Normal
End Sub

Private Sub IncreaseFont_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    If textBox1.FontSize < 18 Then
        textBox1.FontSize += 2
    End If
End Sub

Private Sub DecreaseFont_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    If textBox1.FontSize > 10 Then
        textBox1.FontSize -= 2
    End If
End Sub

```

使用键盘快捷方式的菜单项

键盘快捷方式是可通过键盘输入来调用 [Menu](#) 命令的字符组合。例如，“复制”的快捷方式是 CTRL+C。有两个属性可用于键盘快捷方式和菜单项：[InputGestureText](#) 或 [Command](#)。

InputGestureText

以下示例演示如何使用 [InputGestureText](#) 属性将键盘快捷方式文本分配到 [MenuItem](#) 控件。这仅将键盘快捷方式放置在菜单项中。它不会将命令与 [MenuItem](#) 相关联。应用程序必须处理用户的输入才能执行操作。

XAML

```

<MenuItem Header="_Cut" InputGestureText="Ctrl+X"/>
<MenuItem Header="_Find" InputGestureText="Ctrl+F"/>

```

命令

以下示例演示如何使用 [Command](#) 属性将 Open 和 Save 命令与 [MenuItem](#) 控件相关联。命令属性不仅将命令与 [MenuItem](#) 相关联，它还提供输入手势文本以用作快捷方式。

XAML

```

<MenuItem Header="_Open" Command="ApplicationCommands.Open"/>
<MenuItem Header="_Save" Command="ApplicationCommands.Save"/>

```

`MenuItem` 类还有一个 `CommandTarget` 属性，该属性指定发生命令的元素。如果未设置 `CommandTarget`，则具有键盘焦点的元素接收命令。有关命令的详细信息，请参阅[命令概述](#)。

菜单样式设置

使用菜单样式设置，可以显著更改 `Menu` 控件的外观和行为，而无需编写自定义控件。除了设置视觉属性，还可以将 `Style` 应用到控件的个别部件，通过属性更改控件部件的行为，或者添加其他部件或更改控件的布局。以下示例演示向 `Menu` 控件添加 `Style` 的几种方法。

第一个代码示例定义了称为 `Simple` 的 `Style`，演示如何在样式中使用当前系统设置。代码将 `MenuHighlightBrush` 的颜色分配为菜单的背景色，将 `MenuTextBrush` 分配为菜单的前景色。请注意，使用资源键分配画笔。

XAML

```
<Style x:Key="Simple" TargetType="{x:Type MenuItem}">
    <Setter Property = "Background" Value= "{DynamicResource {x:Static SystemColors.MenuHighlightBrushKey}}"/>
    <Setter Property = "Foreground" Value= "{DynamicResource {x:Static SystemColors.MenuTextBrushKey}}"/>
    <Setter Property = "Height" Value= "{DynamicResource {x:Static SystemParameters.CaptionHeightKey}}"/>
</Style>
```

以下示例使用 `Trigger` 元素，这些元素可用于更改 `MenuItem` 的外观以响应在 `Menu` 上发生的事件。在 `Menu` 上移动鼠标时，菜单项的前景色和字体特征会变化。

XAML

```
<Style x:Key="Triggers" TargetType="{x:Type MenuItem}">
    <Style.Triggers>
        <Trigger Property="MenuItem.IsMouseOver" Value="true">
            <Setter Property = "Foreground" Value="Red"/>
            <Setter Property = "FontSize" Value="16"/>
            <Setter Property = "FontStyle" Value="Italic"/>
        </Trigger>
    </Style.Triggers>
</Style>
```

另请参阅

- WPF 控件库示例 ↗

陪审团

项目 • 2023/02/06

[Panel](#) 是在 Windows Presentation Foundation (WPF) 中支持应用程序布局的所有元素的基类。

本节内容

[面板概述](#)

[操作指南主题](#)

参考

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

相关章节

[布局](#)

[演练：我的第一个 WPF 桌面应用程序](#)

[ScrollViewer 概述](#)

面板概述

项目 · 2022/09/27

[Panel](#) 元素是控制元素呈现（它们的大小和尺寸、位置和子内容的排列）的组件。

Windows Presentation Foundation (WPF) 提供许多预定义的 [Panel](#) 元素以及构造自定义 [Panel](#) 元素的功能。

本主题包含以下各节：

- [Panel 类](#)
- [Panel 元素公共成员](#)
- [派生 Panel 元素](#)
- [用户界面 Panel](#)
- [嵌套 Panel 元素](#)
- [自定义 Panel 元素](#)
- [本地化/全球化支持](#)

Panel 类

[Panel](#) 是在 Windows Presentation Foundation (WPF) 中提供布局支持的所有元素的基本类。派生 [Panel](#) 元素用于在 Extensible Application Markup Language (XAML) 和代码中定位和排列元素。

WPF 包含一套全面的派生面板实现，可支持许多复杂的布局。这些派生类公开可实现大部分标准用户界面 (UI) 方案的属性和方法。如果开发者找不到满足其需求的子排列行为，可以通过重写 [ArrangeOverride](#) 和 [MeasureOverride](#) 方法创建新的布局。有关自定义布局行为的详细信息，请参阅[自定义 Panel 元素](#)。

Panel 公共成员

所有 [Panel](#) 元素都支持 [FrameworkElement](#) 定义的基本大小调整和定位属性，包括 [Height](#)、[Width](#)、[HorizontalAlignment](#)、[VerticalAlignment](#)、[Margin](#) 和 [LayoutTransform](#)。有关 [FrameworkElement](#) 定义的定位属性的其他信息，请参阅[对齐、边距和填充概述](#)。

[Panel](#) 公开对了解和使用布局至关重要的其他属性。[Background](#) 属性用于借助 [Brush](#) 填充派生面板元素的边界之间的区域。[Children](#) 表示组成 [Panel](#) 的子元素集合。

`InternalChildren` 表示 `Children` 集合的内容以及由数据绑定生成的成员。两者均由父 `Panel` 内承载的子元素的 `UIElementCollection` 组成。

面板还公开可用于在派生 `Panel` 中实现分层顺序的 `Panel.ZIndex` 附加属性。面板的 `Children` 集合中具有较高 `Panel.ZIndex` 值的成员显示在具有较低 `Panel.ZIndex` 值的成员之前。对于 `Canvas` 和 `Grid` 等使子级共享相同坐标空间的面板，这尤其有用。

`Panel` 还定义 `OnRender` 方法，该方法可用于重写 `Panel` 的默认呈现行为。

附加属性

派生面板元素广泛使用附加属性。附加属性是没有常规传统公共语言运行时 (CLR) 属性“包装器”的依赖属性的特殊化形式。附加属性在 Extensible Application Markup Language (XAML) 中具有特殊化的语法，可在后面的几个示例中看到。

附加属性的一个用途是允许子元素存储实际上由父元素定义的属性的唯一值。此功能的一项应用是让子元素通知父级它们希望如何在用户界面 (UI) 中呈现，这对应用程序布局非常有用。有关详细信息，请参阅[附加属性概述](#)。

派生 Panel 元素

许多对象派生自 `Panel`，但并非全部都旨在用作根布局提供程序。有六个专为创建应用程序 UI 而设计的已定义面板类（`Canvas`、`DockPanel`、`Grid`、`StackPanel`、`VirtualizingStackPanel` 和 `WrapPanel`）。

每个面板元素都封装自己的特殊功能，如下表所示。

元素名称	是否 为 UI 面 板？	说明
<code>Canvas</code>	是	定义一个区域，可在其中通过相对于 <code>Canvas</code> 区域的坐标显式定位子元素。
<code>DockPanel</code>	是	定义一个区域，从中可以按相对位置水平或垂直排列各个子元素。
<code>Grid</code>	是	定义一个由列和行组成的灵活网格区域。可以使用 <code>Margin</code> 属性精确定位 <code>Grid</code> 的子元素。
<code>StackPanel</code>	是	将子元素排列成水平或垂直的一行。
<code>TabPanel</code>	否	在 <code>TabControl</code> 中处理选项卡按钮的布局。
<code>ToolBarOverflowPanel</code>	否	在 <code>ToolBar</code> 控件中排列内容。

元素名称	是否 为 UI 面 板？	说明
UniformGrid	否	UniformGrid 用于在单元格大小全部相等的网格中排列子级。
VirtualizingPanel	否	为可以“虚拟化”其子级集合的面板提供基类。
VirtualizingStackPanel	是	在水平或垂直方向上将内容排列为一行并使其虚拟化。
WrapPanel	是	WrapPanel 按从左到右的顺序位置定位子元素，在包含框的边缘处将内容切换到下一行。后续排序按照从上至下或从右至左的顺序进行，具体取决于 Orientation 属性的值。

用户界面 Panel

UI 方案中有六个面板类可用：[Canvas](#)、[DockPanel](#)、[Grid](#)、[StackPanel](#)、[VirtualizingStackPanel](#) 和 [WrapPanel](#)。这些面板元素易于使用、功能齐全并且可扩展，足以适用于大多数应用程序。

每个派生 [Panel](#) 元素都以不同方式处理大小调整约束。了解 [Panel](#) 如何处理水平或垂直方向上的约束可以使布局更容易预测。

Panel 名称	x 维度	y 维度
Canvas	按内容约束	按内容约束
DockPanel	约束	约束
StackPanel (垂直方向)	约束	按内容约束
StackPanel (水平方向)	按内容约束	约束
Grid	约束	约束，Auto 应用于行和列的情形除外
WrapPanel	按内容约束	按内容约束

可在下方找到其中每种元素的更多详细说明和使用示例。

画布

[Canvas](#) 元素支持根据绝对 x 和 y 坐标定位内容。元素可以在唯一位置绘制；或者，如果元素占用了相同坐标，则这些元素在标记中显示的顺序决定它们的绘制顺序。

[Canvas](#) 提供任何 [Panel](#) 的最灵活布局支持。 Height 和 Width 属性用于定义画布的区域，并为内部的元素分配相对于父 [Canvas](#) 的区域的绝对坐标。对于附加属性 [Canvas.Left](#)、[Canvas.Top](#)、[Canvas.Right](#) 和 [Canvas.Bottom](#)，允许对 [Canvas](#) 内的对象放置进行精细控制，使开发者可以在屏幕上精确定位和排列元素。

Canvas 内 ClipToBounds

[Canvas](#) 可以将子元素定位在屏幕上的任何位置，甚至可以定位在其自己定义的 Height 和 Width 以外的坐标。此外，[Canvas](#) 不受其子级的大小影响。因此，子元素可以将其他元素过度绘制到父 [Canvas](#) 的边框之外。[Canvas](#) 的默认行为是允许在父 [Canvas](#) 的边界之外绘制子级。如果不需要此行为，可以将 [ClipToBounds](#) 属性设置为 `true`。这将导致 [Canvas](#) 剪裁到它自身的大小。[Canvas](#) 是唯一允许在其边界外绘制子级的布局元素。

[宽度属性比较示例](#) 中以图形方式展示了此行为。

定义和使用 Canvas

只需使用 Extensible Application Markup Language (XAML) 或代码即可实例化 [Canvas](#)。以下示例演示如何使用 [Canvas](#) 对内容进行绝对定位。此代码生成三个 100 像素正方形。第一个正方形为红色，其左上角的 (x, y) 位置指定为 (0, 0)。第二个正方形为绿色，其左上角位置为 (100, 100)，在第一个正方形的右下方。第三个正方形为蓝色，其左上角为 (50, 50)，因此包含了第一个正方形的右下四分之一部分和第二个正方形的左上四分之一部分。由于第三个正方形最后布置，因此它看起来在另外两个正方形上方，即，重叠部分采用第三个正方形的颜色。

C#

```
// Create the application's main window
mainWindow = new Window();
mainWindow.Title = "Canvas Sample";

// Create the Canvas
myParentCanvas = new Canvas();
myParentCanvas.Width = 400;
myParentCanvas.Height = 400;

// Define child Canvas elements
myCanvas1 = new Canvas();
myCanvas1.Background = Brushes.Red;
myCanvas1.Height = 100;
myCanvas1.Width = 100;
Canvas.SetTop(myCanvas1, 0);
Canvas.SetLeft(myCanvas1, 0);

myCanvas2 = new Canvas();
```

```

myCanvas2.Background = Brushes.Green;
myCanvas2.Height = 100;
myCanvas2.Width = 100;
Canvas.SetTop(myCanvas2, 100);
Canvas.SetLeft(myCanvas2, 100);

myCanvas3 = new Canvas();
myCanvas3.Background = Brushes.Blue;
myCanvas3.Height = 100;
myCanvas3.Width = 100;
Canvas.SetTop(myCanvas3, 50);
Canvas.SetLeft(myCanvas3, 50);

// Add child elements to the Canvas' Children collection
myParentCanvas.Children.Add(myCanvas1);
myParentCanvas.Children.Add(myCanvas2);
myParentCanvas.Children.Add(myCanvas3);

// Add the parent Canvas as the Content of the Window Object
mainWindow.Content = myParentCanvas;
mainWindow.Show ();

```

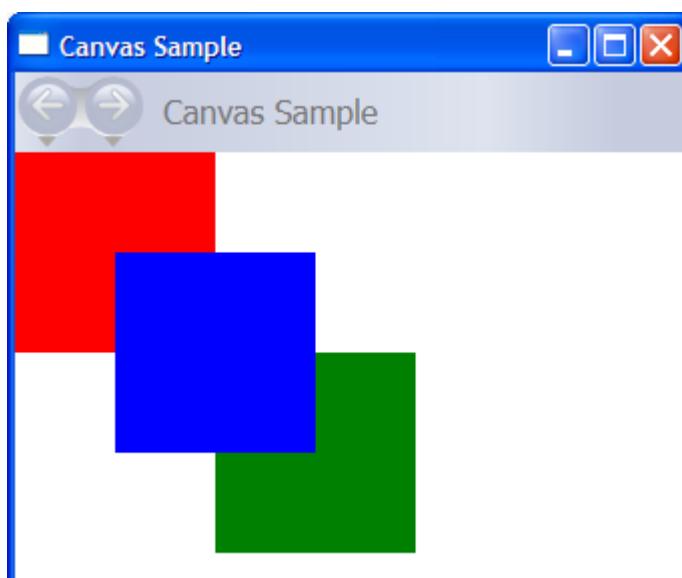
XAML

```

<Page WindowTitle="Canvas Sample"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
<Canvas Height="400" Width="400">
    <Canvas Height="100" Width="100" Top="0" Left="0" Background="Red"/>
    <Canvas Height="100" Width="100" Top="100" Left="100"
Background="Green"/>
    <Canvas Height="100" Width="100" Top="50" Left="50" Background="Blue"/>
</Canvas>
</Page>

```

已编译的应用程序将生成如下所示的新 UI。



DockPanel

[DockPanel](#) 元素使用在子内容元素中设置的 [DockPanel.Dock](#) 附加属性沿容器边缘定位内容。 [DockPanel.Dock](#) 设置为 [Top](#) 或 [Bottom](#) 时，它会将子元素放置在彼此的上方或下方。 [DockPanel.Dock](#) 设置为 [Left](#) 或 [Right](#) 时，它会将子元素放置在彼此的左侧或右侧。[LastChildFill](#) 属性确定添加为 [DockPanel](#) 的子级的最后一个元素的位置。

可以使用 [DockPanel](#) 定位一组相关控件，如一组按钮。或者，可以使用它创建“平移”的 UI，类似于 Microsoft Outlook 中的 UI。

按内容调整大小

如果未指定其 [Height](#) 和 [Width](#) 属性，则 [DockPanel](#) 会调整大小以适应其内容。大小可以增大或减小以容纳其子元素的大小。但是，当已指定这些属性，并且不再有空间能容纳下一个指定子元素时，[DockPanel](#) 将不会显示该子元素或后续子元素，并且不会测量后续子元素。

LastChildFill

默认情况下，[DockPanel](#) 元素的最后一个子元素会“填充”剩余的未分配空间。如果不希望此行为，请将 [LastChildFill](#) 属性设置为 `false`。

定义和使用 DockPanel

以下示例演示如何使用 [DockPanel](#) 为空间分区。五个 [Border](#) 元素添加为父 [DockPanel](#) 的子级。每个都使用不同的 [DockPanel](#) 位置属性来为空间分区。最后一个元素“填充”剩余的未分配空间。

C#

```
// Create the application's main window
mainWindow = new Window();
mainWindow.Title = "DockPanel Sample";

// Create the DockPanel
DockPanel myDockPanel = new DockPanel();
myDockPanel.LastChildFill = true;

// Define the child content
Border myBorder1 = new Border();
myBorder1.Height = 25;
myBorder1.Background = Brushes.SkyBlue;
myBorder1.BorderBrush = Brushes.Black;
myBorder1.BorderThickness = new Thickness(1);
```

```

DockPanel.SetDock(myBorder1, Dock.Top);
TextBlock myTextBlock1 = new TextBlock();
myTextBlock1.Foreground = Brushes.Black;
myTextBlock1.Text = "Dock = Top";
myBorder1.Child = myTextBlock1;

Border myBorder2 = new Border();
myBorder2.Height = 25;
myBorder2.Background = Brushes.SkyBlue;
myBorder2.BorderBrush = Brushes.Black;
myBorder2.BorderThickness = new Thickness(1);
DockPanel.SetDock(myBorder2, Dock.Top);
TextBlock myTextBlock2 = new TextBlock();
myTextBlock2.Foreground = Brushes.Black;
myTextBlock2.Text = "Dock = Top";
myBorder2.Child = myTextBlock2;

Border myBorder3 = new Border();
myBorder3.Height = 25;
myBorder3.Background = Brushes.LemonChiffon;
myBorder3.BorderBrush = Brushes.Black;
myBorder3.BorderThickness = new Thickness(1);
DockPanel.SetDock(myBorder3, Dock.Bottom);
TextBlock myTextBlock3 = new TextBlock();
myTextBlock3.Foreground = Brushes.Black;
myTextBlock3.Text = "Dock = Bottom";
myBorder3.Child = myTextBlock3;

Border myBorder4 = new Border();
myBorder4.Width = 200;
myBorder4.Background = Brushes.PaleGreen;
myBorder4.BorderBrush = Brushes.Black;
myBorder4.BorderThickness = new Thickness(1);
DockPanel.SetDock(myBorder4, Dock.Left);
TextBlock myTextBlock4 = new TextBlock();
myTextBlock4.Foreground = Brushes.Black;
myTextBlock4.Text = "Dock = Left";
myBorder4.Child = myTextBlock4;

Border myBorder5 = new Border();
myBorder5.Background = Brushes.White;
myBorder5.BorderBrush = Brushes.Black;
myBorder5.BorderThickness = new Thickness(1);
TextBlock myTextBlock5 = new TextBlock();
myTextBlock5.Foreground = Brushes.Black;
myTextBlock5.Text = "This content will fill the remaining space";
myBorder5.Child = myTextBlock5;

// Add child elements to the DockPanel Children collection
myDockPanel.Children.Add(myBorder1);
myDockPanel.Children.Add(myBorder2);
myDockPanel.Children.Add(myBorder3);
myDockPanel.Children.Add(myBorder4);
myDockPanel.Children.Add(myBorder5);

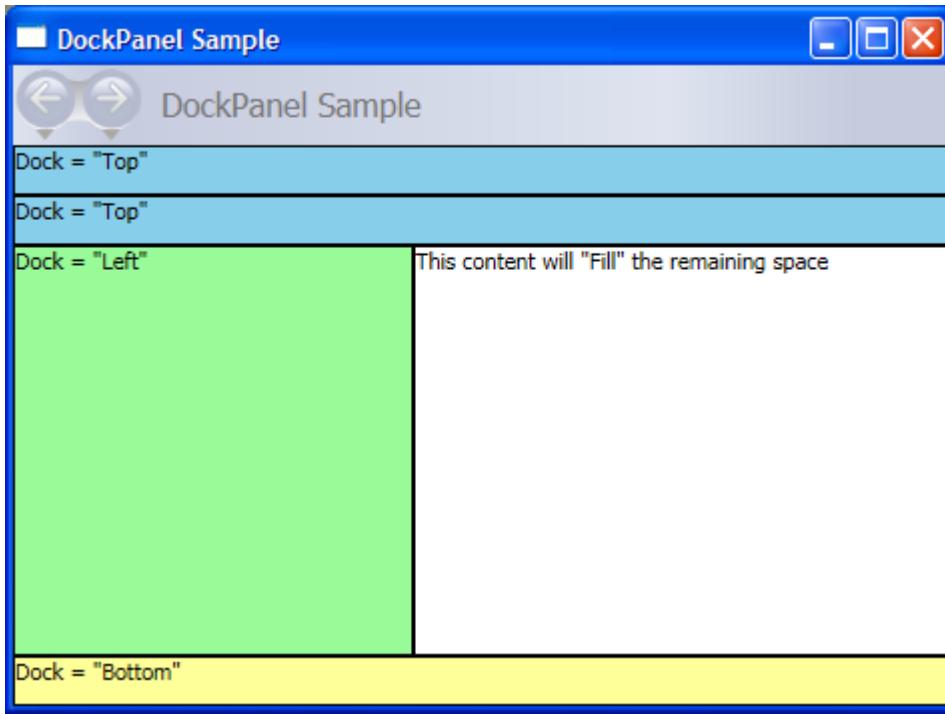
```

```
// Add the parent Canvas as the Content of the Window Object
mainWindow.Content = myDockPanel;
mainWindow.Show();
```

XAML

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
WindowTitle="DockPanel Sample">
    <DockPanel LastChildFill="True">
        <Border Height="25" Background="SkyBlue" BorderBrush="Black"
BorderThickness="1" DockPanel.Dock="Top">
            <TextBlock Foreground="Black">Dock = "Top"</TextBlock>
        </Border>
        <Border Height="25" Background="SkyBlue" BorderBrush="Black"
BorderThickness="1" DockPanel.Dock="Top">
            <TextBlock Foreground="Black">Dock = "Top"</TextBlock>
        </Border>
        <Border Height="25" Background="LemonChiffon" BorderBrush="Black"
BorderThickness="1" DockPanel.Dock="Bottom">
            <TextBlock Foreground="Black">Dock = "Bottom"</TextBlock>
        </Border>
        <Border Width="200" Background="PaleGreen" BorderBrush="Black"
BorderThickness="1" DockPanel.Dock="Left">
            <TextBlock Foreground="Black">Dock = "Left"</TextBlock>
        </Border>
        <Border Background="White" BorderBrush="Black" BorderThickness="1">
            <TextBlock Foreground="Black">This content will "Fill" the remaining
space</TextBlock>
        </Border>
    </DockPanel>
</Page>
```

已编译的应用程序将生成如下所示的新 UI。



网格

[Grid](#) 元素将绝对位置的功能与表格数据控件合并。 使用 [Grid](#)，可轻松定位元素并为其设置样式。[Grid](#) 允许定义灵活的行和列分组，甚至提供在多个 [Grid](#) 元素之间共享大小调整信息的机制。

网格与表格的不同之处

[Table](#) 和 [Grid](#) 有一些共同的功能，但每个都有其各自最适合的场景。[Table](#) 设计为在流内容内使用（有关流内容的详细信息，请参阅[流文档概述](#)）。 网格最适合在表单中（主要在流内容以外的任意位置）使用。 在 [FlowDocument](#) 中，[Table](#) 支持分页、列重排和内容选择等流内容行为，而 [Grid](#) 不支持。 另一方面，[Grid](#) 最适合在 [FlowDocument](#) 之外使用，其原因有多种，例如 [Grid](#) 基于行和列索引添加元素，而 [Table](#) 不是。[Grid](#) 元素支持对子内容进行分层，从而允许多个元素共存于单个“单元格”内，而 [Table](#) 不支持分层。[Grid](#) 的子元素可相对于其“单元格”边界区域进行绝对定位。[Table](#) 不支持此功能。 最后，[Grid](#) 比 [Table](#) 更轻量。

列和行的大小调整行为

在 [Grid](#) 中定义的列和行可以充分利用 [Star](#) 大小调整来按比例分配剩余空间。 当选择 [Star](#) 作为行或列的 [Height](#) 或 [Width](#) 时，该列或行将收到剩余可用空间的加权比例。 这与 [Auto](#) 相反，后者根据列或行内的内容大小平均分配空间。 使用 Extensible Application Markup Language (XAML) 时，此值表示为 `*` 或 `2*`。 在第一种情况下，行或列将得到一倍的可用空间，在第二种情况下，将得到两倍的可用空间，依此类推。 通过将这一按比例分配空间的技术与 [Stretch](#) 的 [HorizontalAlignment](#) 和 [VerticalAlignment](#) 值结合使

用，可以按屏幕空间的百分比为布局空间分区。 Grid 是唯一可以按此方式分配空间的布局面板。

定义和使用 Grid

以下示例演示如何生成 UI，该 UI 类似于 Windows“开始”菜单上提供的“运行”对话框上的 UI。

C#

```
// Create the Grid.  
grid1 = new Grid();  
grid1.Background = Brushes.Gainsboro;  
grid1.HorizontalAlignment = HorizontalAlignment.Left;  
grid1.VerticalAlignment = VerticalAlignment.Top;  
grid1.ShowGridLines = true;  
grid1.Width = 425;  
grid1.Height = 165;  
  
// Define the Columns.  
colDef1 = new ColumnDefinition();  
colDef1.Width = new GridLength(1, GridUnitType.Auto);  
colDef2 = new ColumnDefinition();  
colDef2.Width = new GridLength(1, GridUnitType.Star);  
colDef3 = new ColumnDefinition();  
colDef3.Width = new GridLength(1, GridUnitType.Star);  
colDef4 = new ColumnDefinition();  
colDef4.Width = new GridLength(1, GridUnitType.Star);  
colDef5 = new ColumnDefinition();  
colDef5.Width = new GridLength(1, GridUnitType.Star);  
grid1.ColumnDefinitions.Add(colDef1);  
grid1.ColumnDefinitions.Add(colDef2);  
grid1.ColumnDefinitions.Add(colDef3);  
grid1.ColumnDefinitions.Add(colDef4);  
grid1.ColumnDefinitions.Add(colDef5);  
  
// Define the Rows.  
rowDef1 = new RowDefinition();  
rowDef1.Height = new GridLength(1, GridUnitType.Auto);  
rowDef2 = new RowDefinition();  
rowDef2.Height = new GridLength(1, GridUnitType.Auto);  
rowDef3 = new RowDefinition();  
rowDef3.Height = new GridLength(1, GridUnitType.Star);  
rowDef4 = new RowDefinition();  
rowDef4.Height = new GridLength(1, GridUnitType.Auto);  
grid1.RowDefinitions.Add(rowDef1);  
grid1.RowDefinitions.Add(rowDef2);  
grid1.RowDefinitions.Add(rowDef3);  
grid1.RowDefinitions.Add(rowDef4);  
  
// Add the Image.
```

```
img1 = new Image();
img1.Source = new System.Windows.Media.Imaging.BitmapImage(new
Uri("runicon.png", UriKind.Relative));
Grid.SetRow(img1, 0);
Grid.SetColumn(img1, 0);

// Add the main application dialog.
txt1 = new TextBlock();
txt1.Text = "Type the name of a program, folder, document, or Internet
resource, and Windows will open it for you.";
txt1.TextWrapping = TextWrapping.Wrap;
Grid.SetColumnSpan(txt1, 4);
Grid.SetRow(txt1, 0);
Grid.SetColumn(txt1, 1);

// Add the second text cell to the Grid.
txt2 = new TextBlock();
txt2.Text = "Open:";
Grid.SetRow(txt2, 1);
Grid.SetColumn(txt2, 0);

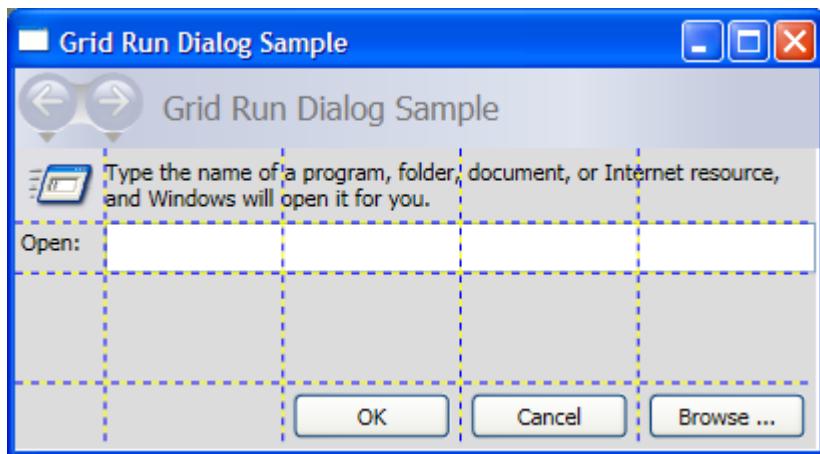
// Add the TextBox control.
tb1 = new TextBox();
Grid.SetRow(tb1, 1);
Grid.SetColumn(tb1, 1);
Grid.SetColumnSpan(tb1, 5);

// Add the buttons.
button1 = new Button();
button2 = new Button();
button3 = new Button();
button1.Content = "OK";
button2.Content = "Cancel";
button3.Content = "Browse ...";
Grid.SetRow(button1, 3);
Grid.SetColumn(button1, 2);
button1.Margin = new Thickness(10, 0, 10, 15);
button2.Margin = new Thickness(10, 0, 10, 15);
button3.Margin = new Thickness(10, 0, 10, 15);
Grid.SetRow(button2, 3);
Grid.SetColumn(button2, 3);
Grid.SetRow(button3, 3);
Grid.SetColumn(button3, 4);

grid1.Children.Add(img1);
grid1.Children.Add(txt1);
grid1.Children.Add(txt2);
grid1.Children.Add(tb1);
grid1.Children.Add(button1);
grid1.Children.Add(button2);
grid1.Children.Add(button3);

mainWindow.Content = grid1;
```

已编译的应用程序将生成如下所示的新 UI。



StackPanel

通过 [StackPanel](#)，可以在指定方向上“堆叠”元素。默认堆叠方向为垂直方向。

[Orientation](#) 属性可用于控制内容流。

StackPanel 与 DockPanel

尽管 [DockPanel](#) 也可以“堆叠”子元素，但 [DockPanel](#) 和 [StackPanel](#) 在某些使用情况下不会产生相似结果。例如，子元素的顺序可能影响它们在 [DockPanel](#) 中的大小，但不会影响这些元素在 [StackPanel](#) 中的大小。这是因为 [StackPanel](#) 在 [PositiveInfinity](#) 的堆叠方向上测量，而 [DockPanel](#) 仅测量可用大小。

以下示例演示此主要区别。

C#

```
// Create the application's main window
mainWindow = new Window();
mainWindow.Title = "StackPanel vs. DockPanel";

// Add root Grid
myGrid = new Grid();
myGrid.Width = 175;
myGrid.Height = 150;
RowDefinition myRowDef1 = new RowDefinition();
RowDefinition myRowDef2 = new RowDefinition();
myGrid.RowDefinitions.Add(myRowDef1);
myGrid.RowDefinitions.Add(myRowDef2);

// Define the DockPanel
myDockPanel = new DockPanel();
Grid.SetRow(myDockPanel, 0);
```

```

//Define an Image and Source
Image myImage = new Image();
BitmapImage bi = new BitmapImage();
bi.BeginInit();
bi.UriSource = new Uri("smiley_stackpanel.png", UriKind.Relative);
bi.EndInit();
myImage.Source = bi;

Image myImage2 = new Image();
BitmapImage bi2 = new BitmapImage();
bi2.BeginInit();
bi2.UriSource = new Uri("smiley_stackpanel.png", UriKind.Relative);
bi2.EndInit();
myImage2.Source = bi2;

Image myImage3 = new Image();
BitmapImage bi3 = new BitmapImage();
bi3.BeginInit();
bi3.UriSource = new Uri("smiley_stackpanel.PNG", UriKind.Relative);
bi3.EndInit();
myImage3.Stretch = Stretch.Fill;
myImage3.Source = bi3;

// Add the images to the parent DockPanel
myDockPanel.Children.Add(myImage);
myDockPanel.Children.Add(myImage2);
myDockPanel.Children.Add(myImage3);

//Define a StackPanel
myStackPanel = new StackPanel();
myStackPanel.Orientation = Orientation.Horizontal;
Grid.SetRow(myStackPanel, 1);

Image myImage4 = new Image();
BitmapImage bi4 = new BitmapImage();
bi4.BeginInit();
bi4.UriSource = new Uri("smiley_stackpanel.png", UriKind.Relative);
bi4.EndInit();
myImage4.Source = bi4;

Image myImage5 = new Image();
BitmapImage bi5 = new BitmapImage();
bi5.BeginInit();
bi5.UriSource = new Uri("smiley_stackpanel.png", UriKind.Relative);
bi5.EndInit();
myImage5.Source = bi5;

Image myImage6 = new Image();
BitmapImage bi6 = new BitmapImage();
bi6.BeginInit();
bi6.UriSource = new Uri("smiley_stackpanel.PNG", UriKind.Relative);
bi6.EndInit();
myImage6.Stretch = Stretch.Fill;
myImage6.Source = bi6;

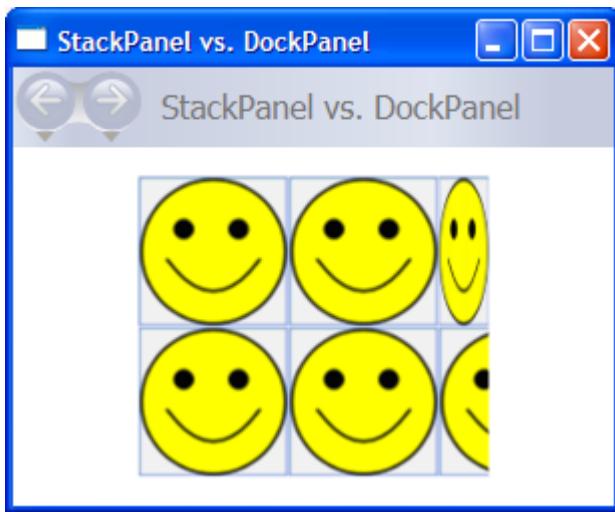
```

```
// Add the images to the parent StackPanel  
myStackPanel.Children.Add(myImage4);  
myStackPanel.Children.Add(myImage5);  
myStackPanel.Children.Add(myImage6);  
  
// Add the layout panels as children of the Grid  
myGrid.Children.Add(myDockPanel);  
myGrid.Children.Add(myStackPanel);  
  
// Add the Grid as the Content of the Parent Window Object  
mainWindow.Content = myGrid;  
mainWindow.Show();
```

XAML

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
      WindowTitle="StackPanel vs. DockPanel">  
  <Grid Width="175" Height="150">  
    <Grid.ColumnDefinitions>  
      <ColumnDefinition />  
    </Grid.ColumnDefinitions>  
    <Grid.RowDefinitions>  
      <RowDefinition />  
      <RowDefinition />  
    </Grid.RowDefinitions>  
  
    <DockPanel Grid.Column="0" Grid.Row="0">  
      <Image Source="smiley_stackpanel.png" />  
      <Image Source="smiley_stackpanel.png" />  
      <Image Source="smiley_stackpanel.png" Stretch="Fill"/>  
    </DockPanel>  
  
    <StackPanel Grid.Column="0" Grid.Row="1" Orientation="Horizontal">  
      <Image Source="smiley_stackpanel.png" />  
      <Image Source="smiley_stackpanel.png" />  
      <Image Source="smiley_stackpanel.png" Stretch="Fill"/>  
    </StackPanel>  
  </Grid>  
</Page>
```

通过此图像可以看到呈现行为的区别。



定义和使用 StackPanel

以下示例演示如何使用 [StackPanel](#) 创建一组垂直定位的按钮。对于水平定位，请将 [Orientation](#) 属性设置为 [Horizontal](#)。

C#

```
// Create the application's main window
mainWindow = new Window();
mainWindow.Title = "StackPanel Sample";

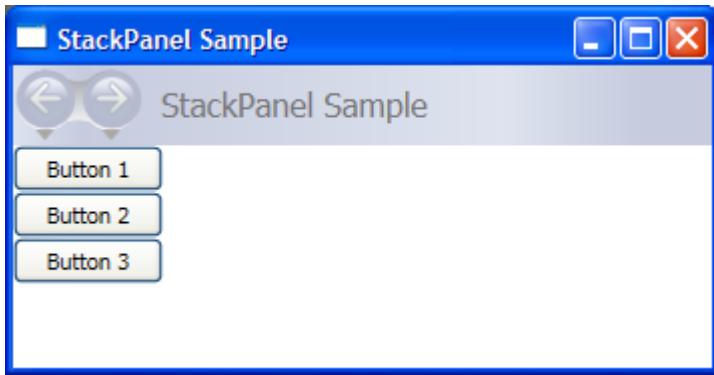
// Define the StackPanel
myStackPanel = new StackPanel();
myStackPanel.HorizontalAlignment = HorizontalAlignment.Left;
myStackPanel.VerticalAlignment = VerticalAlignment.Top;

// Define child content
Button myButton1 = new Button();
myButton1.Content = "Button 1";
Button myButton2 = new Button();
myButton2.Content = "Button 2";
Button myButton3 = new Button();
myButton3.Content = "Button 3";

// Add child elements to the parent StackPanel
myStackPanel.Children.Add(myButton1);
myStackPanel.Children.Add(myButton2);
myStackPanel.Children.Add(myButton3);

// Add the StackPanel as the Content of the Parent Window Object
mainWindow.Content = myStackPanel;
mainWindow.Show();
```

已编译的应用程序将生成如下所示的新 UI。



VirtualizingStackPanel

WPF 还提供 [StackPanel](#) 元素的一个变体，用于自动“虚拟化”数据绑定子级内容。在此上下文中，“虚拟化”一词指的是一种技术：通过此技术根据屏幕上哪些项可见，从较多的数据项中生成一个元素子集。如果在指定时刻只有少量 UI 元素位于屏幕上，则此时生成大量 UI 元素需要占用大量内存和处理器。[VirtualizingStackPanel](#)（通过 [VirtualizingPanel](#) 提供的功能）计算可见项，并与来自 [ItemsControl](#)（如 [ListBox](#) 或 [ListView](#)）的 [ItemContainerGenerator](#) 配合使用，以便仅为可见项创建元素。

[VirtualizingStackPanel](#) 元素自动设置为 [ListBox](#) 等控件的项宿主。在承载数据绑定集合时，只要内容在 [ScrollViewer](#) 的边界内，内容就将自动虚拟化。在承载许多子项时，这将大幅提高性能。

以下标记演示如何使用 [VirtualizingStackPanel](#) 作为项宿主。

[VirtualizingStackPanel.IsVirtualizingProperty](#) 附加属性必须设置为 `true`（默认值）才能进行虚拟化。

```
XAML

<StackPanel DataContext="{Binding Source={StaticResource Leagues}}">
    <TextBlock Text="{Binding XPath=@name}" FontFamily="Arial" FontSize="18"
    Foreground="Black"/>
    <ListBox VirtualizingStackPanel.IsVirtualizing="True"
        ItemsSource="{Binding XPath=Team}"
        ItemTemplate="{DynamicResource NameDataStyle}"/>
</StackPanel>
```

WrapPanel

[WrapPanel](#) 用于按从左到右的顺序位置定位子元素，在内容到达其父容器边缘时将其切换到下一行。内容可以设置为水平或垂直方向。[WrapPanel](#) 对于简单流用户界面 (UI) 方案很有用。它还可以用来将统一大小调整应用于其所有子元素。

以下示例演示如何创建一个 [WrapPanel](#) 来显示在其到达容器边缘时换行的 [Button](#) 控件。

C#

```
// Create the application's main window
mainWindow = new System.Windows.Window();
mainWindow.Title = "WrapPanel Sample";

// Instantiate a new WrapPanel and set properties
myWrapPanel = new WrapPanel();
myWrapPanel.Background = System.Windows.Media.Brushes.Azure;
myWrapPanel.Orientation = Orientation.Horizontal;
myWrapPanel.Width = 200;
myWrapPanel.HorizontalAlignment = HorizontalAlignment.Left;
myWrapPanel.VerticalAlignment = VerticalAlignment.Top;

// Define 3 button elements. The last three buttons are sized at width
// of 75, so the forth button wraps to the next line.
btn1 = new Button();
btn1.Content = "Button 1";
btn1.Width = 200;
btn2 = new Button();
btn2.Content = "Button 2";
btn2.Width = 75;
btn3 = new Button();
btn3.Content = "Button 3";
btn3.Width = 75;
btn4 = new Button();
btn4.Content = "Button 4";
btn4.Width = 75;

// Add the buttons to the parent WrapPanel using the Children.Add method.
myWrapPanel.Children.Add(btn1);
myWrapPanel.Children.Add(btn2);
myWrapPanel.Children.Add(btn3);
myWrapPanel.Children.Add(btn4);

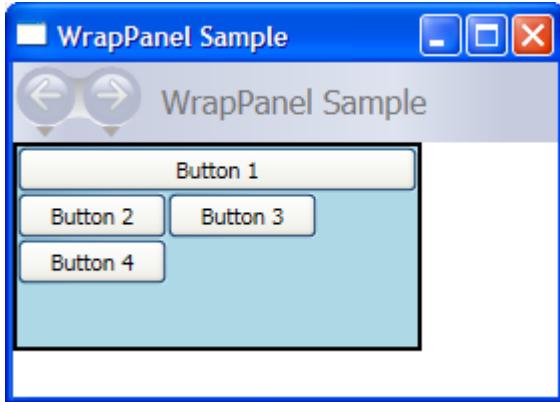
// Add the WrapPanel to the MainWindow as Content
mainWindow.Content = myWrapPanel;
mainWindow.Show();
```

XAML

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
WindowTitle="WrapPanel Sample">
    <Border HorizontalAlignment="Left" VerticalAlignment="Top"
BorderBrush="Black" BorderThickness="2">
        <WrapPanel Background="LightBlue" Width="200" Height="100">
            <Button Width="200">Button 1</Button>
            <Button>Button 2</Button>
            <Button>Button 3</Button>
            <Button>Button 4</Button>
```

```
</WrapPanel>
</Border>
</Page>
```

已编译的应用程序将生成如下所示的新 UI。



嵌套 Panel 元素

Panel 元素可以在彼此内嵌套，以便生成复杂的布局。在有一个 Panel 最适合 UI 的一个部分，但不满足 UI 其他部分需求的情况下，这可能非常有用。

对于应用程序可以支持的嵌套数量，并没有实际限制，但通常最好限制应用程序仅使用预期布局实际所需的面板。在许多情况下，由于 Grid 元素具有作为布局容器的灵活性，因此可以使用它代替嵌套面板。这可以通过将不必要的元素排除在树以外来提高应用程序的性能。

以下示例演示如何创建可充分利用嵌套 Panel 元素的 UI，以便实现特定的布局。在此特定情况下，DockPanel 元素用于提供 UI 结构，而嵌套 StackPanel 元素、Grid 和 Canvas 用于将子元素精确定位在父 DockPanel 内。

C#

```
// Define the DockPanel.
myDockPanel = new DockPanel();

// Add the Left Docked StackPanel
Border myBorder2 = new Border();
myBorder2.BorderThickness = new Thickness(1);
myBorder2.BorderBrush = Brushes.Black;
DockPanel1.SetDock(myBorder2, Dock.Left);
StackPanel myStackPanel = new StackPanel();
Button myButton1 = new Button();
myButton1.Content = "Left Docked";
myButton1.Margin = new Thickness(5);
Button myButton2 = new Button();
myButton2.Content = "StackPanel";
```

```

myButton2.Margin = new Thickness(5);
myStackPanel.Children.Add(myButton1);
myStackPanel.Children.Add(myButton2);
myBorder2.Child = myStackPanel;

// Add the Top Docked Grid.
Border myBorder3 = new Border();
myBorder3.BorderThickness = new Thickness(1);
myBorder3.BorderBrush = Brushes.Black;
DockPanel.SetDock(myBorder3, Dock.Top);
Grid myGrid = new Grid();
myGrid.ShowGridLines = true;
RowDefinition myRowDef1 = new RowDefinition();
RowDefinition myRowDef2 = new RowDefinition();
ColumnDefinition myColDef1 = new ColumnDefinition();
ColumnDefinition myColDef2 = new ColumnDefinition();
ColumnDefinition myColDef3 = new ColumnDefinition();
myGrid.ColumnDefinitions.Add(myColDef1);
myGrid.ColumnDefinitions.Add(myColDef2);
myGrid.ColumnDefinitions.Add(myColDef3);
myGrid.RowDefinitions.Add(myRowDef1);
myGrid.RowDefinitions.Add(myRowDef2);
TextBlock myTextBlock1 = new TextBlock();
myTextBlock1.FontSize = 20;
myTextBlock1.Margin = new Thickness(10);
myTextBlock1.Text = "Grid Element Docked at the Top";
Grid.SetRow(myTextBlock1, 0);
Grid.SetColumnSpan(myTextBlock1, 3);
Button myButton3 = new Button();
myButton3.Margin = new Thickness(5);
myButton3.Content = "A Row";
Grid.SetColumn(myButton3, 0);
Grid.SetRow(myButton3, 1);
Button myButton4 = new Button();
myButton4.Margin = new Thickness(5);
myButton4.Content = "of Button";
Grid.SetColumn(myButton4, 1);
Grid.SetRow(myButton4, 1);
Button myButton5 = new Button();
myButton5.Margin = new Thickness(5);
myButton5.Content = "Elements";
Grid.SetColumn(myButton5, 2);
Grid.SetRow(myButton5, 1);
myGrid.Children.Add(myTextBlock1);
myGrid.Children.Add(myButton3);
myGrid.Children.Add(myButton4);
myGrid.Children.Add(myButton5);
myBorder3.Child = myGrid;

// Add the Bottom Docked StackPanel.
Border myBorder4 = new Border();
myBorder4.BorderBrush = Brushes.Black;
myBorder4.BorderThickness = new Thickness(1);
DockPanel.SetDock(myBorder4, Dock.Bottom);
StackPanel myStackPanel2 = new StackPanel();

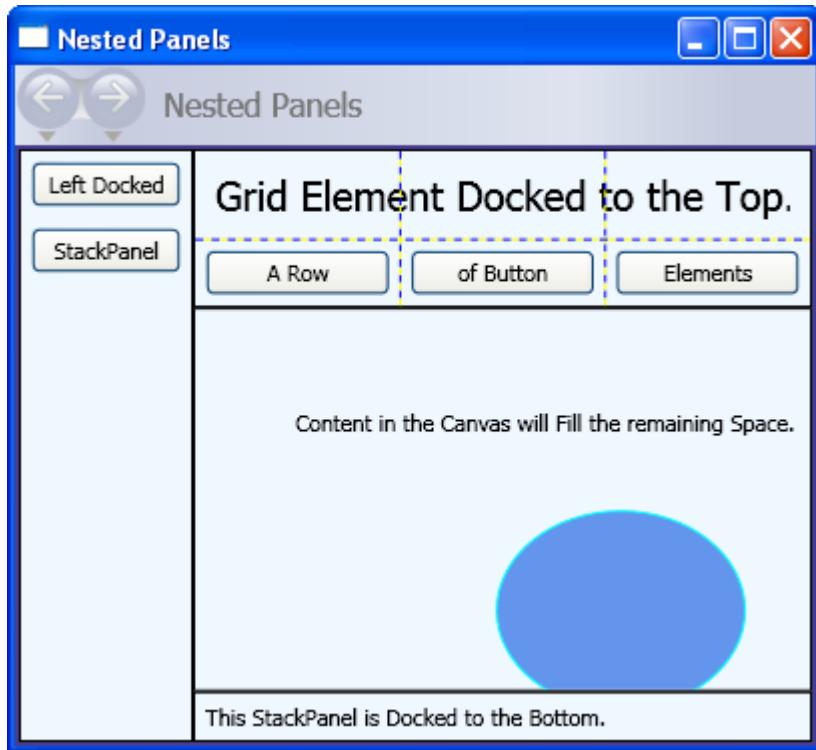
```

```
myStackPanel2.Orientation = Orientation.Horizontal;
TextBlock myTextBlock2 = new TextBlock();
myTextBlock2.Text = "This StackPanel is Docked to the Bottom";
myTextBlock2.Margin = new Thickness(5);
myStackPanel2.Children.Add(myTextBlock2);
myBorder4.Child = myStackPanel2;

// Add the Canvas, that fills remaining space.
Border myBorder5 = new Border();
myBorder4.BorderBrush = Brushes.Black;
myBorder5.BorderThickness = new Thickness(1);
Canvas myCanvas = new Canvas();
myCanvas.ClipToBounds = true;
TextBlock myTextBlock3 = new TextBlock();
myTextBlock3.Text = "Content in the Canvas will Fill the remaining space.";
Canvas.SetTop(myTextBlock3, 50);
Canvas.SetLeft(myTextBlock3, 50);
Ellipse myEllipse = new Ellipse();
myEllipse.Height = 100;
myEllipse.Width = 125;
myEllipse.Fill = Brushes.CornflowerBlue;
myEllipse.Stroke = Brushes.Aqua;
Canvas.SetTop(myEllipse, 100);
Canvas.SetLeft(myEllipse, 150);
myCanvas.Children.Add(myTextBlock3);
myCanvas.Children.Add(myEllipse);
myBorder5.Child = myCanvas;

// Add child elements to the parent DockPanel.
myDockPanel.Children.Add(myBorder2);
myDockPanel.Children.Add(myBorder3);
myDockPanel.Children.Add(myBorder4);
myDockPanel.Children.Add(myBorder5);
```

已编译的应用程序将生成如下所示的新 UI。



自定义 Panel 元素

尽管 WPF 提供了一系列灵活的布局控件，但通过重写 [ArrangeOverride](#) 和 [MeasureOverride](#) 方法也可以实现自定义布局行为。可以通过在这些重写方法内定义新的位置行为来实现自定义大小调整和位置。

同样，可以通过重写其 [ArrangeOverride](#) 和 [MeasureOverride](#) 方法定义基于派生类（如 [Canvas](#) 或 [Grid](#)）的自定义布局行为。

以下标记演示如何创建自定义 [Panel](#) 元素。这一定义为 [PlotPanel](#) 的新 [Panel](#) 支持通过使用硬编码的 x 和 y 坐标来定位子元素。在此示例中，[Rectangle](#) 元素（未显示）定位在 50 (x) 和 50 (y) 的绘图点。

C#

```
public class PlotPanel : Panel
{
    // Default public constructor
    public PlotPanel()
        : base()
    {
    }

    // Override the default Measure method of Panel
    protected override Size MeasureOverride(Size availableSize)
    {
        Size panelDesiredSize = new Size();

        // In our example, we just have one child.
    }
}
```

```

    // Report that our panel requires just the size of its only child.
    foreach (UIElement child in InternalChildren)
    {
        child.Measure(availableSize);
        panelDesiredSize = child.DesiredSize;
    }

    return panelDesiredSize ;
}

protected override Size ArrangeOverride(Size finalSize)
{
    foreach (UIElement child in InternalChildren)
    {
        double x = 50;
        double y = 50;

        child.Arrange(new Rect(new Point(x, y), child.DesiredSize));
    }
    return finalSize; // Returns the final Arranged size
}
}

```

若要查看更复杂的自定义面板实现，请参阅[创建自定义内容换行面板示例](#)。

本地化/全球化支持

UI。

所有面板元素本身支持 [FlowDirection](#) 属性，该属性可用于根据用户的区域设置或语言设置动态地重排内容。有关详细信息，请参阅 [FlowDirection](#)。

[SizeToContent](#) 属性提供使应用程序开发者可以预测本地化 UI 需求的机制。使用此属性的 [WidthAndHeight](#) 值，父 [Window](#) 始终可以动态调整大小以适应内容，并且不受人为的高度或宽度限制的约束。

[DockPanelGrid](#) 和 [StackPanel](#) 都是适用于可本地化 UI 的好选择。但是，[Canvas](#) 不是很好的选择，因为它以绝对方式定位内容，使其难以本地化。

有关创建带有可本地化用户界面 (UI) 的 WPF 应用程序的其他信息，请参阅[使用自动布局概述](#)。

另请参阅

- [演练：我的第一个 WPF 桌面应用程序](#)
- [WPF 布局库示例](#)
- [布局](#)

- [WPF 控件库示例 ↗](#)
- [Alignment、Margin 和 Padding 概述](#)
- [创建自定义内容换行面板示例](#)
- [附加属性概述](#)
- [使用自动布局概述](#)
- [布局和示例](#)

Panel 帮助主题

项目 • 2023/02/06

本部分中的主题介绍如何使用 [Panel](#) 元素及关联的 API。

本节内容

[创建自定义 Panel 元素](#)

[重写面板的 OnRender 方法](#)

[设置元素的高度属性](#)

[设置元素的宽度属性](#)

参考

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

相关章节

[布局](#)

[演练：我的第一个 WPF 桌面应用程序](#)

[ScrollViewer 概述](#)

如何：创建自定义 Panel 元素

项目 • 2023/02/06

示例

此示例演示如何替代 [Panel](#) 元素的默认布局行为，并创建派生自 [Panel](#) 的自定义布局元素。

示例定义了一个名为 `PlotPanel` 的简单自定义 [Panel](#) 元素，该元素根据两个硬编码的 x 坐标和 y 坐标定位子元素。在此示例中，`x` 和 `y` 均设置为 `50`；因此，所有子元素都位于 x 和 y 轴上的该位置。

为了实现自定义 [Panel](#) 行为，该示例使用 `MeasureOverride` 和 `ArrangeOverride` 方法。每个方法都返回 `Size` 定位和呈现子元素所需的数据。

C#

```
public class PlotPanel : Panel
{
    // Default public constructor
    public PlotPanel()
        : base()
    {

    }

    // Override the default Measure method of Panel
    protected override Size MeasureOverride(Size availableSize)
    {
        Size panelDesiredSize = new Size();

        // In our example, we just have one child.
        // Report that our panel requires just the size of its only child.
        foreach (UIElement child in InternalChildren)
        {
            child.Measure(availableSize);
            panelDesiredSize = child.DesiredSize;
        }

        return panelDesiredSize ;
    }
    protected override Size ArrangeOverride(Size finalSize)
    {
        foreach (UIElement child in InternalChildren)
        {
            double x = 50;
            double y = 50;

            child.Arrange(new Rect(new Point(x, y), child.DesiredSize));
        }
    }
}
```

```
        }
        return finalSize; // Returns the final Arranged size
    }
}
```

另请参阅

- [Panel](#)
- [面板概述](#)

如何：重写面板的 OnRender 方法

项目 • 2023/02/06

此示例演示如何替代 [Panel](#) 的 [OnRender](#) 方法以将自定义图形效果添加到布局元素。

示例

使用 [OnRender](#) 方法以将图形效果添加到呈现面板元素。例如，可以使用此方法添加自定义边框或背景效果。[DrawingContext](#) 对象作为参数传递，它提供用于绘制形状、文本、图像或视频的方法。因此，此方法可用于自定义面板对象。

C#

```
// Override the OnRender call to add a Background and Border to the
OffSetPanel
protected override void OnRender(DrawingContext dc)
{
    SolidColorBrush mySolidColorBrush = new SolidColorBrush();
    mySolidColorBrush.Color = Colors.LimeGreen;
    Pen myPen = new Pen(Brushes.Blue, 10);
    Rect myRect = new Rect(0, 0, 500, 500);
    dc.DrawRectangle(mySolidColorBrush, myPen, myRect);
}
```

另请参阅

- [Panel](#)
- [面板概述](#)
- [操作指南主题](#)

如何：设置元素的高度属性

项目 • 2023/02/06

示例

此示例直观显示 Windows Presentation Foundation (WPF) 中与高度相关的四个属性之间的呈现行为差异。

`FrameworkElement` 类公开描述元素高度特征的四个属性。这四个属性可能会产生冲突；冲突发生时，优先的值按如下方式确定：`MinHeight` 值优先于 `MaxHeight` 值，而后者又优先于 `Height` 值。第四个属性 `ActualHeight` 为只读，并报告在布局的交互过程中确定的实际高度。

以下 Extensible Application Markup Language (XAML) 示例将 `Rectangle` 元素 (`rect1`) 绘制为 `Canvas` 的子元素。可以使用一系列代表 `MinHeight`、`MaxHeight` 和 `Height` 的属性值的 `ListBox` 元素来更改 `Rectangle` 的高度属性。以这种方式，可以直观地显示每个属性的优先级。

XAML

```
<Canvas Height="200" MinWidth="200" Background="#b0c4de"
VerticalAlignment="Top" HorizontalAlignment="Center" Name="myCanvas"
Margin="0,0,0,50">
    <Rectangle HorizontalAlignment="Center" Canvas.Top="50" Canvas.Left="50"
Name="rect1" Fill="#4682b4" Height="100" Width="100"/>
</Canvas>
```

XAML

```
<TextBlock Grid.Row="1" Grid.Column="0" Margin="10,0,0,0"
TextWrapping="Wrap">Set the Rectangle Height:</TextBlock>
<ListBox Grid.Column="1" Grid.Row="1" Margin="10,0,0,0" Height="50"
Width="50" SelectionChanged="changeHeight">
    <ListBoxItem>25</ListBoxItem>
    <ListBoxItem>50</ListBoxItem>
    <ListBoxItem>75</ListBoxItem>
    <ListBoxItem>100</ListBoxItem>
    <ListBoxItem>125</ListBoxItem>
    <ListBoxItem>150</ListBoxItem>
    <ListBoxItem>175</ListBoxItem>
    <ListBoxItem>200</ListBoxItem>
</ListBox>

<TextBlock Grid.Row="1" Grid.Column="2" Margin="10,0,0,0"
TextWrapping="Wrap">Set the Rectangle MinHeight:</TextBlock>
<ListBox Grid.Column="3" Grid.Row="1" Margin="10,0,0,0" Height="50"
```

```

Width="50" SelectionChanged="changeMinHeight">
<ListBoxItem>25</ListBoxItem>
<ListBoxItem>50</ListBoxItem>
<ListBoxItem>75</ListBoxItem>
<ListBoxItem>100</ListBoxItem>
<ListBoxItem>125</ListBoxItem>
<ListBoxItem>150</ListBoxItem>
<ListBoxItem>175</ListBoxItem>
<ListBoxItem>200</ListBoxItem>
</ListBox>

<TextBlock Grid.Row="1" Grid.Column="4" Margin="10,0,0,0"
TextWrapping="Wrap">Set the Rectangle MaxHeight:</TextBlock>
<ListBox Grid.Column="5" Grid.Row="1" Margin="10,0,0,0" Height="50"
Width="50" SelectionChanged="changeMaxHeight">
<ListBoxItem>25</ListBoxItem>
<ListBoxItem>50</ListBoxItem>
<ListBoxItem>75</ListBoxItem>
<ListBoxItem>100</ListBoxItem>
<ListBoxItem>125</ListBoxItem>
<ListBoxItem>150</ListBoxItem>
<ListBoxItem>175</ListBoxItem>
<ListBoxItem>200</ListBoxItem>
</ListBox>

```

以下代码隐藏示例处理 `SelectionChanged` 事件引发的事件。每个处理程序从 `ListBox` 获得输入，将值分析为 `Double`，并将值应用于指定的高度相关属性。高度值也会转换为字符串，并写入各种 `TextBlock` 元素（这些元素的定义未显示在所选 XAML 中）。

C#

```

private void changeHeight(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    Double sz1 = Double.Parse(li.Content.ToString());
    rect1.Height = sz1;
    rect1.UpdateLayout();
    txt1.Text= "ActualHeight is set to " + rect1.ActualHeight;
    txt2.Text= "Height is set to " + rect1.Height;
    txt3.Text= "MinHeight is set to " + rect1.MinHeight;
    txt4.Text= "MaxHeight is set to " + rect1.MaxHeight;
}
private void changeMinHeight(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    Double sz1 = Double.Parse(li.Content.ToString());
    rect1.MinHeight = sz1;
    rect1.UpdateLayout();
    txt1.Text= "ActualHeight is set to " + rect1.ActualHeight;
    txt2.Text= "Height is set to " + rect1.Height;
    txt3.Text= "MinHeight is set to " + rect1.MinHeight;
    txt4.Text= "MaxHeight is set to " + rect1.MaxHeight;
}

```

```
private void changeMaxHeight(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    Double sz1 = Double.Parse(li.Content.ToString());
    rect1.MaxHeight = sz1;
    rect1.UpdateLayout();
    txt1.Text= "ActualHeight is set to " + rect1.ActualHeight;
    txt2.Text= "Height is set to " + rect1.Height;
    txt3.Text= "MinHeight is set to " + rect1.MinHeight;
    txt4.Text= "MaxHeight is set to " + rect1.MaxHeight;
}
```

有关完整示例，请参阅[高度属性示例](#)。

另请参阅

- [FrameworkElement](#)
- [ListBox](#)
- [ActualHeight](#)
- [MaxHeight](#)
- [MinHeight](#)
- [Height](#)
- [设置元素的宽度属性](#)
- [面板概述](#)
- [高度属性示例](#)

如何：设置元素的宽度属性

项目 • 2023/02/06

示例

此示例直观地显示了 Windows Presentation Foundation (WPF) 中四个与宽度相关的属性在呈现行为方面的差异。

`FrameworkElement` 类公开了描述元素宽度特征的四个属性。这四个属性可能会产生冲突；冲突发生时，优先的值按如下方式确定：`MinWidth` 值优先于 `MaxWidth` 值，而后者又优先于 `Width` 值。第四个属性 `ActualWidth` 为只读，并报告在布局的交互过程中确定的实际宽度。

以下 Extensible Application Markup Language (XAML) 示例将 `Rectangle` 元素 (`rect1`) 绘制为 `Canvas` 的子元素。可以使用一系列代表 `MinWidth`、`MaxWidth` 和 `Width` 的属性值的 `ListBox` 元素来更改 `Rectangle` 的宽度属性。以这种方式，可以直观地显示每个属性的优先级。

XAML

```
<Canvas Height="200" MinWidth="200" Background="#b0c4de"
VerticalAlignment="Top" HorizontalAlignment="Center" Name="myCanvas">
    <Rectangle HorizontalAlignment="Center" Canvas.Top="50" Canvas.Left="50"
Name="rect1" Fill="#4682b4" Width="100" Height="100"/>
</Canvas>
```

XAML

```
<TextBlock Grid.Row="1" Grid.Column="0" Margin="10,0,0,0"
TextWrapping="Wrap">Set the Rectangle Width:</TextBlock>
<ListBox Grid.Column="1" Grid.Row="1" Margin="10,0,0,0" Width="50"
Height="50" SelectionChanged="changeWidth">
    <ListBoxItem>25</ListBoxItem>
    <ListBoxItem>50</ListBoxItem>
    <ListBoxItem>75</ListBoxItem>
    <ListBoxItem>100</ListBoxItem>
    <ListBoxItem>125</ListBoxItem>
    <ListBoxItem>150</ListBoxItem>
    <ListBoxItem>175</ListBoxItem>
    <ListBoxItem>200</ListBoxItem>
    <ListBoxItem>225</ListBoxItem>
    <ListBoxItem>250</ListBoxItem>
</ListBox>

<TextBlock Grid.Row="1" Grid.Column="2" Margin="10,0,0,0"
TextWrapping="Wrap">Set the Rectangle MinWidth:</TextBlock>
```

```

<ListBox Grid.Column="3" Grid.Row="1" Margin="10,0,0,0" Width="50"
Height="50" SelectionChanged="changeMinWidth">
    <ListBoxItem>25</ListBoxItem>
    <ListBoxItem>50</ListBoxItem>
    <ListBoxItem>75</ListBoxItem>
    <ListBoxItem>100</ListBoxItem>
    <ListBoxItem>125</ListBoxItem>
    <ListBoxItem>150</ListBoxItem>
    <ListBoxItem>175</ListBoxItem>
    <ListBoxItem>200</ListBoxItem>
    <ListBoxItem>225</ListBoxItem>
    <ListBoxItem>250</ListBoxItem>
</ListBox>

<TextBlock Grid.Row="1" Grid.Column="4" Margin="10,0,0,0"
TextWrapping="Wrap">Set the Rectangle MaxWidth:</TextBlock>
<ListBox Grid.Column="5" Grid.Row="1" Margin="10,0,0,0" Width="50"
Height="50" SelectionChanged="changeMaxWidth">
    <ListBoxItem>25</ListBoxItem>
    <ListBoxItem>50</ListBoxItem>
    <ListBoxItem>75</ListBoxItem>
    <ListBoxItem>100</ListBoxItem>
    <ListBoxItem>125</ListBoxItem>
    <ListBoxItem>150</ListBoxItem>
    <ListBoxItem>175</ListBoxItem>
    <ListBoxItem>200</ListBoxItem>
    <ListBoxItem>225</ListBoxItem>
    <ListBoxItem>250</ListBoxItem>
</ListBox>

```

以下代码隐藏示例处理 `SelectionChanged` 事件引发的事件。每个自定义方法从 `ListBox` 获取输入，将值分析为 `Double`，并将值应用于指定的与宽度相关的属性。宽度值也会转换为字符串，并写入各种 `TextBlock` 元素（这些元素的定义未显示在所选 XAML 中）。

C#

```

private void changeWidth(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    Double sz1 = Double.Parse(li.Content.ToString());
    rect1.Width = sz1;
    rect1.UpdateLayout();
    txt1.Text = "ActualWidth is set to " + rect1.ActualWidth;
    txt2.Text = "Width is set to " + rect1.Width;
    txt3.Text = "MinWidth is set to " + rect1.MinWidth;
    txt4.Text = "MaxWidth is set to " + rect1.MaxWidth;
}
private void changeMinWidth(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    Double sz1 = Double.Parse(li.Content.ToString());
    rect1.MinWidth = sz1;
    rect1.UpdateLayout();
}

```

```
txt1.Text = "ActualWidth is set to " + rect1.ActualWidth;
txt2.Text = "Width is set to " + rect1.Width;
txt3.Text = "MinWidth is set to " + rect1.MinWidth;
txt4.Text = "MaxWidth is set to " + rect1.MaxWidth;
}
private void changeMaxWidth(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    Double sz1 = Double.Parse(li.Content.ToString());
    rect1.MaxWidth = sz1;
    rect1.UpdateLayout();
    txt1.Text = "ActualWidth is set to " + rect1.ActualWidth;
    txt2.Text = "Width is set to " + rect1.Width;
    txt3.Text = "MinWidth is set to " + rect1.MinWidth;
    txt4.Text = "MaxWidth is set to " + rect1.MaxWidth;
}
```

有关完整示例，请参阅[宽度属性比较示例](#)。

另请参阅

- [ListBox](#)
- [FrameworkElement](#)
- [ActualWidth](#)
- [MaxWidth](#)
- [MinWidth](#)
- [Width](#)
- [面板概述](#)
- [设置元素的高度属性](#)
- [宽度属性比较示例](#)

PasswordBox

项目 • 2023/02/06

[PasswordBox](#) 控件用于输入敏感信息或专用信息。

另请参阅

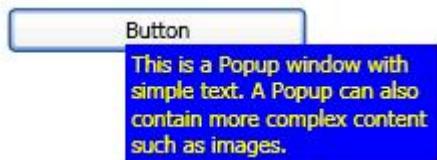
- [TextBox](#)
- [RichTextBox](#)
- [控件库](#)

弹出项

项目 • 2023/02/06

[Popup](#) 控件在一个浮动在当前应用程序窗口上的单独窗口中显示内容。

下图显示了一个 [Popup](#) 控件，该控件位于相对于 [Button](#) (即其父级) 的位置：



本节内容

[Popup 概述](#)

[Popup 放置行为](#)

[操作指南主题](#)

参考

[Popup](#)

相关章节

Popup 概述

项目 • 2023/02/06

Popup 控件提供一种在单独窗口中显示内容的方法，该窗口相对于指定元素或屏幕坐标在当前应用程序窗口上浮动。本主题介绍 Popup 控件，并提供关于其用法的信息。

什么是 Popup？

Popup 控件在相对于元素或屏幕上的点的单独窗口中显示内容。当 Popup 可见时，`IsOpen` 属性设置为 `true`。

① 备注

当鼠标指针移动到 Popup 的父对象上方时，它不会自动打开。如果要让 Popup 自动打开，请使用 `ToolTip` 或 `ToolTipService` 类。有关详细信息，请参阅 [ToolTip 概述](#)。

创建弹出项。

以下示例演示如何定义作为 `ToggleButton` 控件子元素的 `Popup` 控件。因为 `ToggleButton` 只能有一个子元素，所以此示例将 `ToggleButton` 和 `Popup` 控件的文本放置在 `StackPanel` 中。`Popup` 的内容显示在一个单独的窗口中，该窗口浮动在相关 `ToggleButton` 控件附近的应用程序窗口上方。

XAML

```
<ToggleButton x:Name="TogglePopupButton" Height="30" Width="150"
HorizontalAlignment="Left">
    <StackPanel>
        <TextBlock VerticalAlignment="Center" HorizontalAlignment="Center">
            <Run Text="Is button toggled? " />
            <Run Text="{Binding IsChecked, ElementName=TogglePopupButton}" />
        </TextBlock>

        <Popup Name="myPopup" IsOpen="{Binding IsChecked, ElementName=TogglePopupButton}">
            <Border BorderThickness="1">
                <TextBlock Name="myPopupText" Background="LightBlue" Foreground="Blue" Padding="30">
                    Popup Text
                </TextBlock>
            </Border>
        </Popup>
    </StackPanel>
</ToggleButton>
```

```
</Popups>
</StackPanel>
</ToggleButton>
```

实现 Popup 的控件

可以在其他控件中生成 [Popup](#) 控件。以下控件针对特定用途实现 [Popup](#) 控件：

- [ToolTip](#). 如果要为元素创建工具提示，请使用 [ToolTip](#) 和 [ToolTipService](#) 类。有关详细信息，请参阅 [ToolTip 概述](#)。
- [ContextMenu](#). 如果要为元素创建上下文菜单，请使用 [ContextMenu](#) 控件。有关详细信息，请参阅 [ContextMenu 概述](#)。
- [ComboBox](#). 如果要创建具有可显示或隐藏的下拉列表框的选择控件，请使用 [ComboBox](#) 控件。
- [Expander](#). 如果要创建显示标题（该标题带有显示内容的可折叠区域）的控件，请使用 [Expander](#) 控件。有关详细信息，请参阅 [Expander 概述](#)。

Popup 行为和外观

[Popup](#) 控件提供可用于自定义其行为和外观的功能。例如，可以设置打开和关闭行为、动画和位图效果以及 [Popup](#) 大小和位置。

打开和关闭行为

当 [IsOpen](#) 属性设置为 `true` 时，[Popup](#) 控件显示其内容。默认情况下，[Popup](#) 保持打开状态，直到 [IsOpen](#) 属性设置为 `false`。但是，可以通过将 [StaysOpen](#) 属性设置为 `false` 来更改默认行为。将此属性设置为 `false` 时，[Popup](#) 内容窗口具有鼠标捕获。在 [Popup](#) 窗口外发生鼠标事件时，[Popup](#) 失去鼠标捕获，并且窗口关闭。

当打开或关闭 [Popup](#) 内容窗口时，将引发 [Opened](#) 和 [Closed](#) 事件。

动画

[Popup](#) 控件为通常与淡入和滑入之类的行为关联的动画提供内置支持。可以通过将 [PopupAnimation](#) 属性设置为 [PopupAnimation](#) 枚举值来打开这些动画。若要使 [Popup](#) 动画正常工作，必须将 [AllowsTransparency](#) 属性设置为 `true`。

还可以将 [Storyboard](#) 之类的动画应用到 [Popup](#) 控件。

不透明度和位图效果

[Popup](#) 控件的 [Opacity](#) 属性对其内容不产生任何影响。默认情况下，[Popup](#) 内容窗口不透明。若要创建透明的 [Popup](#)，请将 [AllowsTransparency](#) 属性设置为 `true`。

[Popup](#) 的内容不会继承位图效果（例如 [DropShadowBitmapEffect](#)），这样便可以直接在 [Popup](#) 控件或父窗口中的任何其他元素上进行设置。若要使位图效果在 [Popup](#) 的内容上显示，必须在其内容上直接设置位图效果。例如，如果 [Popup](#) 的子级是 [StackPanel](#)，请在 [StackPanel](#) 上设置位图效果。

Popup 大小

默认情况下，[Popup](#) 根据其内容自动调整大小。由于为 [Popup](#) 内容定义的屏幕区域的默认大小的空间有限，不能为要显示的位图效果提供足够的空间，因此在自动调整大小时，某些位图效果可能会隐藏。

在内容上设置 [RenderTransform](#) 时，[Popup](#) 的内容也可能被遮挡住。在这种情况下，如果转换的 [Popup](#) 的内容超出了原始 [Popup](#) 的区域，则某些内容可能会隐藏。如果位图效果或转换需要更多空间，可以在 [Popup](#) 内容周围定义一个边距，从而为该控件提供更多区域。

定义 Popup 位置

可通过设置 [PlacementTarget](#)、[PlacementRectangle](#)、[Placement](#)、[HorizontalOffset](#) 和 [VerticalOffsetProperty](#) 属性来定位 [Popup](#)。有关详细信息，请参阅 [Popup 放置行为](#)。当 [Popup](#) 已显示在屏幕上时，如果其父级已重新定位，则其本身不会重新定位。

自定义 Popup 放置

可以通过在希望显示 [Popup](#) 的位置指定一组相对于 [PlacementTarget](#) 的坐标，来自定义 [Popup](#) 控件的位置。

若要自定义位置，请将 [Placement](#) 属性设置为 [Custom](#)。然后定义一个 [CustomPopupPlacementCallback](#) 委托，该委托返回 [Popup](#) 的一组可能的放置点和主轴（按优先顺序排列）。会自动选中显示 [Popup](#) 最大部分的点。有关示例，请参阅[指定自定义 Popup 位置](#)。

Popup 和可视化树

[Popup](#) 控件没有自己的可视化树；当调用 [Popup](#) 的 [MeasureOverride](#) 方法时，它返回 0（零）大小。但是，将 [Popup](#) 的 [IsOpen](#) 属性设置为 `true` 时，将创建一个具有自己的

可视化树的新窗口。 新窗口包含 [Popup](#) 的 [Child](#) 内容。 新窗口的宽度和高度不能超过屏幕宽度或高度的 75%。

[Popup](#) 控件保留对将其 [Child](#) 内容作为逻辑子级的引用。 创建新窗口后，[Popup](#) 的内容成为该窗口的一个可视化子级并保留 [Popup](#) 的逻辑子级。 相反，[Popup](#) 将保留其 [Child](#) 内容的逻辑父级。

另请参阅

- [Popup](#)
- [PopupPrimaryAxis](#)
- [PlacementMode](#)
- [CustomPopupPlacement](#)
- [CustomPopupPlacementCallback](#)
- [ToolTip](#)
- [ToolTipService](#)
- [操作指南主题](#)
- [操作指南主题](#)

Popup 放置行为

项目 • 2023/02/06

Popup 控件在一个浮动在应用程序上的单独窗口中显示内容。可通过使用 PlacementTarget、Placement、PlacementRectangle、HorizontalOffset 和 VerticalOffset 属性来指定 Popup 相对于控件、鼠标或屏幕的位置。这些属性协同工作，使你可以灵活地指定 Popup 的位置。

① 备注

ToolTip 和 ContextMenu 类还定义了这五个属性，并且两者的行为相似。

定位 Popup

可将 Popup 相对于 UIElement 或整个屏幕进行放置。以下示例创建了四个相对于 UIElement（在本例中，它是一个图像）的 Popup 控件。所有这些 Popup 控件都将 PlacementTarget 属性设置为 image1，但每个 Popup 具有不同的 Placement 属性值。

XAML

```
<Canvas Width="200" Height="150">
    <Image Name="image1"
        Canvas.Left="75"
        Source="Water_lilies.jpg" Height="200" Width="200"/>
    <Popup IsOpen="True" PlacementTarget="{Binding ElementName=image1}"
        Placement="Bottom">
        <TextBlock FontSize="14" Background="LightGreen">Placement=Bottom</TextBlock>
    </Popup>
    <Popup IsOpen="True" PlacementTarget="{Binding ElementName=image1}"
        Placement="Top">
        <TextBlock FontSize="14" Background="LightGreen">Placement=Top</TextBlock>
    </Popup>
    <Popup IsOpen="True" PlacementTarget="{Binding ElementName=image1}"
        Placement="Left">
        <TextBlock FontSize="14" Background="LightGreen">Placement=Left</TextBlock>
    </Popup>
    <Popup IsOpen="True" PlacementTarget="{Binding ElementName=image1}"
        Placement="Right">
        <TextBlock FontSize="14" Background="LightGreen">Placement=Right</TextBlock>
    </Popup>
</Canvas>
```

下图显示了该图像和这些 Popup 控件



这一简单的示例演示了如何设置 `PlacementTarget` 和 `Placement` 属性，但你可通过使用 `PlacementRectangle`、`HorizontalOffset` 和 `VerticalOffset` 属性，更好地控制 `Popup` 的位置。

术语定义：Popup 解析

对于理解 `PlacementTarget`、`Placement`、`PlacementRectangle`、`HorizontalOffset` 和 `VerticalOffset` 属性彼此关联的方式以及它们与 `Popup` 关联的方式，以下术语非常实用：

- “目标对象”
- 目标区域
- 目标原点
- `Popup` 对齐点

这些术语为引用 `Popup` 的各个方面及其关联的控件提供了一种便捷的方法。

目标对象

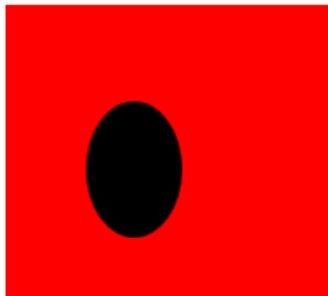
目标对象是与 `Popup` 关联的元素。如果已设置 `PlacementTarget` 属性，则它指定目标对象。如果未设置 `PlacementTarget` 并且 `Popup` 具有父级，则父级就是目标对象。如果没有 `PlacementTarget` 值并且没有父级，则没有目标对象并且 `Popup` 相对于屏幕进行定位。

以下示例创建了一个作为 `Canvas` 的子级的 `Popup`。该示例未对 `Popup` 设置 `PlacementTarget` 属性。`Placement` 的默认值为 `PlacementMode.Bottom`，因此 `Popup` 在 `Canvas` 下方显示。

XAML

```
<Canvas Margin="5" Background="Red" Width="200" Height="150" >  
  
    <Ellipse Canvas.Top="60" Canvas.Left="50"  
        Height="85" Width="60"  
        Fill="Black"/>  
  
    <Popup IsOpen="True" >  
        <TextBlock Background="LightBlue" FontSize="18">This is a Popup</TextBlock>  
    </Popup>  
</Canvas>
```

下图显示了 `Popup` 相对于 `Canvas` 进行定位。



以下示例创建了一个作为 `Canvas` 的子级的 `Popup`，但这一次将 `PlacementTarget` 设置为 `ellipse1`，因此 `Popup` 在 `Ellipse` 下方显示。

XAML

```

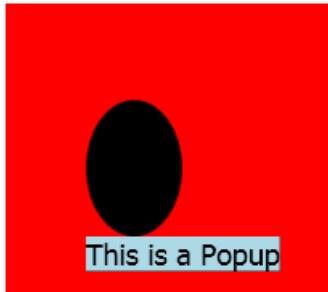
<Canvas Margin="5" Background="Red" Width="200" Height="150" >

    <Ellipse Name="ellipse1"
        Canvas.Top="60" Canvas.Left="50"
        Height="85" Width="60"
        Fill="Black"/>

    <Popup IsOpen="True" PlacementTarget="{Binding ElementName=ellipse1}">
        <TextBlock Background="LightBlue" FontSize="18">This is a Popup</TextBlock>
    </Popup>
</Canvas>

```

下图显示了 [Popup](#) 相对于 [Ellipse](#) 进行定位。



的弹出窗口

① 备注

对于 [ToolTip](#) , [Placement](#) 的默认值是 [Mouse](#)。对于 [ContextMenu](#) , [Placement](#) 的默认值是 [MousePoint](#)。这些值将在后面的“属性如何协同工作”部分中进行说明。

目标区域

目标区域是屏幕上与 [Popup](#) 相对的区域。在前面的示例中，[Popup](#) 与目标对象的边界对齐，但在某些情况下，即使 [Popup](#) 具有目标对象，[Popup](#) 也将与其他边界对齐。如果已设置 [PlacementRectangle](#) 属性，则目标区域与目标对象的边界不同。

以下示例创建了两个 [Canvas](#) 对象，每个对象包含一个 [Rectangle](#) 和一个 [Popup](#)。在这两种情况下，[Popup](#) 的目标对象都是 [Canvas](#)。第一个 [Canvas](#) 中的 [Popup](#) 设置了 [PlacementRectangle](#)，其 [X](#)、[Y](#)、[Width](#) 和 [Height](#) 属性分别设置为 50、50、50 和 100。第二个 [Canvas](#) 中的 [Popup](#) 未设置 [PlacementRectangle](#)。因此，第一个 [Popup](#) 位于 [PlacementRectangle](#) 下方，而第二个 [Popup](#) 位于 [Canvas](#) 下方。每个 [Canvas](#) 还包含一个 [Rectangle](#)，它与第一个 [Popup](#) 的 [PlacementRectangle](#) 具有相同的边界。请注意，[PlacementRectangle](#) 不会在应用程序中创建可见元素；该示例创建了一个表示 [PlacementRectangle](#) 的 [Rectangle](#)。

XAML

```

<StackPanel Orientation="Horizontal" Margin="50,50,0,0">

    <Canvas Width="200" Height="200" Background="Red">
        <Rectangle Canvas.Top="50" Canvas.Left="50"
            Width="50" Height="100"
            Stroke="White" StrokeThickness="3"/>
        <Popup IsOpen="True" PlacementRectangle="50,50,50,100">
            <TextBlock FontSize="14" Background="Yellow"
                Width="140" TextWrapping="Wrap">
                This is a popup with a PlacementRectangle.
            </TextBlock>
        </Popup>
    </Canvas>

```

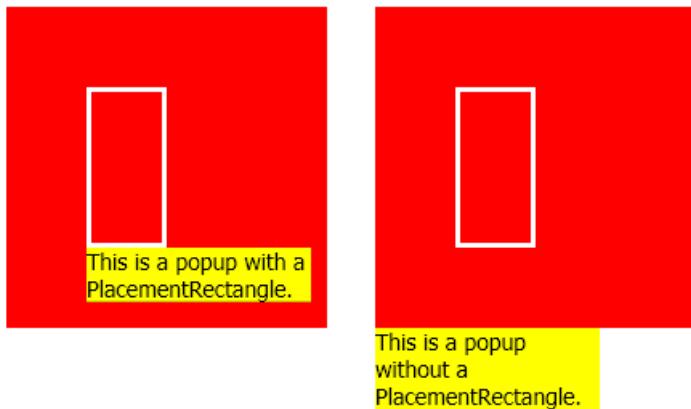
```

<Canvas Width="200" Height="200" Background="Red" Margin="30,0,0,0">
    <Rectangle Canvas.Top="50" Canvas.Left="50"
        Width="50" Height="100"
        Stroke="White" StrokeThickness="3"/>
    <Popup IsOpen="True">
        <TextBlock FontSize="14" Background="Yellow"
            Width="140" TextWrapping="Wrap">
            This is a popup without a PlacementRectangle.
        </TextBlock>
    </Popup>
</Canvas>

</StackPanel>

```

下图显示前面示例的结果。



目标原点和目标对齐点

目标原点和 **Popup** 对齐点分别是目标区域和 **Popup** 上用于定位定位的参照点。可使用 **HorizontalOffset** 和 **VerticalOffset** 属性使 **Popup** 从目标区域偏移。**HorizontalOffset** 和 **VerticalOffset** 相对于目标原点和 **Popup** 对齐点。**Placement** 属性的值确定目标原点和 **Popup** 对齐点的位置。

以下示例创建了一个 **Popup**，并将 **HorizontalOffset** 和 **VerticalOffset** 属性设置为 20。**Placement** 属性设置为 **Bottom**（默认值），因此目标原点是目标区域的左下角，而 **Popup** 对齐点是 **Popup** 的左上角。

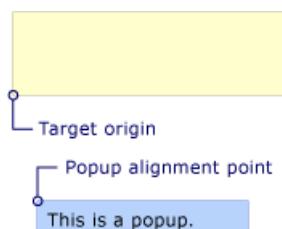
XAML

```

<Canvas Width="200" Height="200" Background="Yellow" Margin="20">
    <Popups IsOpen="True" Placement="Bottom"
        HorizontalOffset="20" VerticalOffset="20">
        <TextBlock FontSize="14" Background="#42F3FD">
            This is a popup.
        </TextBlock>
    </Popups>
</Canvas>

```

下图显示前面示例的结果。



属性如何协同工作

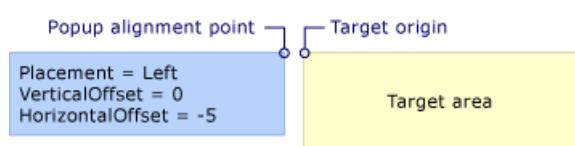
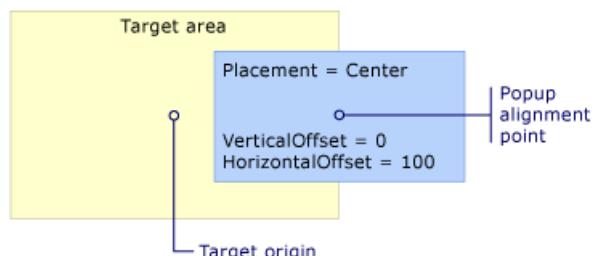
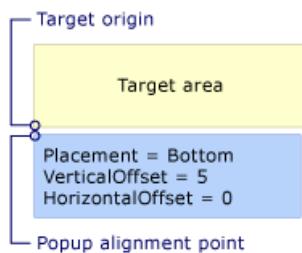
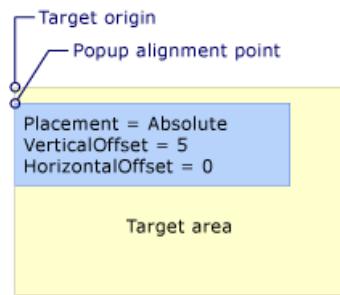
需要将 `PlacementTarget`、`PlacementRectangle` 和 `Placement` 的值一起考虑，才可确定正确的目标区域、目标原点和 Popup 对齐点。例如，如果 `Placement` 的值是 `Mouse`，则目标对象不存在、`PlacementRectangle` 将被忽略并且目标区域是鼠标指针的边界。另一方面，如果 `Placement` 是 `Bottom`，则 `PlacementTarget` 或父级决定目标对象，而 `PlacementRectangle` 决定目标区域。

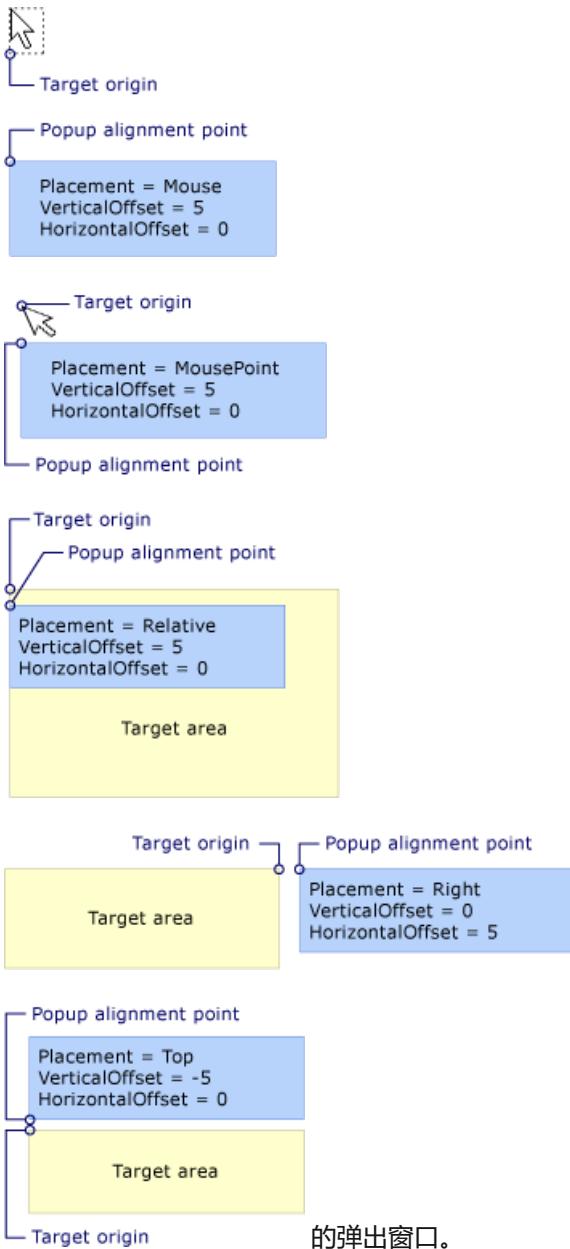
下表介绍了目标对象、目标区域、目标原点和 Popup 对齐点，并指示是否对每个 `PlacementMode` 枚举值使用 `PlacementTarget` 和 `PlacementRectangle`。

PlacementMode	“目标对象”	目标区域	目标原点	Popup 对齐点
Absolute	不适用。 <code>PlacementTarget</code> 将被忽略。	屏幕或 <code>PlacementRectangle</code> (如果已设置)。 <code>PlacementRectangle</code> 相对于屏幕。	目标区域的左上角。	Popup 的左上角。
AbsolutePoint	不适用。 <code>PlacementTarget</code> 将被忽略。	屏幕或 <code>PlacementRectangle</code> (如果已设置)。 <code>PlacementRectangle</code> 相对于屏幕。	目标区域的左上角。	Popup 的左上角。
Bottom	<code>PlacementTarget</code> 或父级。	目标对象或 <code>PlacementRectangle</code> (如果已设置)。 <code>PlacementRectangle</code> 相对于目标对象。	目标区域的左下角。	Popup 的左上角。
Center	<code>PlacementTarget</code> 或父级。	目标对象或 <code>PlacementRectangle</code> (如果已设置)。 <code>PlacementRectangle</code> 相对于目标对象。	目标区域的中心。	Popup 的中心。
Custom	<code>PlacementTarget</code> 或父级。	目标对象或 <code>PlacementRectangle</code> (如果已设置)。 <code>PlacementRectangle</code> 相对于目标对象。	由 <code>CustomPopupPlacementCallback</code> 定义。	由 <code>CustomPopupPlacementCallback</code> 定义。
Left	<code>PlacementTarget</code> 或父级。	目标对象或 <code>PlacementRectangle</code> (如果已设置)。 <code>PlacementRectangle</code> 相对于目标对象。	目标区域的左上角。	Popup 的右上角。
Mouse	不适用。 <code>PlacementTarget</code> 将被忽略。	鼠标指针的边界。 <code>PlacementRectangle</code> 将被忽略。	目标区域的左下角。	Popup 的左上角。
MousePoint	不适用。 <code>PlacementTarget</code> 将被忽略。	鼠标指针的边界。 <code>PlacementRectangle</code> 将被忽略。	目标区域的左上角。	Popup 的左上角。
Relative	<code>PlacementTarget</code> 或父级。	目标对象或 <code>PlacementRectangle</code> (如果已设置)。 <code>PlacementRectangle</code> 相对于目标对象。	目标区域的左上角。	Popup 的左上角。

PlacementMode	“目标对象”	目标区域	目标原点	Popup 对齐点
RelativePoint	PlacementTarget 或父级。	目标对象或 PlacementRectangle (如果已设置)。 PlacementRectangle 相对于目标对象。	目标区域的左上角。	Popup 的左上角。
Right	PlacementTarget 或父级。	目标对象或 PlacementRectangle (如果已设置)。 PlacementRectangle 相对于目标对象。	目标区域的右上角。	Popup 的左上角。
Top	PlacementTarget 或父级。	目标对象或 PlacementRectangle (如果已设置)。 PlacementRectangle 相对于目标对象。	目标区域的左上角。	Popup 的左下角。

下图显示了每个 PlacementMode 值的 Popup、目标区域、目标原点和 Popup 对齐点。在每个图中，目标区域为黄色，而 Popup 为蓝色。





的弹出窗口。

当 Popup 到达屏幕边缘时

出于安全原因，屏幕边缘不能隐藏 Popup。当 Popup 到达屏幕边缘时，将出现以下三种情况之一：

- Popup 沿着将遮挡 Popup 的屏幕边缘重新自行对齐。
- Popup 使用其他 Popup 对齐点。
- Popup 使用其他目标原点和 Popup 对齐点。

将在本部分后面进一步介绍这些选项。

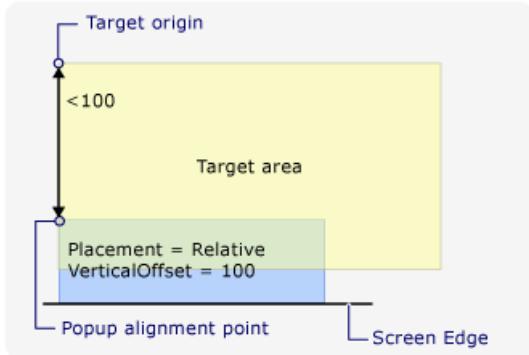
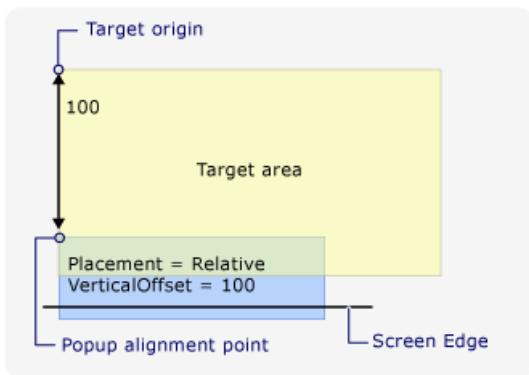
Popup 到达屏幕边缘时的行为取决于 [Placement](#) 属性值以及它所到达的具体屏幕边缘。下表汇总了针对每个 [PlacementMode](#) 值，Popup 到达屏幕边缘时的行为。

PlacementMode	上边缘	下边缘	左边缘	右边缘
Absolute	与上边缘对齐。	与下边缘对齐。	与左边缘对齐。	与右边缘对齐。
AbsolutePoint	与上边缘对齐。	Popup 对齐点更改为 Popup 的左下角。	与左边缘对齐。	Popup 对齐点更改为 Popup 的右上角。

PlacementMode	上边缘	下边缘	左边缘	右边缘
Bottom	与上边缘对齐。	目标原点更改为目标区域的左上角，而 Popup 对齐点更改为 Popup 的左下角。	与左边缘对齐。	与右边缘对齐。
Center	与上边缘对齐。	与下边缘对齐。	与左边缘对齐。	与右边缘对齐。
Left	与上边缘对齐。	与下边缘对齐。	目标原点更改为目标区域的右上角，而 Popup 对齐点更改为 Popup 的左上角。	与右边缘对齐。
Mouse	与上边缘对齐。	目标原点更改为目标区域（鼠标指针的边界）的左上角，而 Popup 对齐点更改为 Popup 的左下角。	与左边缘对齐。	与右边缘对齐。
MousePoint	与上边缘对齐。	Popup 对齐点更改为 Popup 的左下角。	与左边缘对齐。	Popup 对齐点更改为 Popup 的右上角。
Relative	与上边缘对齐。	与下边缘对齐。	与左边缘对齐。	与右边缘对齐。
RelativePoint	与上边缘对齐。	Popup 对齐点更改为 Popup 的左下角。	与左边缘对齐。	Popup 对齐点更改为 Popup 的右上角。
Right	与上边缘对齐。	与下边缘对齐。	与左边缘对齐。	目标原点更改为目标区域的左上角，而 Popup 对齐点更改为 Popup 的右上角。
Top	目标原点更改为目标区域的左下角，而 Popup 对齐点更改为 Popup 的左上角。实际上，这与 Bottom 是 Placement 时相同。	与下边缘对齐。	与左边缘对齐。	与右边缘对齐。

对屏幕边缘对齐

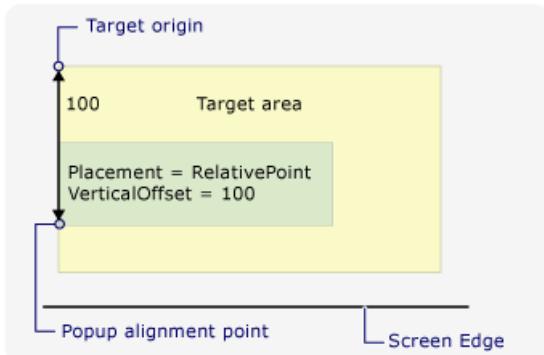
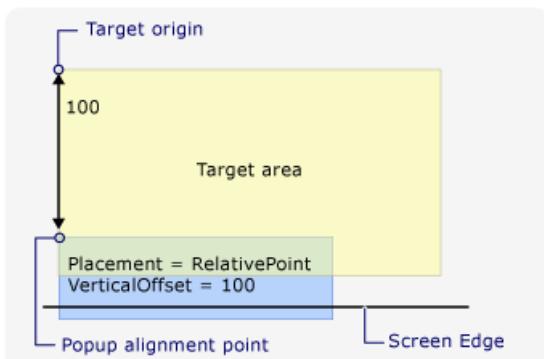
Popup 可通过重新自行定位以与屏幕边缘对齐，使整个 Popup 在屏幕上可见。出现这种情况时，目标原点和 Popup 对齐点之间的距离可能与 HorizontalOffset 和 VerticalOffset 的值不同。当 Placement 是 Absolute、Center 或 Relative 时，Popup 将自身与每个屏幕边缘对齐。例如，假定 Popup 将 Placement 设置为 Relative 并将 VerticalOffset 设置为 100。如果屏幕下边缘隐藏全部或部分 Popup，则 Popup 沿屏幕下边缘重新自行定位，并且目标原点和 Popup 对齐点之间的垂直距离小于 100。下图演示了此情况。



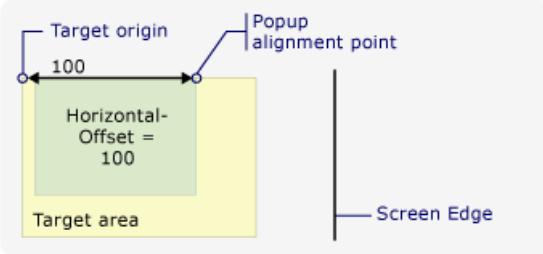
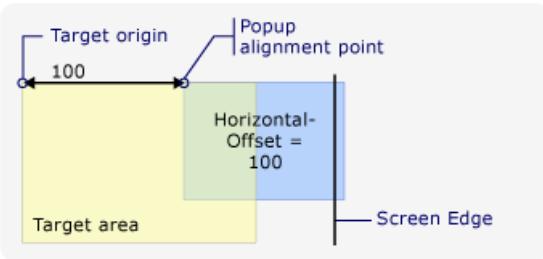
更改 Popup 对齐点

如果 `Placement` 是 `AbsolutePoint`、`RelativePoint` 或 `MousePoint`，则当 Popup 到达屏幕下边缘或右边缘时，Popup 对齐点将发生更改。

下图演示当屏幕下边缘隐藏全部或部分 Popup 时，Popup 对齐点为 `Popup` 的左下角。



下图演示了当屏幕下边缘隐藏 Popup 时，Popup 对齐点为 `Popup` 的右上角。

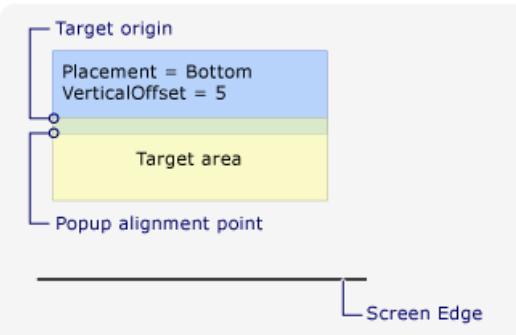
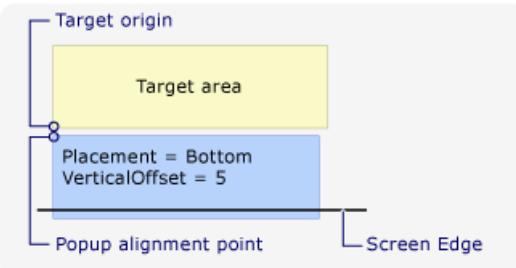


如果 `Popup` 到达屏幕的下边缘和右边缘，则 `Popup` 对齐点为 `Popup` 的右下角。

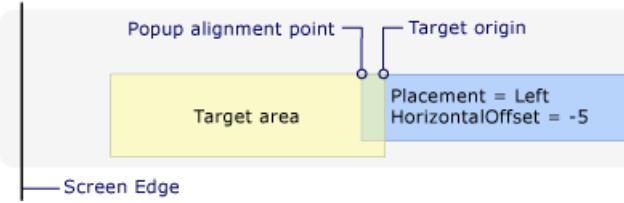
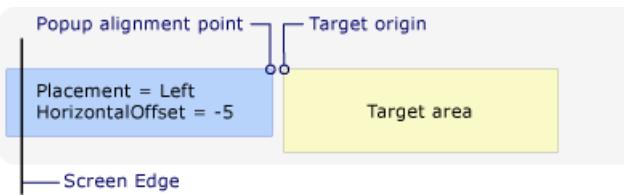
更改目标原点和 Popup 对齐点

当 `Placement` 是 `Bottom`、`Left`、`Mouse`、`Right` 或 `Top` 时，如果 `Popup` 到达某个屏幕边缘，则目标原点和 `Popup` 对齐点将发生更改。导致位置变化的屏幕边缘取决于 `PlacementMode` 值。

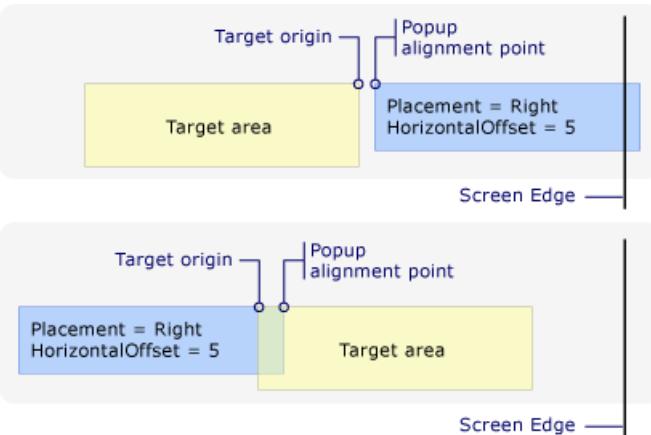
下图演示了当 `Placement` 是 `Bottom` 并且 `Popup` 到达屏幕下边缘时，目标原点是目标区域的左上角，而 `Popup` 对齐点是 `Popup` 的左下角。



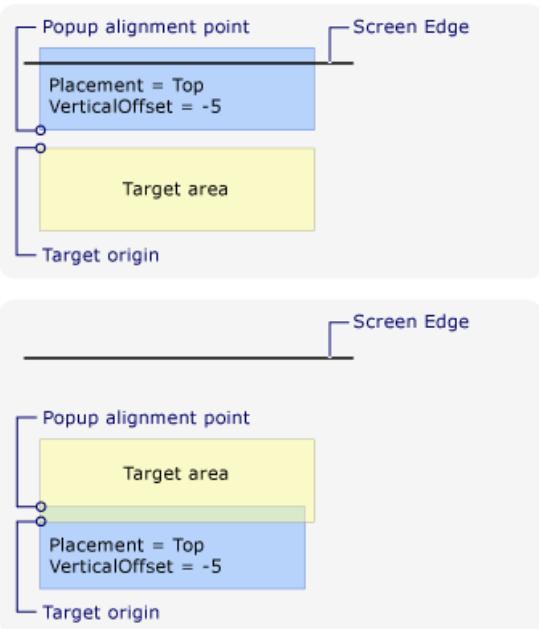
下图演示了当 `Placement` 是 `Left` 并且 `Popup` 到达屏幕左边缘时，目标原点是目标区域的右上角，而 `Popup` 对齐点是 `Popup` 的左上角。



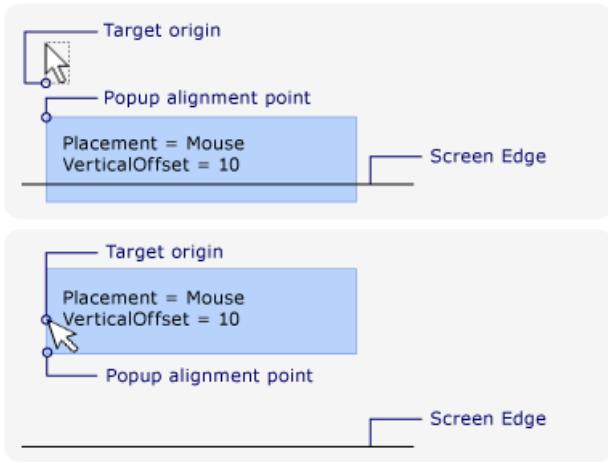
下图演示了当 **Placement** 是 **Right** 并且 **Popup** 到达屏幕右边缘时，目标原点是目标区域的左上角，而 **Popup** 对齐点是 **Popup** 的右上角。



下图演示了当 **Placement** 是 **Top** 并且 **Popup** 到达屏幕上边缘时，目标原点是目标区域的左下角，而 **Popup** 对齐点是 **Popup** 的左上角。



下图演示了当 **Placement** 是 **Mouse** 并且 **Popup** 到达屏幕下边缘时，目标原点是目标区域（鼠标指针的边界）的左上角，而 **Popup** 对齐点是 **Popup** 的左下角。



自定义 Popup 放置

可通过将 `Placement` 属性设置为 `Custom` 来自定义目标原点和 Popup 对齐点。然后定义一个 `CustomPopupPlacementCallback` 委托，该委托返回 `Popup` 的一组可能的放置点和主轴（按优先顺序排列）。显示 `Popup` 最大部分的点被选中。如果屏幕边缘隐藏了 `Popup`，则 `Popup` 的位置会自动调整。有关示例，请参阅 [指定自定义 Popup 位置](#)。

另请参阅

- [Popup 放置示例](#)

Popup 帮助主题

项目 • 2023/02/06

本部分中的主题介绍如何使用 [Popup](#) 控件在浮动在当前应用程序窗口上方的单独窗口中显示内容。

本节内容

[对 Popup 进行动画处理](#)

[指定自定义 Popup 位置](#)

参考

[Popup](#)

相关章节

[Popup 概述](#)

如何：对 Popup 进行动画处理

项目 • 2023/02/06

此示例演示了对 [Popup](#) 控件进行动画处理的两种方法。

示例

以下示例将 [PopupAnimation](#) 属性设置为 [Slide](#) 的一个值，当它出现时会造成 [Popup](#) “滑入”。

为了旋转 [Popup](#)，此示例分配了 [RotateTransform](#) 给 [Canvas](#) 上的 [RenderTransform](#) 属性，这是 [Popup](#) 的子元素。

为了使转换正常工作，本示例必须将 [AllowsTransparency](#) 属性设置为 `true`。此外，[Canvas](#) 上的 [Margin](#) 内容必须指定足够的空间才能使 [Popup](#) 旋转。

XAML

```
<Popup IsOpen="{Binding ElementName=myCheckBox, Path=IsChecked}"  
       PlacementTarget="{Binding ElementName=myCheckBox}"  
       AllowsTransparency="True"  
       PopupAnimation="Slide"  
       HorizontalOffset="50"  
       VerticalOffset="50"  
       >  
    <!--The Margin set on the Canvas provides the additional  
        area around the Popup so that the Popup is visible when  
        it rotates.-->  
    <Canvas Width="100" Height="100" Background="DarkBlue"  
           Margin="150">  
        <Canvas.RenderTransform>  
            <RotateTransform x:Name="theTransform" />  
        </Canvas.RenderTransform>  
        <TextBlock TextWrapping="Wrap" Foreground="White">  
            Rotating Popup  
        </TextBlock>  
    </Canvas>  
</Popup>
```

以下示例演示了 [Click](#) 事件（当单击 [Button](#) 时发生）如何触发 [Storyboard](#) 启动动画的事件。

XAML

```
<Button HorizontalAlignment="Left" Width="200" Margin="20,10,0,0">  
    <Button.Triggers>
```

```
<EventTrigger RoutedEvent="Button.Click">
  <BeginStoryboard>
    <Storyboard>
      <DoubleAnimation
        Storyboard.TargetName="theTransform"
        Storyboard.TargetProperty="(RotateTransform.Angle)"
        From="0" To="360" Duration="0:0:5" AutoReverse="True"/>
    </Storyboard>
  </BeginStoryboard>
</EventTrigger>
</Button.Triggers>
Click to see the Popup animate
</Button>
```

另请参阅

- [RenderTransform](#)
- [BulletDecorator](#)
- [RotateTransform](#)
- [Storyboard](#)
- [Popup](#)
- [操作指南主题](#)
- [Popup 概述](#)

如何：指定自定义 Popup 位置

项目 • 2023/02/06

此示例演示如何在 Placement 属性设置为 Custom 时为 Popup 控件指定自定义位置。

示例

当 Placement 属性设置为 Custom 时，Popup 调用 CustomPopupPlacementCallback 委托的一个已定义实例。此委托返回一组相对于目标区域的左上角和 Popup 的左上角的可能点。在提供最佳可见性的位置放置 Popup。

下面的示例演示如何通过将 Placement 属性设置为 Custom 来定义 Popup 的位置。它还演示如何创建和分配 CustomPopupPlacementCallback 委托，以便定位 Popup。回调委托返回两个 CustomPopupPlacement 对象。如果 Popup 在第一个位置被屏幕边缘隐藏，则 Popup 被放置在第二个位置。

XAML

```
<Popup Name="popup1"
       PlacementTarget ="{Binding ElementName=myButton}"
       Placement="Custom">
    <TextBlock Height="60" Width="200"
               Background="LightGray"
               TextWrapping="Wrap">Popup positioned by using
               CustomPopupPlacement callback delegate</TextBlock>
</Popup>
```

C#

```
public CustomPopupPlacement[] placePopup(Size popupSize,
                                         Size targetSize,
                                         Point offset)
{
    CustomPopupPlacement placement1 =
        new CustomPopupPlacement(new Point(-50, 100),
        PopupPrimaryAxis.Vertical);

    CustomPopupPlacement placement2 =
        new CustomPopupPlacement(new Point(10, 20),
        PopupPrimaryAxis.Horizontal);

    CustomPopupPlacement[] ttplaces =
        new CustomPopupPlacement[] { placement1, placement2 };
    return ttplaces;
}
```

C#

```
popup1.CustomPopupPlacementCallback =
    new CustomPopupPlacementCallback(placePopup);
```

有关完整的示例，请参阅[弹出项放置示例](#)。

另请参阅

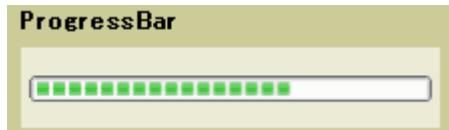
- [Popup](#)
- [弹出项概述](#)
- [操作指南文章](#)

ProgressBar

项目 • 2023/02/06

ProgressBar 指示操作的进度。 ProgressBar 控件包含一个窗口，其中填充系统高亮色来显示操作进度。

下图展示了典型的 ProgressBar。



本节内容

参考

[ProgressBar](#)

[StatusBar](#)

相关章节

PrintDialog

项目 • 2023/02/06

PrintDialog 控件用于实例化根据用户输入自动配置 PrintTicket 和 PrintQueue 的标准打印对话框。

参考

[PrintDialog](#)

[PrintTicket](#)

[PrintQueue](#)

请参阅

- [打印概述](#)
- [WPF 中的文档](#)

RadioButton

项目 • 2023/02/06

[RadioButton](#) 控件通常组合在一起，让用户可以在多个选项中进行选择；一次只能选择一个按钮。

下图显示了 [RadioButton](#) 控件的示例。



典型 RadioButton

参考

[ToggleButton](#)

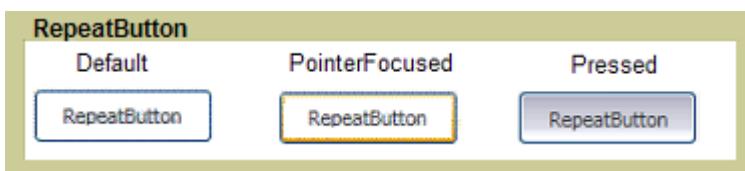
相关章节

RepeatButton

项目 • 2023/02/06

RepeatButton 类似于 Button。但是，RepeatButton 元素使你能够控制 Click 事件发生的时间和方式。

下图显示了重复按钮控件的三种状态：Default、PointerFocused 和 Pressed。第一个按钮显示 RepeatButton 的默认状态。第二个显示了当鼠标指针悬停在按钮上并使其聚焦时，按钮的外观如何变化。最后一个按钮显示用户在控件上按下鼠标按钮时 RepeatButton 的外观。



典型 RepeatButton

本节内容

参考

[RepeatButton](#)

相关章节

RichTextBox

项目 • 2023/02/06

[RichTextBox](#) 元素定义一个编辑控件，其中内置了剪切粘贴、丰富的文档演示和内容选择等功能支持。

本节内容

[RichTextBox 概述](#)

[操作指南主题](#)

另请参阅

- [TextBox](#)
- [WPF 中的文档](#)
- [流文档概述](#)

RichTextBox 概述

项目 • 2022/09/27

使用 [RichTextBox](#) 控件可以显示或编辑流内容，包括段落、图像、表格等。本主题介绍 [TextBox](#) 类，并提供有关如何在 Extensible Application Markup Language (XAML) 和 C# 中使用它的示例。

使用 TextBox 还是 RichTextBox？

[RichTextBox](#) 和 [TextBox](#) 都允许用户编辑文本，但两个控件用于不同场景。当用户需要编辑带格式的文本、图像、表格或其他多种格式的内容时，[RichTextBox](#) 是更好的选择。例如，编辑需要格式、图像等的文档、文章或博客时，最好使用 [RichTextBox](#) 来实现。

[TextBox](#) 需要的系统资源比 [RichTextBox](#) 少，因此非常适合只需要编辑纯文本的场景（例如在窗体中使用）。有关详细信息，请参阅 [TextBox](#) 中的 [TextBox 概述](#)。下表汇总了 [TextBox](#) 和 [RichTextBox](#) 的主要功能。

控制	实时拼写检查	上下文菜单	格式命令，例如 ToggleBold (Ctr+B)	FlowDocument 内容，例如图像、段落、表格等
TextBox	是	是	否	否。
RichTextBox	是	是	是	是

① 备注

虽然 [TextBox](#) 不支持与格式相关的命令（例如 [ToggleBold \(Ctr+B\)](#)），但是两个控件均支持许多基本命令，例如 [MoveToLineEnd](#)。

后面将更详细地介绍上表中的功能。

创建 RichTextBox

下面的代码演示如何创建 [RichTextBox](#)，以供用户在其中编辑多种格式的内容。

XAML

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

    <!-- A RichTextBox with no initial content in it. -->
    <RichTextBox />
```

```
</Page>
```

具体而言，在 [RichTextBox](#) 中编辑的内容是流内容。流内容可包含许多类型的元素，包括带格式的文本、图像、列表和表格。有关流文档的详细信息，请参阅[流文档概述](#)。为了包含流内容，[RichTextBox](#) 托管 [FlowDocument](#) 对象，后者也包含可编辑的内容。为了在 [RichTextBox](#) 中演示流内容，以下代码演示如何创建带有段落和某些加粗文本的 [RichTextBox](#)。

XAML

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

    <StackPanel>
        <RichTextBox>
            <FlowDocument>
                <Paragraph>
                    This is flow content and you can <Bold>edit me!</Bold>
                </Paragraph>
            </FlowDocument>
        </RichTextBox>
    </StackPanel>

</Page>
```

C#

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Documents;
namespace SDKSample
{
    public partial class BasicRichTextBoxWithContentExample : Page
    {
        public BasicRichTextBoxWithContentExample()
        {
            StackPanel myStackPanel = new StackPanel();

            // Create a FlowDocument to contain content for the RichTextBox.
            FlowDocument myFlowDoc = new FlowDocument();

            // Create a Run of plain text and some bold text.
            Run myRun = new Run("This is flow content and you can ");
            Bold myBold = new Bold(new Run("edit me!"));

            // Create a paragraph and add the Run and Bold to it.
            Paragraph myParagraph = new Paragraph();
            myParagraph.Inlines.Add(myRun);
```

```
myParagraph.Inlines.Add(myBold);

// Add the paragraph to the FlowDocument.
myFlowDoc.Blocks.Add(myParagraph);

RichTextBox myRichTextBox = new RichTextBox();

// Add initial content to the RichTextBox.
myRichTextBox.Document = myFlowDoc;

myStackPanel.Children.Add(myRichTextBox);
this.Content = myStackPanel;
}

}

}
```

下图显示了此示例的呈现效果。

This is flow content and you can **edit me!**

Paragraph 和 Bold 等元素可决定 RichTextBox 内部内容的显示方式。当用户编辑 RichTextBox 内容时，此流内容会发生更改。若要了解有关流内容的功能及其工作方式的详细信息，请参阅[流文档概述](#)。

① 备注

RichTextBox 内部的流内容行为与其他控件中包含的流内容行为并不完全相同。例如，RichTextBox 没有分列，因此没有自动重设大小行为。另外，在 RichTextBox 中不能使用内置功能，例如搜索、查看模式、页面导航和缩放。

实时拼写检查

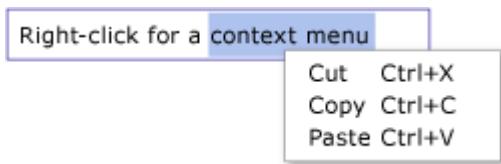
可以在 TextBox 或 RichTextBox 中启用实时拼写检查。启用拼写检查时，任何拼写错误的字词下方都会出现红线（见下图）。

The following word is Missspelled. |

若要了解如何启用拼写检查，请参阅[在文本编辑控件中启用拼写检查](#)。

上下文菜单

默认情况下，TextBox 和 RichTextBox 都有一个上下文菜单，该菜单在用户在控件内右键单击时显示。上下文菜单使用户可以剪切、复制或粘贴（见下图）。



可以创建自己的自定义上下文菜单来重写默认的上下文菜单。有关详细信息，请参阅[在 RichTextBox 中定位自定义上下文菜单](#)。

编辑命令

用户可使用编辑命令为 [RichTextBox](#) 内的可编辑内容设置格式。除了基本编辑命令，[RichTextBox](#) 还包括 [TextBox](#) 不支持的格式设置命令。例如，在编辑 [RichTextBox](#) 时，用户可以按 [Ctrl+B](#) 来切换加粗文本格式。有关完整的可用命令列表，请参阅[EditingCommands](#)。除了使用键盘快捷方式，还可以将命令与按钮之类的其他控件挂钩。以下示例演示如何创建如何创建简单的工具栏，其中包含用户可用来更改文本格式的按钮。

XAML

```
<Window x:Class="RichTextBoxInputPanelDemo.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Height="400"
    Width="600"
    >
    <Grid>

        <!-- Set the styles for the tool bar. -->
        <Grid.Resources>
            <Style TargetType="{x:Type Button}" x:Key="formatTextStyle">
                <Setter Property="FontFamily" Value="Palatino Linotype"></Setter>
                <Setter Property="Width" Value="30"></Setter>
                <Setter Property="FontSize" Value ="14"></Setter>
                <Setter Property="CommandTarget" Value="{Binding
ElementName=mainRTB}"></Setter>
            </Style>

            <Style TargetType="{x:Type Button}" x:Key="formatImageStyle">
                <Setter Property="Width" Value="30"></Setter>
                <Setter Property="CommandTarget" Value="{Binding
ElementName=mainRTB}"></Setter>
            </Style>
        </Grid.Resources>

        <DockPanel Name="mainPanel">

            <!-- This tool bar contains all the editing buttons. -->
            <ToolBar Name="mainToolBar" Height="30" DockPanel.Dock="Top">

                <Button Style="{StaticResource formatImageStyle}">
```

```

        Command="ApplicationCommands.Cut" ToolTip="Cut">
            <Image Source="Images>EditCut.png"></Image>
        
```

```

        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="ApplicationCommands.Copy" ToolTip="Copy">
            <Image Source="Images>EditCopy.png"></Image>
        
```

```

        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="ApplicationCommands.Paste" ToolTip="Paste">
            <Image Source="Images>EditPaste.png"></Image>
        
```

```

        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="ApplicationCommands.Undo" ToolTip="Undo">
            <Image Source="Images>EditUndo.png"></Image>
        
```

```

        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="ApplicationCommands.Redo" ToolTip="Redo">
            <Image Source="Images>EditRedo.png"></Image>
        
```

```

        </Button>

        <Button Style="{StaticResource formatTextStyle}" Command="EditingCommands.ToggleBold" ToolTip="Bold">
            <TextBlock FontWeight="Bold">B</TextBlock>
        
```

```

        </Button>
        <Button Style="{StaticResource formatTextStyle}" Command="EditingCommands.ToggleItalic" ToolTip="Italic">
            <TextBlock FontStyle="Italic" FontWeight="Bold">I</TextBlock>
        
```

```

        </Button>
        <Button Style="{StaticResource formatTextStyle}" Command="EditingCommands.ToggleUnderline" ToolTip="Underline">
            <TextBlock TextDecorations="Underline" FontWeight="Bold">U</TextBlock>
        
```

```

        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.IncreaseFontSize" ToolTip="Grow Font">
            <Image Source="Images\CharacterGrowFont.png"></Image>
        
```

```

        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.DecreaseFontSize" ToolTip="Shrink Font">
            <Image Source="Images\CharacterShrinkFont.png"></Image>
        
```

```

        </Button>

        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.ToggleBullets" ToolTip="Bullets">
            <Image Source="Images>ListBullets.png"></Image>
        
```

```

        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.ToggleNumbering" ToolTip="Numbering">
            <Image Source="Images>ListNumbering.png"></Image>
        
```

```

        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.AlignLeft" ToolTip="Align Left">
            <Image Source="Images\ParagraphLeftJustify.png"></Image>
        
```

```

        </Button>
        <Button Style="{StaticResource formatImageStyle}">
    
```

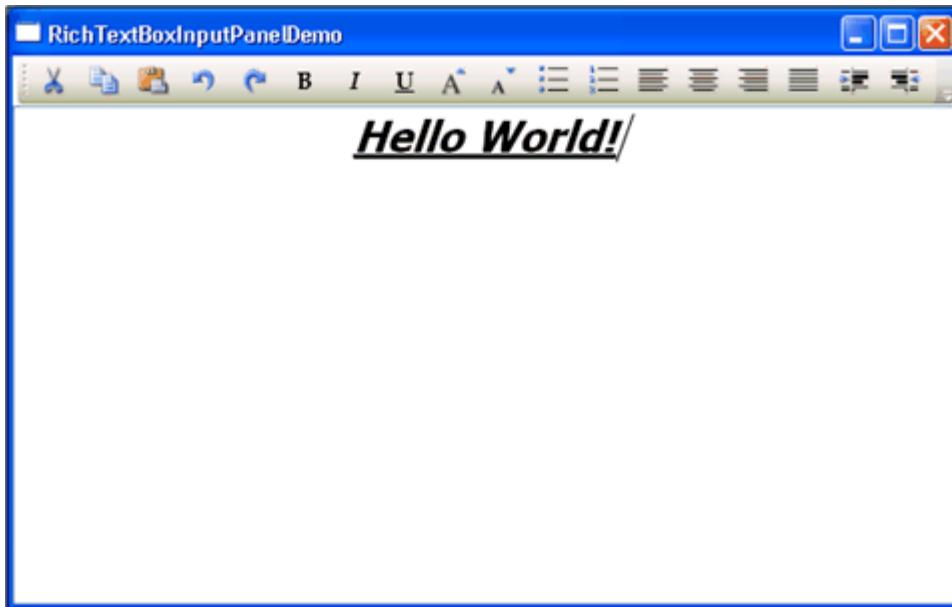
```

        Command="EditingCommands.AlignCenter" ToolTip="Align Center">
            <Image Source="Images\ParagraphCenterJustify.png"></Image>
        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.AlignRight" ToolTip="Align Right">
            <Image Source="Images\ParagraphRightJustify.png"></Image>
        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.AlignJustify" ToolTip="Align Justify">
            <Image Source="Images\ParagraphFullJustify.png"></Image>
        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.IncreaseIndentation" ToolTip="Increase Indent">
            <Image Source="Images\ParagraphIncreaseIndentation.png"></Image>
        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.DecreaseIndentation" ToolTip="Decrease Indent">
            <Image Source="Images\ParagraphDecreaseIndentation.png"></Image>
        </Button>
    </ToolBar>

    <!-- By default pressing tab moves focus to the next control. Setting AcceptsTab to true allows the RichTextBox to accept tab characters. -->
    <RichTextBox Name="mainRTB" AcceptsTab="True"></RichTextBox>
</DockPanel>
</Grid>
</Window>

```

下图显示此示例的显示效果。



检测内容何时更改

通常 `TextChanged` 事件应该用于检测 `TextBox` 或 `RichTextBox` 中文本的更改，而不是你可能认为的 `KeyDown`。有关示例，请参阅[检测 TextBox 中的文本何时更改](#)。

保存、加载和打印 RichTextBox 内容

以下示例演示如何将 `RichTextBox` 的内容保存到文件，将该内容加载回 `RichTextBox`，并打印内容。下面是示例的标记。

XAML

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      x:Class="SDKSample.SaveLoadPrintRTB" >

    <StackPanel>
        <RichTextBox Name="richTB">
            <FlowDocument>
                <Paragraph>
                    <Run>Paragraph 1</Run>
                </Paragraph>
            </FlowDocument>
        </RichTextBox>

        <Button Click="SaveRTBContent">Save RTB Content</Button>
        <Button Click="LoadRTBContent">Load RTB Content</Button>
        <Button Click="PrintRTBContent">Print RTB Content</Button>
    </StackPanel>

</Page>
```

下面是该示例的隐藏代码。

C#

```
using System;
using System.IO;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Media;

namespace SDKSample
{
    public partial class SaveLoadPrintRTB : Page
    {
        // Handle "Save RichTextBox Content" button click.
        void SaveRTBContent(Object sender, RoutedEventArgs args)
        {
```

```

        // Send an arbitrary URL and file name string specifying
        // the location to save the XAML in.
        SaveXamlPackage("C:\\test.xaml");
    }

    // Handle "Load RichTextBox Content" button click.
    void LoadRTBContent(Object sender, RoutedEventArgs args)
    {
        // Send URL string specifying what file to retrieve XAML
        // from to load into the RichTextBox.
        LoadXamlPackage("C:\\test.xaml");
    }

    // Handle "Print RichTextBox Content" button click.
    void PrintRTBContent(Object sender, RoutedEventArgs args)
    {
        PrintCommand();
    }

    // Save XAML in RichTextBox to a file specified by _fileName
    void SaveXamlPackage(string _fileName)
    {
        TextRange range;
        FileStream fStream;
        range = new TextRange(richTB.Document.ContentStart,
richTB.Document.ContentEnd);
        fStream = new FileStream(_fileName, FileMode.Create);
        range.Save(fStream, DataFormats.XamlPackage);
        fStream.Close();
    }

    // Load XAML into RichTextBox from a file specified by _fileName
    void LoadXamlPackage(string _fileName)
    {
        TextRange range;
        FileStream fStream;
        if (File.Exists(_fileName))
        {
            range = new TextRange(richTB.Document.ContentStart,
richTB.Document.ContentEnd);
            fStream = new FileStream(_fileName, FileMode.OpenOrCreate);
            range.Load(fStream, DataFormats.XamlPackage);
            fStream.Close();
        }
    }

    // Print RichTextBox content
    private void PrintCommand()
    {
        PrintDialog pd = new PrintDialog();
        if ((pd.ShowDialog() == true))
        {
            //use either one of the below
            pd.PrintVisual(richTB as Visual, "printing as visual");
        }
    }
}

```

```
pd.PrintDocument(((IDocumentPaginatorSource)richTB.Document).DocumentPaginator), "printing as paginator");
        }
    }
}
```

另请参阅

- [操作指南主题](#)
- [TextBox 概述](#)

RichTextBox 帮助主题

项目 • 2023/02/06

本部分提供的示例演示如何使用 [RichTextBox](#) 控件完成常见任务。

本节内容

[从 RichTextBox 提取文本内容](#)

[以编程方式更改 RichTextBox 中的选定内容](#)

[保存、加载和打印 RichTextBox 内容](#)

[在 RichTextBox 中确定自定义上下文菜单的位置](#)

另请参阅

- [TextBox](#)
- [WPF 中的文档](#)
- [流文档概述](#)

如何：从 RichTextBox 中提取文本内容

项目 • 2023/02/06

此示例演示如何将 [RichTextBox](#) 的内容提取为纯文本。

描述 RichTextBox 控件

以下 Extensible Application Markup Language (XAML) 代码描述了具有简单内容的命名 [RichTextBox](#) 控件。

XAML

```
<RichTextBox Name="richTB">
  <FlowDocument>
    <Paragraph>
      <Run>Paragraph 1</Run>
    </Paragraph>
    <Paragraph>
      <Run>Paragraph 2</Run>
    </Paragraph>
    <Paragraph>
      <Run>Paragraph 3</Run>
    </Paragraph>
  </FlowDocument>
</RichTextBox>
```

将 RichTextBox 用作参数的代码示例

以下代码实现了一个方法，将 [RichTextBox](#) 用作参数，返回一个表示 [RichTextBox](#) 的纯文本内容的字符串。

该方法从 [RichTextBox](#) 的内容创建了一个新的 [TextRange](#)，使用 [ContentStart](#) 和 [ContentEnd](#) 来指示要提取的内容的范围。[ContentStart](#) 和 [ContentEnd](#) 属性各自返回了一个 [TextPointer](#)，并且可以在表示 [RichTextBox](#) 内容的基础 [FlowDocument](#) 上访问。[TextRange](#) 提供了一个 [Text](#) 属性，以字符串形式返回 [TextRange](#) 的纯文本部分。

C#

```
string StringFromRichTextBox(RichTextBox rtb)
{
    TextRange textRange = new TextRange(
        // TextPointer to the start of content in the RichTextBox.
        rtb.Document.ContentStart,
        // TextPointer to the end of content in the RichTextBox.
        rtb.Document.ContentEnd
```

```
);

// The Text property on a TextRange object returns a string
// representing the plain text content of the TextRange.
return textRange.Text;
}
```

另请参阅

- [RichTextBox 概述](#)
- [TextBox 概述](#)

以编程方式更改 RichTextBox 中的选定内容

项目 • 2023/02/06

此示例演示如何以编程方式更改 [RichTextBox](#) 中的选定内容。此选定内容与用户使用用户界面选择的内容相同。

RichTextBox 控件的代码示例

以下可扩展应用程序标记语言 (XAML) 代码描述具有简单内容的 [RichTextBox](#) 控件。

XAML

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      x:Class="SDKSample.ChangeSelectionProgrammaticaly" >

    <StackPanel>
        <RichTextBox GotMouseCapture="ChangeSelection" Name="richTB">
            <FlowDocument>
                <Paragraph Name="myParagraph">
                    <Run>
                        When the user clicks in the RichTextBox, the selected
                        text changes programmatically.
                    </Run>
                </Paragraph>
            </FlowDocument>
        </RichTextBox>
    </StackPanel>

</Page>
```

从 RichTextBox 中选择文本的代码示例

用户在 [RichTextBox](#) 内单击时，下面的代码会以编程方式选择一些任意文本。

C#

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;

namespace SDKSample
{
```

```
public partial class ChangeSelectionProgrammaticaly : Page
{
    // Change the current selection.
    void ChangeSelection(Object sender, RoutedEventArgs args)
    {
        // Create two arbitrary TextPointers to specify the range of
        content to select.
        TextPointer myTextPointer1 =
myParagraph.ContentStart.GetPositionAtOffset(20);
        TextPointer myTextPointer2 =
myParagraph.ContentEnd.GetPositionAtOffset(-10);

        // Programmatically change the selection in the RichTextBox.
        richTB.Selection.Select(myTextPointer1, myTextPointer2);
    }
}
```

另请参阅

- [RichTextBox 概述](#)
- [TextBox 概述](#)

如何：保存、加载和打印 RichTextBox 内容

项目 • 2023/02/06

以下示例演示如何将 [RichTextBox](#) 的内容保存到文件，将该内容加载回 [RichTextBox](#)，并打印内容。

用于保存、加载和打印 RichTextBox 内容的标记

下面是示例的标记。

XAML

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      x:Class="SDKSample.SaveLoadPrintRTB" >

    <StackPanel>
        <RichTextBox Name="richTB">
            <FlowDocument>
                <Paragraph>
                    <Run>Paragraph 1</Run>
                </Paragraph>
            </FlowDocument>
        </RichTextBox>

        <Button Click="SaveRTBContent">Save RTB Content</Button>
        <Button Click="LoadRTBContent">Load RTB Content</Button>
        <Button Click="PrintRTBContent">Print RTB Content</Button>
    </StackPanel>

</Page>
```

用于保存、加载和打印 RichTextBox 内容的代码

下面是该示例的隐藏代码。

C#

```
using System;
using System.IO;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Media;
```

```
namespace SDKSample
{
    public partial class SaveLoadPrintRTB : Page
    {
        // Handle "Save RichTextBox Content" button click.
        void SaveRTBContent(Object sender, RoutedEventArgs args)
        {
            // Send an arbitrary URL and file name string specifying
            // the location to save the XAML in.
            SaveXamlPackage("C:\\test.xaml");
        }

        // Handle "Load RichTextBox Content" button click.
        void LoadRTBContent(Object sender, RoutedEventArgs args)
        {
            // Send URL string specifying what file to retrieve XAML
            // from to load into the RichTextBox.
            LoadXamlPackage("C:\\test.xaml");
        }

        // Handle "Print RichTextBox Content" button click.
        void PrintRTBContent(Object sender, RoutedEventArgs args)
        {
            PrintCommand();
        }

        // Save XAML in RichTextBox to a file specified by _fileName
        void SaveXamlPackage(string _fileName)
        {
            TextRange range;
            FileStream fStream;
            range = new TextRange(richTB.Document.ContentStart,
richTB.Document.ContentEnd);
            fStream = new FileStream(_fileName, FileMode.Create);
            range.Save(fStream, DataFormats.XamlPackage);
            fStream.Close();
        }

        // Load XAML into RichTextBox from a file specified by _fileName
        void LoadXamlPackage(string _fileName)
        {
            TextRange range;
            FileStream fStream;
            if (File.Exists(_fileName))
            {
                range = new TextRange(richTB.Document.ContentStart,
richTB.Document.ContentEnd);
                fStream = new FileStream(_fileName, FileMode.OpenOrCreate);
                range.Load(fStream, DataFormats.XamlPackage);
                fStream.Close();
            }
        }
}
```

```
}

// Print RichTextBox content
private void PrintCommand()
{
    PrintDialog pd = new PrintDialog();
    if ((pd.ShowDialog() == true))
    {
        //use either one of the below
        pd.PrintVisual(richTB as Visual, "printing as visual");

        pd.PrintDocument(((IDocumentPaginatorSource)richTB.Document).DocumentPaginator, "printing as paginator");
    }
}
```

另请参阅

- [RichTextBox 概述](#)
- [TextBox 概述](#)

如何：在 RichTextBox 中定位自定义上下文菜单

项目 • 2023/02/06

此示例演示如何为 [RichTextBox](#) 定位自定义上下文菜单。

当为 [RichTextBox](#) 定位自定义上下文菜单时，你负责处理上下文菜单的定位。默认情况下，自定义上下文菜单在 [RichTextBox](#) 的中心打开。

为 ContextMenuOpening 事件添加侦听器

若要重写默认定位行为，请为 [ContextMenuOpening](#) 事件添加侦听器。下面的示例演示如何通过编程方式执行此操作。

C#

```
richTextBox.ContextMenuOpening += new  
ContextMenuEventHandler(richTextBox_ContextMenuOpening);
```

ContextMenuOpening 事件侦听器的实现

以下示例演示相应 [ContextMenuOpening](#) 事件侦听器的实现。

C#

```
// This method is intended to listen for the ContextMenuOpening event from a  
RichTextBox.  
// It will position the custom context menu at the end of the current  
selection.  
void richTextBox_ContextMenuOpening(object sender, ContextMenuEventArgs e)  
{  
    // Sender must be RichTextBox.  
    RichTextBox rtb = sender as RichTextBox;  
    if (rtb == null) return;  
  
    ContextMenu contextMenu = rtb.ContextMenu;  
    contextMenu.PlacementTarget = rtb;  
  
    // This uses HorizontalOffset and VerticalOffset properties to position  
    // the menu,  
    // relative to the upper left corner of the parent element (RichTextBox  
    // in this case).  
    contextMenu.Placement = PlacementMode.RelativePoint;  
  
    // Compute horizontal and vertical offsets to place the menu relative to
```

```
selection end.  
    TextPointer position = rtb.Selection.End;  
  
    if (position == null) return;  
  
    Rect positionRect = position.GetCharacterRect(LogicalDirection.Forward);  
    contextMenu.HorizontalOffset = positionRect.X;  
    contextMenu.VerticalOffset = positionRect.Y;  
  
    // Finally, mark the event has handled.  
    contextMenu.IsOpen = true;  
    e.Handled = true;  
}
```

另请参阅

- [RichTextBox 概述](#)
- [TextBox 概述](#)

ScrollBar

项目 • 2023/02/06

利用 [ScrollBar](#)，可以通过滑动 [Thumb](#) 使内容可见来查看当前查看区域之外的内容。

本节内容

[自定义滚动条上的 Thumb 大小](#)

参考

[ScrollBar](#)

[Track](#)

[Thumb](#)

[ScrollViewer](#)

如何：自定义滚动条上的滚动块大小

项目 • 2023/02/06

本主题说明如何将 ScrollBar 的 Thumb 设置为固定大小，以及如何为 ScrollBar 的 Thumb 指定最小大小。

创建具有固定滚动块大小的滚动条

以下示例创建一个具有固定大小的 Thumb 的 ScrollBar。该示例将 Thumb 的 ViewportSize 属性设置为 NaN 并设置 Thumb 的高度。若要创建具有固定宽度的 Thumb 的水平 ScrollBar，请设置 Thumb 的宽度。

XAML

```
<Style TargetType="ScrollBar">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="ScrollBar">
                <Grid Name="Bg"
                    Background="{TemplateBinding Background}"
                    SnapsToDevicePixels="true">
                    <Grid.RowDefinitions>
                        <RowDefinition MaxHeight="{DynamicResource {x:Static SystemParameters.VerticalScrollBarButtonHeightKey}}"/>
                        <RowDefinition Height="0.0001*"/>
                        <RowDefinition MaxHeight="{DynamicResource {x:Static SystemParameters.VerticalScrollBarButtonHeightKey}}"/>
                    </Grid.RowDefinitions>
                    <RepeatButton Style="{StaticResource ScrollBarButton}"
                        IsEnabled="{TemplateBinding IsMouseOver}"
                        Height="18"
                        Command="ScrollBar.LineUpCommand"
                        Content="M 0 4 L 8 4 L 4 0 Z" />
                    <!-- Set the ViewporSize to NaN to disable autosizing of the
                    Thumb. -->
                    <Track Name="PART_Track"
                        ViewportSize="NaN"
                        IsDirectionReversed="true"
                        Grid.Row="1"
                        Grid.ZIndex="-1">
                        <Track.DecreaseRepeatButton>
                            <RepeatButton Style="{StaticResource VerticalScrollBarPageButton}"
                                Command="ScrollBar.PageUpCommand"/>
                        </Track.DecreaseRepeatButton>
                        <Track.IncreaseRepeatButton>
                            <RepeatButton Style="{StaticResource VerticalScrollBarPageButton}"
```

```

        Command="ScrollBar.PageDownCommand"/>
    </Track.IncreaseRepeatButton>
    <Track.Thumb>
        <!-- Set the height of the Thumb.-->
        <Thumb Height="30"/>
    </Track.Thumb>
</Track>
<RepeatButton
    Grid.Row="2"
    Style="{StaticResource ScrollBarButton}"
    Height="18"
    Command="ScrollBar.LineDownCommand"
    Content="M 0 0 L 4 4 L 8 0 Z"/>

</Grid>
<ControlTemplate.Triggers>
    <Trigger SourceName="PART_Track" Property="IsEnabled"
Value="false">
        <Setter TargetName="PART_Track" Property="Visibility"
Value="Hidden"/>
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
<Setter.Value>
</Setter>
</Style>

```

创建具有固定滚动块大小的滚动条

以下示例创建一个具有最小大小的 Thumb 的 ScrollBar。此示例设置 VerticalScrollBarHeightKey 的值。若要创建具有最小宽度的 Thumb 的水平 ScrollBar，请设置 HorizontalScrollBarWidthKey。

XAML

```

<Style TargetType="ScrollBar">
<Setter Property="Template">
<Setter.Value>
    <ControlTemplate TargetType="ScrollBar">
        <Grid Name="Bg"
            Background="{TemplateBinding Background}"
            SnapsToDevicePixels="true">
            <Grid.RowDefinitions>
                <RowDefinition MaxHeight="{DynamicResource
{x:Static SystemParameters.VerticalScrollBarHeightKey}}"/>
                <RowDefinition Height="0.0001*"/>
                <RowDefinition MaxHeight="{DynamicResource
{x:Static SystemParameters.VerticalScrollBarHeightKey}}"/>
            </Grid.RowDefinitions>
        <RepeatButton Style="{StaticResource ScrollBarButton}"
            IsEnabled="{TemplateBinding IsMouseOver}">

```

```

        Height="18"
        Command="ScrollBar.LineUpCommand"
        Content="M 0 4 L 8 4 L 4 0 Z" />
    <Track Name="PART_Track"
        IsDirectionReversed="true"
        Grid.Row="1"
        Grid.ZIndex="-1">
        <Track.Resources>
            <!-- Set the Thumb's minimum height to 50.
            The Thumb's minimum height is half the
            value of VerticalScrollBarHeightKey. -->
            <sys:Double
                x:Key="{x:Static
SystemParameters.VerticalScrollBarHeightKey}">
                100
            </sys:Double>
        </Track.Resources>
        <Track.DecreaseRepeatButton>
            <RepeatButton Style="{StaticResource
VerticalScrollBarPageButton}"
                Command="ScrollBar.PageUpCommand"/>
        </Track.DecreaseRepeatButton>
        <Track.IncreaseRepeatButton>
            <RepeatButton Style="{StaticResource
VerticalScrollBarPageButton}"
                Command="ScrollBar.PageDownCommand"/>
        </Track.IncreaseRepeatButton>
        <Track.Thumb>
            <Thumb/>
        </Track.Thumb>
    </Track>
    <RepeatButton
        Grid.Row="2"
        Style="{StaticResource ScrollBarButton}"
        Height="18"
        Command="ScrollBar.LineDownCommand"
        Content="M 0 0 L 4 4 L 8 0 Z"/>
</Grid>
<ControlTemplate.Triggers>
    <Trigger SourceName="PART_Track"
        Property="IsEnabled" Value="false">
        <Setter TargetName="PART_Track"
            Property="Visibility" Value="Hidden"/>
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

另请参阅

- ScrollBar 样式和模板

ScrollView

项目 • 2023/02/06

ScrollView 控件将创建可滚动的区域，可在其中水平或垂直滚动内容。

本节内容

[ScrollView 概述](#)

[操作指南主题](#)

参考

[ScrollBar](#)

[ScrollView](#)

请参阅

- [面板概述](#)
- [布局](#)

ScrollViewer 概述

项目 • 2022/09/27

用户界面中的内容通常比计算机屏幕的显示区域大。利用 [ScrollViewer](#) 控件可以方便地在 Windows Presentation Foundation (WPF) 应用程序中滚动内容。本主题介绍 [ScrollViewer](#) 元素，并提供若干用法示例。

ScrollViewer 控件

WPF 应用程序中有两个支持滚动的预定义元素：[ScrollBar](#) 和 [ScrollViewer](#)。[ScrollViewer](#) 控件封装了水平和垂直 [ScrollBar](#) 元素以及一个内容容器（如 [Panel](#) 元素），以便在可滚动的区域中显示其他可见元素。必须生成自定义对象才能使用 [ScrollBar](#) 元素实现内容滚动。不过，可以单独使用 [ScrollViewer](#) 元素，因为它是一个封装了 [ScrollBar](#) 功能的复合控件。

[ScrollViewer](#) 控件既响应鼠标命令，也响应键盘命令，并定义许多可用于按预设的增量滚动内容的方法。可以使用 [ScrollChanged](#) 事件来检测 [ScrollViewer](#) 状态的变化。

[ScrollViewer](#) 只能有一个子元素，通常是可承载 [Children](#) 元素集合的 [Panel](#) 元素。[Content](#) 属性定义 [ScrollViewer](#) 的唯一子元素。

物理与逻辑滚动

物理滚动用于按预设的物理增量（通常按以像素为单位声明的值）滚动内容。逻辑滚动用于滚动到逻辑树中的下一项。物理滚动是大多数 [Panel](#) 元素的默认滚动行为。WPF 同时支持这两种类型的滚动。

IScrollInfo 接口

[IScrollInfo](#) 接口表示 [ScrollViewer](#) 或派生控件内的主滚动区域。该接口定义可由 [Panel](#) 元素实现的滚动属性和方法，这些元素需要按逻辑单位（而不是按物理增量）滚动。通过将 [IScrollInfo](#) 的实例转换为派生的 [Panel](#)，然后使用其滚动方法，为滚动到子集合中的下一个逻辑单位（而不是按像素增量滚动）提供了有用的方式。默认情况下，[ScrollViewer](#) 控件支持按物理单位滚动。

[StackPanel](#) 和 [VirtualizingStackPanel](#) 都实现了 [IScrollInfo](#) 并且原生支持逻辑滚动。对于原生支持逻辑滚动的布局控件，仍然可以通过将宿主 [Panel](#) 元素放在 [ScrollViewer](#) 中并将 [CanContentScroll](#) 属性设置为 `false` 来实现物理滚动。

以下代码示例演示如何将 `IScrollInfo` 的实例转换为 `StackPanel` 并使用接口定义的内容滚动方法（`LineUp` 和 `LineDown`）。

C#

```
private void spLineUp(object sender, RoutedEventArgs e)
{
    ((IScrollInfo)sp1).LineUp();
}

private void spLineDown(object sender, RoutedEventArgs e)
{
    ((IScrollInfo)sp1).LineDown();
}
```

定义和使用 ScrollViewer 元素

以下示例在包含一些文本和一个矩形的窗口中创建 `ScrollViewer`。 `ScrollBar` 元素仅在需要时出现。重设窗口大小时，由于 `ComputedHorizontalScrollBarVisibility` 和 `ComputedVerticalScrollBarVisibility` 属性的值已更新，`ScrollBar` 元素将在出现后消失。

C#

```
// Create the application's main window
mainWindow = new Window();
mainWindow.Title = "ScrollViewer Sample";

// Define a ScrollViewer
myScrollViewer = new ScrollViewer();
myScrollViewer.HorizontalScrollBarVisibility = ScrollBarVisibility.Auto;

// Add Layout control
myStackPanel = new StackPanel();
myStackPanel.HorizontalAlignment = HorizontalAlignment.Left;
myStackPanel.VerticalAlignment = VerticalAlignment.Top;

TextBlock myTextBlock = new TextBlock();
myTextBlock.TextWrapping = TextWrapping.Wrap;
myTextBlock.Margin = new Thickness(0, 0, 0, 20);
myTextBlock.Text = "Scrolling is enabled when it is necessary. Resize the Window, making it larger and smaller.";

Rectangle myRectangle = new Rectangle();
myRectangle.Fill = Brushes.Red;
myRectangle.Width = 500;
myRectangle.Height = 500;

// Add child elements to the parent StackPanel
myStackPanel.Children.Add(myTextBlock);
myStackPanel.Children.Add(myRectangle);
```

```
// Add the StackPanel as the lone child of the ScrollViewer  
myScrollViewer.Content = myStackPanel;  
  
// Add the ScrollViewer as the Content of the parent Window object  
mainWindow.Content = myScrollViewer;  
mainWindow.Show();
```

XAML

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
      WindowTitle="ScrollViewer Sample">  
  <ScrollViewer HorizontalScrollBarVisibility="Auto">  
    <StackPanel VerticalAlignment="Top" HorizontalAlignment="Left">  
      <TextBlock TextWrapping="Wrap" Margin="0,0,0,20">Scrolling is enabled  
      when it is necessary.  
      Resize the window, making it larger and smaller.</TextBlock>  
      <Rectangle Fill="Red" Width="500" Height="500"></Rectangle>  
    </StackPanel>  
  </ScrollViewer>  
</Page>
```

设置 ScrollViewer 的样式

与 Windows Presentation Foundation 中的所有控件一样，可以设置 [ScrollViewer](#) 的样式以便更改该控件的默认呈现行为。有关控件样式设置的其他信息，请参阅[样式设置和模板化](#)。

对文档进行分页

对于文档内容，一种替代滚动的方法是选择支持分页的文档容器。[FlowDocument](#) 适用于设计为承载于查看控件（例如 [FlowDocumentPageViewer](#)）内的文档，该控件支持跨多个页面的内容分页，从而无需进行滚动。[DocumentViewer](#) 提供了一个用于查看 [FixedDocument](#) 内容的解决方案，该解决方案使用传统的滚动来显示超出显示区域范围的内容。

有关文档格式和演示选项的其他信息，请参阅 [WPF 中的文档](#)。

另请参阅

- [ScrollViewer](#)
- [ScrollBar](#)

- [IScrollInfo](#)
- [如何：创建滚动查看器](#)
- [WPF 中的文档](#)
- [ScrollBar 样式和模板](#)
- [控件](#)

ScrollViewer 帮助主题

项目 • 2023/02/06

本节中的主题演示如何使用 `ScrollViewer` 元素在应用程序中创建可滚动区域。

本节内容

[处理 `ScrollChanged` 事件](#)

[使用 `IScrollInfo` 接口滚动内容](#)

[使用 `ScrollViewer` 的内容滚动方法](#)

参考

[ScrollBar](#)

[ScrollViewer](#)

请参阅

- [面板概述](#)
- [布局](#)

如何：处理 ScrollChanged 事件

项目 • 2023/02/06

示例

此示例演示如何处理 `ScrollViewer` 的 `ScrollChanged` 事件。

具有 `Paragraph` 部件的 `FlowDocument` 元素在 XAML 中定义。由于用户交互而发生 `ScrollChanged` 事件时，将调用处理程序，并且指示事件已发生的文本将写入 `TextBlock`。

XAML

```
<ScrollViewer Name="svrContent" CanContentScroll="False"
    ScrollChanged="sChanged">

    <FlowDocument FontFamily="Arial" PageWidth="400">
        <Paragraph>
            Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam
            nonummy nibh euismod tincidunt ut
            laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim
            veniam, quis nostrud exerci tation
            ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.
            Duis autem vel eum iriure.
            Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam
            nonummy nibh euismod tincidunt ut
            laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim
            veniam, quis nostrud exerci tation
            ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.
            Duis autem vel eum iriure.
            Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam
            nonummy nibh euismod tincidunt ut
            laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim
            veniam, quis nostrud exerci tation
            ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.
            Duis autem vel eum iriure.
        </Paragraph>
```

XAML

```
<Paragraph>
    Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam
    nonummy nibh euismod tincidunt ut
    laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim
    veniam, quis nostrud exerci tation
    ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.
    Duis autem vel eum iriure.
    Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam
```

```

nonummy nibh euismod tincidunt ut
    laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim
    veniam, quis nostrud exerci tation
        ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.
Duis autem vel eum iriure.
    Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam
nonummy nibh euismod tincidunt ut
    laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim
    veniam, quis nostrud exerci tation
        ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.
Duis autem vel eum iriure.
</Paragraph>

</FlowDocument>
<ScrollView>

```

C#

```

private void sChanged(object sender, ScrollChangedEventArgs e)
{
    if (svrContent.CanContentScroll == true)
    {
        tBlock1.Foreground = System.Windows.Media.Brushes.Red;
        tBlock1.Text = "ScrollChangedEvent just Occurred";
        tBlock2.Text = "ExtentHeight is now " + e.ExtentHeight.ToString();
        tBlock3.Text = "ExtentWidth is now " + e.ExtentWidth.ToString();
        tBlock4.Text = "ExtentHeightChange was " +
e.ExtentHeightChange.ToString();
        tBlock5.Text = "ExtentWidthChange was " +
e.ExtentWidthChange.ToString();
        tBlock6.Text = "HorizontalOffset is now " +
e.HorizontalOffset.ToString();
        tBlock7.Text = "VerticalOffset is now " +
e.VerticalOffset.ToString();
        tBlock8.Text = "HorizontalChange was " +
e.HorizontalChange.ToString();
        tBlock9.Text = "VerticalChange was " + e.VerticalChange.ToString();
        tBlock10.Text = "ViewportHeight is now " +
e.ViewportHeight.ToString();
        tBlock11.Text = "ViewportWidth is now " +
e.ViewportWidth.ToString();
        tBlock12.Text = "ViewportHeightChange was " +
e.ViewportHeightChange.ToString();
        tBlock13.Text = "ViewportWidthChange was " +
e.ViewportWidthChange.ToString();
    }
    else
    {
        tBlock1.Text = "";
    }
}

```

另请参阅

- [ScrollViewer](#)
- [ScrollChanged](#)
- [ScrollChangedEventHandler](#)
- [ScrollChangedEventArgs](#)

如何：使用 IScrollInfo 接口来滚动内容

项目 • 2023/02/06

此示例演示如何使用 [IScrollInfo](#) 接口滚动内容。

示例

下面的示例展示 [IScrollInfo](#) 接口的功能。该示例在 Extensible Application Markup Language (XAML) 中创建了一个嵌套在父级 [ScrollView](#) 中的 [StackPanel](#) 元素。通过使用 [IScrollInfo](#) 接口定义的方法并在代码中转换为 [StackPanel](#) (`sp1`) 的实例，可以在逻辑上滚动 [StackPanel](#) 的子元素。

XAML

```
<Border BorderBrush="Black" Background="White" BorderThickness="2"
Width="500" Height="500">
    <ScrollView Name="sv1" CanContentScroll="True"
VerticalScrollBarVisibility="Visible"
HorizontalScrollBarVisibility="Visible">
        <StackPanel Name="sp1">
            <Button>Button 1</Button>
            <Button>Button 2</Button>
            <Button>Button 3</Button>
            <Button>Button 4</Button>
            <Button>Button 5</Button>
            <Rectangle Width="700" Height="500" Fill="Purple"/>
            <TextBlock>Rectangle 1</TextBlock>
            <Rectangle Width="700" Height="500" Fill="Red"/>
            <TextBlock>Rectangle 2</TextBlock>
            <Rectangle Width="700" Height="500" Fill="Green"/>
            <TextBlock>Rectangle 3</TextBlock>
        </StackPanel>
    </ScrollView>
</Border>
```

XAML 文件中的每个 [Button](#) 都会触发一个相关的自定义方法来控制 [StackPanel](#) 中的滚动行为。下面的示例演示如何使用 [LineUp](#) 和 [LineDown](#) 方法。它还概括地演示如何使用 [IScrollInfo](#) 类定义的所有定位方法。

C#

```
private void spLineUp(object sender, RoutedEventArgs e)
{
    ((IScrollInfo)sp1).LineUp();
}

private void spLineDown(object sender, RoutedEventArgs e)
{
```

```
((IScrollInfo)sp1).LineDown();  
}
```

另请参阅

- [ScrollViewer](#)
- [IScrollInfo](#)
- [StackPanel](#)
- [ScrollViewer 概述](#)
- [操作指南主题](#)
- [面板概述](#)

如何：使用 ScrollViewer 的内容滚动方法

项目 • 2023/02/06

此示例演示如何使用 `ScrollViewer` 元素的滚动方法。这些方法通过 `ScrollViewer` 按行或按页增量滚动内容。

示例

以下示例创建一个名为 `sv1` 的 `ScrollViewer`，其承载一个子 `TextBlock` 元素。由于其 `TextBlock` 大于父 `ScrollViewer`，因此显示滚动条以启用滚动。表示各种滚动方法的 `Button` 元素停靠在左侧单独的 `StackPanel` 中。XAML 文件中的每个 `Button` 都调用一个相关的自定义方法来控制 `ScrollViewer` 中的滚动行为。

XAML

```
<StackPanel DockPanel.Dock="Left" Width="150">
    <Button Margin="3,0,0,2" Background="White" Click="svLineUp">Adjust Line Up</Button>
    <Button Margin="3,0,0,2" Background="White" Click="svLineDown">Adjust Line Down</Button>
    <Button Margin="3,0,0,2" Background="White" Click="svLineRight">Adjust Line Right</Button>
    <Button Margin="3,0,0,2" Background="White" Click="svLineLeft">Adjust Line Left</Button>
    <Button Margin="3,0,0,2" Background="White" Click="svPageUp">Adjust Page Up</Button>
    <Button Margin="3,0,0,2" Background="White" Click="svPageDown">Adjust Page Down</Button>
    <Button Margin="3,0,0,2" Background="White" Click="svPageRight">Adjust Page Right</Button>
    <Button Margin="3,0,0,2" Background="White" Click="svPageLeft">Adjust Page Left</Button>
    <TextBlock Name="txt2" TextWrapping="Wrap"/>
</StackPanel>

<Border BorderBrush="Black" Background="White" BorderThickness="2" Height="520" Width="520" VerticalAlignment="Top">
    <ScrollViewer VerticalScrollBarVisibility="Visible" HorizontalScrollBarVisibility="Auto" Name="sv1">
        <TextBlock TextWrapping="Wrap" Width="800" Height="1000" Name="txt1"/>
    </ScrollViewer>
</Border>
```

下面的示例使用 `LineUp` 和 `LineDown` 方法。

C#

```
private void svLineUp(object sender, RoutedEventArgs e)
{
    sv1.LineUp();
}
private void svLineDown(object sender, RoutedEventArgs e)
{
    sv1.LineDown();
}
```

另请参阅

- [ScrollViewer](#)
- [StackPanel](#)

Separator

项目 • 2023/02/06

Separator 控件在控件（例如 [ListBox](#)、[Menu](#) 和 [ToolBar](#)）中的项之间绘制一条水平线或垂直线。

本节内容

参考

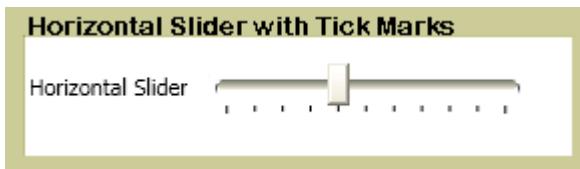
[Separator](#)

Slider

项目 • 2023/02/06

Slider 允许通过沿着 Track 移动 Thumb 来从一系列值中进行选择。

下图演示了一个水平 Slider 控件的示例。



本节内容

[自定义滑块上的刻度](#)

参考

[Slider](#)

[Track](#)

[Thumb](#)

如何：自定义滑块上的刻度

项目 • 2022/09/27

此示例演示如何创建带有刻度线的 [Slider](#) 控件。

示例

将 [TickPlacement](#) 属性设置为除默认值 [None](#) 以外的值时，将显示 [TickBar](#)。

以下示例演示如何创建带有显示刻度线的 [TickBar](#) 的 [Slider](#)。[TickPlacement](#) 和 [TickFrequency](#) 属性定义刻度线的位置以及它们之间的间隔。当移动 [Thumb](#) 时，工具提示将显示 [Slider](#) 的值。[AutoToolTipPlacement](#) 属性定义工具提示的出现位置。由于 [IsSnapToTickEnabled](#) 设置为 `true`，因此 [Thumb](#) 对应于刻度线的位置移动。

下面的示例演示如何使用 [Ticks](#) 属性，沿 [Slider](#) 以不等间隔创建刻度线。

XAML

```
<Slider Width="100" Value="50" Orientation="Horizontal"
        HorizontalAlignment="Left"
        IsSnapToTickEnabled="True" Maximum="3" TickPlacement="BottomRight"
        AutoToolTipPlacement="BottomRight" AutoToolTipPrecision="2"
        Ticks="0, 1.1, 2.5, 3"/>
```

另请参阅

- [Slider](#)
- [TickBar](#)
- [TickPlacement](#)
- [如何：将滑块绑定到属性值](#)

StackPanel

项目 • 2023/02/06

[StackPanel](#) 元素用于水平或垂直堆叠子元素。

本节内容

[操作指南主题](#)

参考

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

相关章节

[布局](#)

[演练：我的第一个 WPF 桌面应用程序](#)

[ScrollViewer 概述](#)

StackPanel 帮助主题

项目 • 2023/02/06

本节的主题介绍如何使用 [StackPanel](#) 元素水平或垂直堆叠内容。

本节内容

[在 StackPanel 和 DockPanel 之间进行选择](#)

[创建 StackPanel](#)

[在 StackPanel 中水平或垂直对齐内容](#)

参考

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

相关章节

[布局](#)

[演练：我的第一个 WPF 桌面应用程序](#)

[ScrollViewer 概述](#)

如何：在 StackPanel 和 DockPanel 之间进行选择

项目 • 2023/02/06

此示例演示在 Panel 中堆叠内容时，如何选择是使用 StackPanel 还是 DockPanel。

示例

尽管可以使用或 DockPanel 或 StackPanel 堆叠子元素，但两个控件并不总是生成相同的结果。例如，放置子元素的顺序可能会影响 DockPanel 中子元素的大小，但不会影响 StackPanel 中子元素的大小。发生这种不同的行为是因为 StackPanel 在 Double.PositiveInfinity 的堆叠方向上测量；但是，DockPanel 仅测量可用大小。

以下示例演示了 DockPanel 和 StackPanel 之间的主要区别。

C#

```
// Create the application's main window
mainWindow = new Window();
mainWindow.Title = "StackPanel vs. DockPanel";

// Add root Grid
myGrid = new Grid();
myGrid.Width = 175;
myGrid.Height = 150;
RowDefinition myRowDef1 = new RowDefinition();
RowDefinition myRowDef2 = new RowDefinition();
myGrid.RowDefinitions.Add(myRowDef1);
myGrid.RowDefinitions.Add(myRowDef2);

// Define the DockPanel
myDockPanel = new DockPanel();
Grid.SetRow(myDockPanel, 0);

//Define an Image and Source
Image myImage = new Image();
BitmapImage bi = new BitmapImage();
bi.BeginInit();
bi.UriSource = new Uri("smiley_stackpanel.png", UriKind.Relative);
bi.EndInit();
myImage.Source = bi;

Image myImage2 = new Image();
BitmapImage bi2 = new BitmapImage();
bi2.BeginInit();
bi2.UriSource = new Uri("smiley_stackpanel.png", UriKind.Relative);
```

```

bi2.EndInit();
myImage2.Source = bi2;

Image myImage3 = new Image();
BitmapImage bi3 = new BitmapImage();
bi3.BeginInit();
bi3.UriSource = new Uri("smiley_stackpanel.PNG", UriKind.Relative);
bi3.EndInit();
myImage3.Stretch = Stretch.Fill;
myImage3.Source = bi3;

// Add the images to the parent DockPanel
myDockPanel.Children.Add(myImage);
myDockPanel.Children.Add(myImage2);
myDockPanel.Children.Add(myImage3);

//Define a StackPanel
myStackPanel = new StackPanel();
myStackPanel.Orientation = Orientation.Horizontal;
Grid.SetRow(myStackPanel, 1);

Image myImage4 = new Image();
BitmapImage bi4 = new BitmapImage();
bi4.BeginInit();
bi4.UriSource = new Uri("smiley_stackpanel.png", UriKind.Relative);
bi4.EndInit();
myImage4.Source = bi4;

Image myImage5 = new Image();
BitmapImage bi5 = new BitmapImage();
bi5.BeginInit();
bi5.UriSource = new Uri("smiley_stackpanel.png", UriKind.Relative);
bi5.EndInit();
myImage5.Source = bi5;

Image myImage6 = new Image();
BitmapImage bi6 = new BitmapImage();
bi6.BeginInit();
bi6.UriSource = new Uri("smiley_stackpanel.PNG", UriKind.Relative);
bi6.EndInit();
myImage6.Stretch = Stretch.Fill;
myImage6.Source = bi6;

// Add the images to the parent StackPanel
myStackPanel.Children.Add(myImage4);
myStackPanel.Children.Add(myImage5);
myStackPanel.Children.Add(myImage6);

// Add the layout panels as children of the Grid
myGrid.Children.Add(myDockPanel);
myGrid.Children.Add(myStackPanel);

// Add the Grid as the Content of the Parent Window Object
mainWindow.Content = myGrid;

```

```
mainWindow.Show();
```

XAML

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      WindowTitle="StackPanel vs. DockPanel">
<Grid Width="175" Height="150">
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>

    <DockPanel Grid.Column="0" Grid.Row="0">
        <Image Source="smiley_stackpanel.png" />
        <Image Source="smiley_stackpanel.png" />
        <Image Source="smiley_stackpanel.png" Stretch="Fill"/>
    </DockPanel>

    <StackPanel Grid.Column="0" Grid.Row="1" Orientation="Horizontal">
        <Image Source="smiley_stackpanel.png" />
        <Image Source="smiley_stackpanel.png" />
        <Image Source="smiley_stackpanel.png" Stretch="Fill"/>
    </StackPanel>
</Grid>
</Page>
```

另请参阅

- StackPanel
- DockPanel
- 面板概述

如何：创建 StackPanel

项目 • 2023/02/06

此示例演示如何创建 StackPanel。

示例

通过 StackPanel，可以在指定方向上堆叠元素。 通过使用在 StackPanel 上定义的属性，内容可垂直流动（默认设置），也可水平流动。

下面的示例可通过使用 StackPanel 垂直堆叠五个 TextBlock 控件，每个控件具有不同的 Border 和 Background。 没有指定 Width 的子元素将拉伸以填充父窗口；但是具有指定 Width 的子元素将在窗口内居中放置。

在 StackPanel 中，默认的堆叠方向为垂直。 若要控制 StackPanel 中的内容流，请使用 Orientation 属性。 可通过使用 HorizontalAlignment 属性控制水平对齐。

XAML

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      WindowTitle="StackPanel Sample">
    <StackPanel>
        <Border Background="SkyBlue" BorderBrush="Black" BorderThickness="1">
            <TextBlock Foreground="Black" FontSize="12">Stacked Item #1</TextBlock>
        </Border>
        <Border Width="400" Background="CadetBlue" BorderBrush="Black" BorderThickness="1">
            <TextBlock Foreground="Black" FontSize="14">Stacked Item #2</TextBlock>
        </Border>
        <Border Background="LightGoldenRodYellow" BorderBrush="Black" BorderThickness="1">
            <TextBlock Foreground="Black" FontSize="16">Stacked Item #3</TextBlock>
        </Border>
        <Border Width="200" Background="PaleGreen" BorderBrush="Black" BorderThickness="1">
            <TextBlock Foreground="Black" FontSize="18">Stacked Item #4</TextBlock>
        </Border>
        <Border Background="White" BorderBrush="Black" BorderThickness="1">
            <TextBlock Foreground="Black" FontSize="20">Stacked Item #5</TextBlock>
        </Border>
    </StackPanel>
</Page>
```

另请参阅

- [StackPanel](#)
- [面板概述](#)
- [操作指南主题](#)

如何：在 StackPanel 中水平或垂直对齐内容

项目 • 2023/02/06

此示例演示如何调整 StackPanel 元素中内容的 Orientation，以及如何调整子内容的 HorizontalAlignment 和 VerticalAlignment。

示例

下面的示例在 Extensible Application Markup Language (XAML) 中创建三个 ListBox 元素。每个 ListBox 表示 StackPanel 的 Orientation、HorizontalAlignment 和 VerticalAlignment 属性的可能值。用户在任何 ListBox 元素中选择值时，StackPanel 及其子 Button 元素的关联属性将更改。

XAML

```
<ListBox VerticalAlignment="Top" SelectionChanged="changeOrientation"
Grid.Row="2" Grid.Column="1" Width="100" Height="50" Margin="0,0,0,10">
    <ListBoxItem>Horizontal</ListBoxItem>
    <ListBoxItem>Vertical</ListBoxItem>
</ListBox>

<ListBox VerticalAlignment="Top" SelectionChanged="changeHorAlign"
Grid.Row="2" Grid.Column="3" Width="100" Height="50" Margin="0,0,0,10">
    <ListBoxItem>Left</ListBoxItem>
    <ListBoxItem>Right</ListBoxItem>
    <ListBoxItem>Center</ListBoxItem>
    <ListBoxItem>Stretch</ListBoxItem>
</ListBox>

<ListBox VerticalAlignment="Top" SelectionChanged="changeVertAlign"
Grid.Row="2" Grid.Column="5" Width="100" Height="50" Margin="0,0,0,10">
    <ListBoxItem>Top</ListBoxItem>
    <ListBoxItem>Bottom</ListBoxItem>
    <ListBoxItem>Center</ListBoxItem>
    <ListBoxItem>Stretch</ListBoxItem>
</ListBox>

<StackPanel Grid.ColumnSpan="6" Grid.Row="3" Name="sp1" Background="Yellow">
    <Button>Button One</Button>
    <Button>Button Two</Button>
    <Button>Button Three</Button>
    <Button>Button Four</Button>
    <Button>Button Five</Button>
    <Button>Button Six</Button>
</StackPanel>
```

下面的代码隐藏文件定义对与 [ListBox](#) 选择更改关联的事件的更改。 StackPanel 由 Name sp1 标识。

C#

```
private void changeOrientation(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    if (li.Content.ToString() == "Horizontal")
    {
        sp1.Orientation = System.Windows.Controls.Orientation.Horizontal;
    }
    else if (li.Content.ToString() == "Vertical")
    {
        sp1.Orientation = System.Windows.Controls.Orientation.Vertical;
    }
}

private void changeHorAlign(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    if (li.Content.ToString() == "Left")
    {
        sp1.HorizontalAlignment = System.Windows.HorizontalAlignment.Left;
    }
    else if (li.Content.ToString() == "Right")
    {
        sp1.HorizontalAlignment = System.Windows.HorizontalAlignment.Right;
    }
    else if (li.Content.ToString() == "Center")
    {
        sp1.HorizontalAlignment = System.Windows.HorizontalAlignment.Center;
    }
    else if (li.Content.ToString() == "Stretch")
    {
        sp1.HorizontalAlignment =
System.Windows.HorizontalAlignment.Stretch;
    }
}

private void changeVertAlign(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    if (li.Content.ToString() == "Top")
    {
        sp1.VerticalAlignment = System.Windows.VerticalAlignment.Top;
    }
    else if (li.Content.ToString() == "Bottom")
    {
        sp1.VerticalAlignment = System.Windows.VerticalAlignment.Bottom;
    }
    else if (li.Content.ToString() == "Center")
    {
```

```
        sp1.VerticalAlignment = System.Windows.VerticalAlignment.Center;
    }
    else if (li.Content.ToString() == "Stretch")
    {
        sp1.VerticalAlignment = System.Windows.VerticalAlignment.Stretch;
    }
}
```

另请参阅

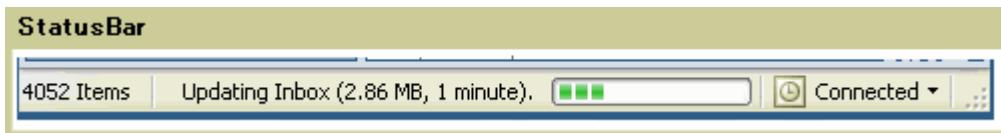
- [StackPanel](#)
- [ListBox](#)
- [HorizontalAlignment](#)
- [VerticalAlignment](#)
- [面板概述](#)

StatusBar

项目 • 2023/02/06

[StatusBar](#) 是窗口底部的水平区域，应用程序可以显示状态信息。

下图显示了 [StatusBar](#) 的一个示例。



本节内容

参考

[StatusBar](#)

[StatusBarItem](#)

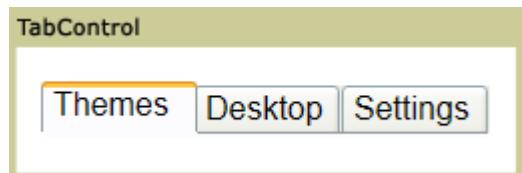
相关章节

TabControl

项目 • 2023/02/06

TabControl 元素显示通过选择相应的选项卡访问的离散页面上的内容。每个选项卡都包含一个 TabItem。

下图显示了一个 TabControl。



典型 TabControl

参考

[TabControl](#)

[TabItem](#)

相关章节

TextBlock

项目 • 2023/02/06

[TextBlock](#) 控件为不需要多个文本段落的 UI 方案提供灵活的文本支持。

本节内容

[TextBlock 概述](#)

参考

[Label](#)

相关章节

[WPF 中的文档](#)

[流文档概述](#)

TextBlock 概述

项目 • 2023/02/06

[TextBlock](#) 控件为不需要多个文本段落的 UI 方案提供灵活的文本支持。它支持许多属性，这些属性支持精确控制表示形式，如 [FontFamily](#)、[FontSize](#)、[FontWeight](#)、[TextEffects](#) 和 [TextWrapping](#)。可以使用 [Text](#) 属性添加文本内容。在 XAML 中使用时，开始标记和结束标记之间的内容隐式添加为元素的文本。

只需使用 XAML 或代码，即可实例化 [TextBlock](#) 元素。

XAML

```
<TextBlock FontSize="18" FontWeight="Bold" FontStyle="Italic">
    Hello, world!
</TextBlock>
```

同样，在代码中使用 [TextBlock](#) 元素也较为简单。

C#

```
TextBlock myTextBlock = new TextBlock();
myTextBlock.FontSize = 18;
myTextBlock.FontWeight = FontWeights.Bold;
myTextBlock.FontStyle = FontStyles.Italic;
myTextBlock.Text = "Hello, world!";
```

另请参阅

- [Label](#)

TextBox

项目 • 2023/02/06

TextBox 控件提供对 WPF 应用程序中的基本文本输入的支持。

本节内容

[TextBox 概述](#)

[操作指南主题](#)

参考

[TextBox](#)

[RichTextBox](#)

[TextBlock](#)

[PasswordBox](#)

请参阅

- [WPF 控件库示例 ↗](#)
- [TextBox 样式和模板](#)

TextBox 概述

项目 • 2023/02/06

借助 [TextBox](#) 类，可以显示或编辑未设置格式的文本。 [TextBox](#) 的常见用途是在窗体中编辑未设置格式的文本。例如，要求输入用户的姓名、电话号码等信息的表单将使用 [TextBox](#) 控件用于文本输入。本主题介绍 [TextBox](#) 类，并提供有关如何在 Extensible Application Markup Language (XAML) 和 C# 中使用它的示例。

使用 TextBox 还是 RichTextBox？

[TextBox](#) 和 [RichTextBox](#) 都允许用户输入文本，但两个控件用于不同场景。[TextBox](#) 需要的系统资源比 [RichTextBox](#) 少，因此非常适合只需要编辑纯文本的场景（例如在窗体中使用）。当用户需要编辑带格式的文本、图像、表格或其他受支持的内容时，[RichTextBox](#) 是更好的选择。例如，编辑需要格式、图像等的文档、文章或博客时，最好使用 [RichTextBox](#) 来实现。下表汇总了 [TextBox](#) 和 [RichTextBox](#) 的主要功能。

控制	实时拼写检查	上下文菜单	格式命令，例如 ToggleBold (Ctrl+B)	FlowDocument 内容，例如图像、段落、表格等
TextBox	是	是	否	否。
RichTextBox	是	是	是（请参阅 RichTextBox 概述）	是（请参阅 RichTextBox 概述 ）

① 备注

虽然 [TextBox](#) 不支持与格式相关的编辑命令（例如 [ToggleBold \(Ctrl+B\)](#)），但是两个控件均支持许多基本命令，例如 [MoveToLineEnd](#)。有关详细信息，请参阅 [EditingCommands](#)。

[TextBox](#) 支持的功能在下面部分中介绍。关于 [RichTextBox](#) 的详细信息，请参阅 [RichTextBox 概述](#)。

实时拼写检查

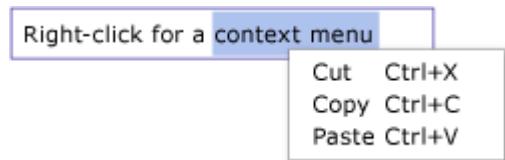
可以在 [TextBox](#) 或 [RichTextBox](#) 中启用实时拼写检查。启用拼写检查时，任何拼写错误的字词下方都会出现红线（见下图）。

The following word is misspelled. |

若要了解如何启用拼写检查，请参阅[在文本编辑控件中启用拼写检查](#)。

上下文菜单

默认情况下，[TextBox](#) 和 [RichTextBox](#) 都有一个上下文菜单，该菜单在用户在控件内右键单击时显示。上下文菜单使用户可以剪切、复制或粘贴（见下图）。



可以创建自己的自定义上下文菜单来重写默认行为。有关详细信息，请参阅[通过 TextBox 使用自定义上下文菜单](#)。

创建 TextBox

[TextBox](#) 的高度可以是单行，也可以包含多行。单行 [TextBox](#) 最适合输入少量纯文本（例如窗体中的“名称”、“电话号码”等）。以下示例演示如何创建单行 [TextBox](#)。

XAML

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <StackPanel>
        <TextBox Width="200" MaxLength="100" />
    </StackPanel>
</Page>
```

还可以创建允许用户输入多行文本的 [TextBox](#)。例如，如果表单要求输入用户的自传概述，将需要使用支持多行文本的 [TextBox](#)。以下示例演示如何使用 Extensible Application Markup Language (XAML) 定义将自动扩展以容纳多行文本的 [TextBox](#) 控件。

XAML

```
<TextBox
    Name="tbMultiLine"
    TextWrapping="Wrap"
    AcceptsReturn="True"
    VerticalScrollBarVisibility="Visible"
>
    This TextBox will allow the user to enter multiple lines of text. When
    the RETURN key is pressed,
    or when typed text reaches the edge of the text box, a new line is
    automatically inserted.
</TextBox>
```

将 [TextWrapping](#) 属性设置为 `Wrap` 会导致输入的文本在到达 [TextBox](#) 控件的边缘时换行，如果需要，会自动扩展 [TextBox](#) 控件以包含新行的空间。

将 [AcceptsReturn](#) 属性设置为 `true` 会导致在按下 RETURN 键时插入新行，如果需要，会再次自动扩展 [TextBox](#) 以包含新行的空间。

[VerticalScrollBarVisibility](#) 属性将滚动条添加到 [TextBox](#)，以便在 [TextBox](#) 扩展超出包围它的框架或窗口的大小时可以滚动 [TextBox](#) 的内容。

有关与使用 [TextBox](#) 相关联的不同任务的详细信息，请参阅[操作说明](#)主题。

检测内容何时更改

通常 [TextChanged](#) 事件应该用于检测 [TextBox](#) 或 [RichTextBox](#) 中文本的更改，而不是你可能认为的 [KeyDown](#)。有关示例，请参阅[检测 TextBox 中的文本何时更改](#)。

另请参阅

- [操作指南主题](#)
- [RichTextBox 概述](#)

TextBox 帮助主题

项目 • 2023/02/06

本部分提供的示例演示如何使用 [TextBox](#) 控件完成常见任务。

本节内容

- [创建多行 TextBox 控件](#)
- [检测 TextBox 中的文本何时更改](#)
- [在 TextBox 控件中启用制表符](#)
- [从 TextBox 获取文本行集合](#)
- [将 TextBox 控件设为只读](#)
- [将光标置于 TextBox 控件中文本的开头或结尾](#)
- [检索选定文本内容](#)
- [在 TextBox 控件中设置焦点](#)
- [设置 TextBox 控件的文本内容](#)
- [在文本编辑控件中启用拼写检查](#)
- [将自定义上下文菜单与 TextBox 结合使用](#)
- [将拼写检查功能与上下文菜单结合使用](#)
- [向 TextBox 添加水印](#)

参考

[TextBox](#)

[RichTextBox](#)

[TextBlock](#)

[PasswordBox](#)

请参阅

- [WPF 控件库示例 ↗](#)
- [TextBox 样式和模板](#)

如何：创建多行 TextBox 控件

项目 • 2023/02/06

此示例演示如何使用 Extensible Application Markup Language (XAML) 定义将自动扩展以容纳多行文本的 [TextBox](#) 控件。

示例

将 [TextWrapping](#) 属性设置为 Wrap 会导致输入的文本在到达 [TextBox](#) 控件的边缘时换行，如果需要，会自动扩展 [TextBox](#) 控件以包含新行的空间。

将 [AcceptsReturn](#) 属性设置为 true 会导致在按下 RETURN 键时插入新行，如果需要，会再次自动扩展 [TextBox](#) 以包含新行的空间。

[VerticalScrollBarVisibility](#) 属性将滚动条添加到 [TextBox](#)，以便在 [TextBox](#) 扩展超出包围它的框架或窗口的大小时可以滚动 [TextBox](#) 的内容。

XAML

```
<TextBox
    Name="tbMultiLine"
    TextWrapping="Wrap"
    AcceptsReturn="True"
    VerticalScrollBarVisibility="Visible"
>
    This TextBox will allow the user to enter multiple lines of text. When
    the RETURN key is pressed,
        or when typed text reaches the edge of the text box, a new line is
    automatically inserted.
</TextBox>
```

另请参阅

- [TextWrapping](#)
- [TextBox 概述](#)
- [RichTextBox 概述](#)

如何：检测 TextBox 中的文本何时更改

项目 • 2022/09/27

此示例显示了一种在 [TextBox 控件](#) 中的文本发生更改时使用 [TextChanged 事件](#) 执行方法的方法。

在 XAML (包含要监视更改的 [TextBox 控件](#)) 的代码隐藏类中，插入一个在 [TextChanged 事件](#) 触发时要调用的方法。此方法的签名必须与 [TextChangedEventHandler 委托](#) 的预期签名相匹配。

只要用户或以编程方式更改 [TextBox 控件](#) 的内容，就会调用事件处理程序。

① 备注

此事件在创建 [TextBox 控件](#) 并初始填充文本时触发。

定义 TextBox 控件

在定义 [TextBox 控件](#) 的 Extensible Application Markup Language (XAML) 中，使用与事件处理程序方法名称匹配的值指定 [TextChanged 属性](#)。

XAML

```
<TextBox TextChanged="textChangedEventHandler">
    Here is the initial text in my TextBox. Each time the contents of this
    TextBox are changed,
    the TextChanged event fires and textChangedEventHandler is called.
</TextBox>
```

监视 TextBox 控件的更改

在 XAML (包含要监视更改的 [TextBox 控件](#)) 的代码隐藏类中，插入一个在 [TextChanged 事件](#) 触发时要调用的方法。此方法的签名必须与 [TextChangedEventHandler 委托](#) 的预期签名相匹配。

C#

```
// TextChangedEventHandler delegate method.
private void textChangedEventHandler(object sender, TextChangedEventArgs
args)
{
    // Omitted Code: Insert code that does something whenever
```

```
// the text changes...
} // end textChangedEventHandler
```

只要用户或以编程方式更改 `TextBox` 控件的内容，就会调用事件处理程序。

① 备注

此事件在创建 `TextBox` 控件并初始填充文本时触发。

注释

另请参阅

- [TextChangedEventArgs](#)
- [TextBox 概述](#)
- [RichTextBox 概述](#)

如何：在 TextBox 控件中启用制表符

项目 • 2023/02/06

此示例演示如何在 [TextBox](#) 控件中允许接受制表符作为输入。

示例

若要允许在 [TextBox](#) 控件中接受制表符作为输入，请将 [AcceptsTab](#) 属性设置为 true。

XAML

```
<TextBox AcceptsTab="True">
    If the AcceptsTab element is "True", the TextBox control will accept tab
    characters as regular input when the TAB key is pressed.
    If AcceptsTab is "False" (the default), pressing TAB moves the focus to
    the next focusable control.
</TextBox>
```

另请参阅

- [TextBox 概述](#)
- [RichTextBox 概述](#)

如何：从 TextBox 获得线条集合

项目 • 2023/02/06

此示例演示如何从 [TextBox](#) 获得文本行集合。

示例

以下演示了一个简单的方法，该方法以 [TextBox](#) 为自变量，返回 [StringCollection](#)，其中包含“TextBox”中的文本行。 [LineCount](#) 属性用于确定“TextBox”中当前有多少行，然后使用 [GetLineText](#) 方法提取每一行并将其添加到行集合。

C#

```
StringCollection GetLinesCollectionFromTextBox(TextBox textBox)
{
    StringCollection lines = new StringCollection();

    // lineCount may be -1 if TextBox layout info is not up-to-date.
    int lineCount = textBox.LineCount;

    for (int line = 0; line < lineCount; line++)
        // GetLineText takes a zero-based line index.
        lines.Add(textBox.GetLineText(line));

    return lines;
}
```

另请参阅

- [TextBox 概述](#)
- [RichTextBox 概述](#)

如何：将 TextBox 控件设为只读

项目 • 2023/02/06

此示例演示如何将 [TextBox](#) 控件配置为不允许用户输入或修改。

示例

若要防止用户修改 [TextBox](#) 控件的内容，请将 [IsReadOnly](#) 特性设置为 `True`。

XAML

```
<TextBox  
    IsReadOnly="True"  
>  
    The user may not modify the contents of this TextBox.  
</TextBox>
```

[IsReadOnly](#) 特性仅影响用户输入，不会影响 [TextBox](#) 控件的 Extensible Application Markup Language (XAML) 描述中设置的文本，或通过 [Text](#) 属性以编程方式设置的文本。

[IsReadOnly](#) 的默认值为 `False`。

另请参阅

- [TextBox 概述](#)
- [RichTextBox 概述](#)

如何：将光标置于 TextBox 控件中的文本的开头或末尾

项目 • 2023/02/06

此示例演示如何将光标置于 [TextBox](#) 控件的文本内容的开头或末尾。

定义 TextBox 控件

以下 Extensible Application Markup Language (XAML) 代码描述了 [TextBox](#) 控件并为其分配名称。

XAML

```
<TextBox  
    Name="tbPositionCursor"  
>  
    Here is some text in my text box...  
</TextBox>
```

将光标定位在起始位置

若要将光标置于 [TextBox](#) 控件中内容的开头，请调用 [Select](#) 方法，并指定选择内容的起始位置为 0，选择长度为 0。

C#

```
tbPositionCursor.Select(0, 0);
```

将光标定位在末尾

若要将光标置于 [TextBox](#) 控件中内容的末尾，请调用 [Select](#) 方法，并指定选择内容的起始位置与文本内容的长度相等，选择长度为 0。

C#

```
tbPositionCursor.Select(tbPositionCursor.Text.Length, 0);
```

另请参阅

- [TextBox 概述](#)
- [RichTextBox 概述](#)

如何：检索文本选定内容

项目 • 2023/02/06

此示例演示了使用 [SelectedText](#) 属性检索用户在 [TextBox](#) 控件中选择的文本的一种方法。

定义 TextBox 控件

下面的 Extensible Application Markup Language (XAML) 示例演示了包含一些要选择的文本的 [TextBox](#) 控件和具有指定 [OnClick](#) 方法的 [Button](#) 控件的定义。

在此示例中，使用带有关联 [Click](#) 事件处理程序的按钮来检索文本选择。当用户单击该按钮时，[OnClick](#) 方法将文本框中的任何选定文本复制到字符串中。检索文本选择的特定情况（单击按钮），以及对该选择执行的操作（将文本选择复制到字符串）可以轻松修改，以适应各种场景。

XAML

```
<TextBox Name="tbSelectSomeText">
    Some text to select...
</TextBox>

<Button Click="OnClick">Retrieve Selection</Button>
```

OnClick 事件处理程序

以下 C# 示例显示了在 XAML 中为此示例定义的按钮的 [OnClick](#) 事件处理程序。

C#

```
void OnClick(object sender, RoutedEventArgs e)
{
    String sSelectedText = tbSelectSomeText.SelectedText;
}
```

另请参阅

- [TextBox 概述](#)
- [RichTextBox 概述](#)

如何：设置 TextBox 控件中的焦点

项目 • 2023/02/06

此示例演示如何使用 [Focus](#) 方法在 [TextBox](#) 控件上设置焦点。

定义简单的 TextBox 控件

以下 Extensible Application Markup Language (XAML) 代码描述了一个名为 tbFocusMe 的简单 [TextBox](#) 控件

XAML

```
<TextBox Name="tbFocusMe">
    This is the text in my text box.
</TextBox>
```

在 TextBox 控件上设置焦点

以下示例调用 [Focus](#) 方法，在名为 tbFocusMe 的 [TextBox](#) 控件上设置焦点。

C#

```
tbFocusMe.Focus();
```

另请参阅

- [Focusable](#)
- [IsFocused](#)
- [TextBox 概述](#)
- [RichTextBox 概述](#)

如何：设置 TextBox 控件的文本内容

项目 • 2022/09/27

此示例演示如何使用 [Text](#) 属性设置 TextBox 控件的初始文本内容。

① 备注

尽管示例的可扩展应用程序标记语言 (XAML) 版本可以在每个按钮的 TextBox 内容的文本周围使用 `<TextBox.Text>` 标记，但没有必要，因为 TextBox 会将 ContentPropertyAttribute 特性应用于 Text 属性。有关详细信息，请参阅 [WPF 中的 XAML](#)。

使用文本属性设置文本内容

XAML

```
<TextBox Name="tbSettingText">
    Initial text contents of the TextBox.
</TextBox>
```

设置 TextBox 控件的文本内容

C#

```
tbSettingText.Text = "Initial text contents of the TextBox.";
```

另请参阅

- [TextBox 概述](#)
- [RichTextBox 概述](#)

如何：在文本编辑控件中启用拼写检查

项目 • 2023/02/06

以下示例演示如何使用 `SpellCheck` 类的 `.IsEnabled` 属性在 `TextBox` 中启用实时拼写检查。

示例

XAML

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

    <StackPanel>
        <TextBox SpellCheck.IsEnabled="True" Name="myTextBox"></TextBox>
    </StackPanel>

</Page>
```

C#

```
using System;
using System.Windows;
using System.Windows.Controls;

namespace SDKSample
{
    public partial class SpellCheckExample : Page
    {
        public SpellCheckExample()
        {
            StackPanel myStackPanel = new StackPanel();

            //Create TextBox
            TextBox myTextBox = new TextBox();
            myTextBox.Width = 200;

            // Enable spellchecking on the TextBox.
            myTextBox.SpellCheck.IsEnabled = true;

            // Alternatively, the SetIsEnabled method could be used
            // to enable or disable spell checking like this:
            // SpellCheck.SetIsEnabled(myTextBox, true);

            myStackPanel.Children.Add(myTextBox);
            this.Content = myStackPanel;
        }
    }
}
```

```
    }  
}
```

另请参阅

- [将拼写检查功能与上下文菜单结合使用](#)
- [TextBox 概述](#)
- [RichTextBox 概述](#)

如何：通过 TextBox 使用自定义上下文菜单

项目 • 2023/02/06

此示例演示如何为 [TextBox](#) 定义和实现简单的自定义上下文菜单。

定义自定义上下文菜单

以下 Extensible Application Markup Language (XAML) 示例定义包含自定义上下文菜单的 [TextBox](#) 控件。

上下文菜单是使用 [ContextMenu](#) 元素定义的。 上下文菜单本身由一系列 [MenuItem](#) 元素和 [Separator](#) 元素组成。 每个 [MenuItem](#) 元素在上下文菜单中定义命令；属性 [Header](#) 定义菜单命令的显示文本，并且 [Click](#) 属性为每个菜单项指定处理程序方法。 [Separator](#) 元素只会导致在上一个菜单项和后续菜单项之间呈现分隔线。

XAML

```
<TextBox
    Name="c xm TextBox"
    Grid.Row="1"
    AcceptsReturn="True"
    AcceptsTab="True"
    VerticalScrollBarVisibility="Visible"
    TextWrapping="Wrap"
>
<TextBox.ContextMenu>
    <ContextMenu
        Name="c xm"
        Opened="C xm Opened"
    >
        <MenuItem
            Header="Cut"
            Name="c xm ItemCut"
            Click="ClickCut"
        />
        <MenuItem
            Header="Copy"
            Name="c xm ItemCopy"
            Click="ClickCopy"
        />
        <MenuItem
            Header="Paste"
            Name="c xm ItemPaste"
            Click="ClickPaste"
        />
    <Separator/>
```

```

<MenuItem
    Header="Select All"
    Name="c xmItemSelectAll"
    Click="ClickSelectAll"
/>
<MenuItem
    Header="Select Current Line"
    Name="c xmItemSelectLine"
    Click="ClickSelectLine"
/>
<Separator/>
<MenuItem
    Header="Undo Last Action"
    Name="c xmItemUndo"
    Click="ClickUndo"
/>
<MenuItem
    Header="Redo Last Action"
    Name="c xmItemRedo"
    Click="ClickRedo"
/>
<Separator/>
<MenuItem
    Header="Clear All Text"
    Name="c xmItemClear"
    Click="ClickClear"
/>
</ContextMenu>
</TextBox.ContextMenu>
This TextBox uses a simple custom context menu. The context menu can be
disabled by checking
the CheckBox above, which simply sets the TextBox.ContextMenu property to
null.
</TextBox>

```

实现自定义上下文菜单

以下示例显示了上述上下文菜单定义的实现代码，以及启用或禁用上下文菜单的代码。[Opened](#) 事件用于根据 [TextBox](#) 的当前状态动态启动或禁用特定命令。

若要还原默认上下文菜单，请使用 [ClearValue](#) 方法清除 [ContextMenu](#) 属性的值。如需完全禁用上下文菜单，可以将 [ContextMenu](#) 属性设置为 null 引用（Visual Basic 中为 [Nothing](#)）。

C#

```

private void MenuChange(Object sender, RoutedEventArgs args)
{
    RadioButton rb = sender as RadioButton;
    if (rb == null || c xm == null) return;

```

```

        switch (rb.Name)
    {
        case "rbCustom":
            cxmTextBox.ContextMenu = cxm;
            break;
        case "rbDefault":
            // Clearing the value of the ContextMenu property
            // restores the default TextBox context menu.
            cxmTextBox.ClearValue(ContextMenuProperty);
            break;
        case "rbDisabled":
            // Setting the ContextMenu property to
            // null disables the context menu.
            cxmTextBox.ContextMenu = null;
            break;
        default:
            break;
    }
}

void ClickPaste(Object sender, RoutedEventArgs args)      {
    cxmTextBox.Paste(); }

void ClickCopy(Object sender, RoutedEventArgs args)       {
    cxmTextBox.Copy(); }

void ClickCut(Object sender, RoutedEventArgs args)        { cxmTextBox.Cut(); }

void ClickSelectAll(Object sender, RoutedEventArgs args) {
    cxmTextBox.SelectAll(); }

void ClickClear(Object sender, RoutedEventArgs args)      {
    cxmTextBox.Clear(); }

void ClickUndo(Object sender, RoutedEventArgs args)       {
    cxmTextBox.Undo(); }

void ClickRedo(Object sender, RoutedEventArgs args)       {
    cxmTextBox.Redo(); }

void ClickSelectLine(Object sender, RoutedEventArgs args)
{
    int lineIndex =
    cxmTextBox.GetLineIndexFromCharacterIndex(cxmTextBox.CaretIndex);
    int lineStartingCharIndex =
    cxmTextBox.GetCharacterIndexFromLineIndex(lineIndex);
    int lineLength = cxmTextBox.GetLineLength(lineIndex);
    cxmTextBox.Select(lineStartingCharIndex, lineLength);
}

void CxmOpened(Object sender, RoutedEventArgs args)
{
    // Only allow copy/cut if something is selected to copy/cut.
    if (c xmTextBox.SelectedText == "")
        cxmItemCopy.IsEnabled = cxmItemCut.IsEnabled = false;
    else
        cxmItemCopy.IsEnabled = cxmItemCut.IsEnabled = true;

    // Only allow paste if there is text on the clipboard to paste.
}

```

```
if (Clipboard.ContainsText())
    cxmItemPaste.IsEnabled = true;
else
    cxmItemPaste.IsEnabled = false;
}
```

另请参阅

- [将拼写检查功能与上下文菜单结合使用](#)
- [TextBox 概述](#)
- [RichTextBox 概述](#)

如何：使用上下文菜单中的拼写检查功能

项目 • 2023/02/06

默认情况下，在编辑控件（如 [TextBox](#) 或 [RichTextBox](#)）中启用拼写检查功能后，可在上下文菜单中获取拼写检查选项。例如，当用户右键单击拼写错误的单词时，他们会收到一组拼写建议或“全部忽略”选项。但是，使用自己的自定义上下文菜单替代默认上下文菜单时，此功能将丢失，并且需要编写代码才可在上下文菜单中重新启用拼写检查功能。以下示例演示如何在 [TextBox](#) 上启用此功能。

定义上下文菜单

以下示例演示创建了一个 [TextBox](#) 的 Extensible Application Markup Language (XAML)，其中包含一些用于实现上下文菜单的事件。

XAML

```
<Page x:Class="SDKSample.SpellerCustomContextMenu"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Loaded="OnWindowLoaded">

    <TextBox
        Name="myTextBox"
        TextWrapping="Wrap"
        SpellCheck.IsEnabled="True"
        ContextMenuOpening="tb_ContextMenuOpening">
        In a custom menu you need to write code to add speler choices
        because everything in a custom context menu has to be added explicitly.
    </TextBox>

</Page>
```

实现上下文菜单

以下示例演示实现上下文菜单的代码。

C#

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Media;
using System.Windows.Media.Imaging;
```

```
using System.Windows.Shapes;

namespace SDKSample
{
    public partial class SpellerCustomContextMenu : Page
    {

        void OnWindowLoaded(object sender, RoutedEventArgs e)
        {
            //This is required for the first time ContextMenu invocation so
            //that TextEditor doesnt handle it.
            myTextBox.ContextMenu = GetContextMenu();
        }
        void tb_ContextMenuItemOpening(object sender, RoutedEventArgs e)
        {
            int caretIndex, cmdIndex;
            SpellingError spellingError;

            myTextBox.ContextMenu = GetContextMenu();
            caretIndex = myTextBox.CaretIndex;

            cmdIndex = 0;
            spellingError = myTextBox.GetSpellingError(caretIndex);
            if (spellingError != null)
            {
                foreach (string str in spellingError.Suggestions)
                {
                    MenuItem mi = new MenuItem();
                    mi.Header = str;
                    mi.FontWeight = FontWeights.Bold;
                    mi.Command = EditingCommands.CorrectSpellingError;
                    mi.CommandParameter = str;
                    mi.CommandTarget = myTextBox;
                    myTextBox.ContextMenu.Items.Insert(cmdIndex, mi);
                    cmdIndex++;
                }
                Separator separatorMenuItem1 = new Separator();
                myTextBox.ContextMenu.Items.Insert(cmdIndex,
                    separatorMenuItem1);
                cmdIndex++;
                MenuItem ignoreAllMI = new MenuItem();
                ignoreAllMI.Header = "Ignore All";
                ignoreAllMI.Command = EditingCommands.IgnoreSpellingError;
                ignoreAllMI.CommandTarget = myTextBox;
                myTextBox.ContextMenu.Items.Insert(cmdIndex, ignoreAllMI);
                cmdIndex++;
                Separator separatorMenuItem2 = new Separator();
                myTextBox.ContextMenu.Items.Insert(cmdIndex,
                    separatorMenuItem2);
            }
        }

        // Gets a fresh context menu.
        private ContextMenu GetContextMenu()
        {
```

```

ContextMenu cm = new ContextMenu();

//Can create STATIC custom menu items if exists here...
MenuItem m1, m2, m3, m4;
m1 = new MenuItem();
m1.Header = "File";
m2 = new MenuItem();
m2.Header = "Save";
m3 = new MenuItem();
m3.Header = "SaveAs";
m4 = new MenuItem();
m4.Header = "Recent Files";

//Can add functionality for the custom menu items here...

cm.Items.Add(m1);
cm.Items.Add(m2);
cm.Items.Add(m3);
cm.Items.Add(m4);

return cm;
}
}
}

```

使用 [RichTextBox](#) 执行此操作的代码与此类似。 主要区别在于传递给 `GetSpellingError` 方法的参数。 对于 [TextBox](#)，传递插入符号位置的整数索引：

```
spellingError = myTextBox.GetSpellingError(caretIndex);
```

对于 [RichTextBox](#)，传递指定插入符号位置的 [TextPointer](#)：

```
spellingError = myRichTextBox.GetSpellingError(myRichTextBox.CaretPosition);
```

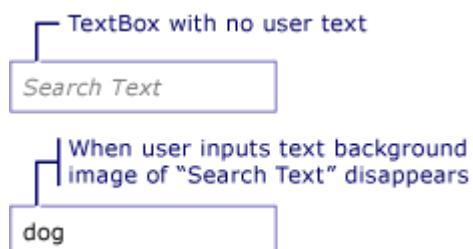
另请参阅

- [TextBox 概述](#)
- [RichTextBox 概述](#)
- [在文本编辑控件中启用拼写检查](#)
- [将自定义上下文菜单与 TextBox 结合使用](#)

如何：在 TextBox 中添加水印

项目 • 2023/02/06

下面的示例演示如何通过在 TextBox 内显示解释性背景图像来帮助 TextBox 的可用性，直到用户输入文本，此时图像将被删除。此外，如果用户删除了其输入，则再次还原背景图像。请参阅下图。



① 备注

在此示例中使用背景图像而不是简单地操作 TextBox 的 Text 属性的原因是背景图像不会干扰数据绑定。

示例

XAML

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.TextBoxBackgroundExample"
>

<StackPanel>
    <TextBox Name="myTextBox" TextChanged="OnTextBoxTextChanged"
Width="200">
        <TextBox.Background>
            <ImageBrush ImageSource="TextBoxBackground.gif" AlignmentX="Left"
Stretch="None" />
        </TextBox.Background>
    </TextBox>
</StackPanel>
</Page>
```

C#

```
using System;
using System.Windows;
using System.Windows.Input;
```

```
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Media.Imaging;

namespace SDKSample
{
    public partial class TextBoxBackgroundExample : Page
    {

        void OnTextBoxTextChanged(object sender, TextChangedEventArgs e)
        {

            if (myTextBox.Text == "")
            {
                // Create an ImageBrush.
                ImageBrush textImageBrush = new ImageBrush();
                textImageBrush.ImageSource =
                    new BitmapImage(
                        new Uri(@"TextBoxBackground.gif", UriKind.Relative)
                    );
                textImageBrush.AlignmentX = AlignmentX.Left;
                textImageBrush.Stretch = Stretch.None;
                // Use the brush to paint the button's background.
                myTextBox.Background = textImageBrush;
            }
            else
            {

                myTextBox.Background = null;
            }
        }
    }
}
```

另请参阅

- [TextBox 概述](#)
- [RichTextBox 概述](#)

ToolBar

项目 • 2023/02/06

ToolBar 控件是一组命令或控件的容器，这些命令或控件的功能通常相关。

下图显示了水平和垂直 ToolBar 控件。



水平工具栏



垂直工具栏

本节内容

[ToolBar 概述](#)

[设置 ToolBar 上控件的样式](#)

参考

[ToolBar](#)

[ToolBarTray](#)

相关章节

ToolBar 概述

项目 • 2023/02/06

ToolBar 控件是通常有关功能的一组命令或控件的容器。ToolBar 通常包含调用命令的按钮。

ToolBar 控件

ToolBar 控件的名称因其按钮或其他控件像条形栏一样排列成一行或一列而得名。WPF ToolBar 控件提供一种溢出机制，可将大小受限的ToolBar 中无法自然容纳的任何项放入特殊的溢出区域。此外，WPF ToolBar 控件通常与相关的ToolBarTray 控件结合使用，可提供特殊的布局行为，并支持用户调整工具栏大小和排列。

在ToolBarTray 中指定ToolBar 位置

使用 Band 和 BandIndex 属性在ToolBarTray 中定位ToolBar。Band 指示ToolBar 在其父级ToolBarTray 中的位置。BandIndex 指示ToolBar 在其区段内的排列顺序。下面的示例演示如何使用此属性在ToolBarTray 中放置ToolBar 控件。

XAML

```
<ToolBarTray Background="White">
    <ToolBar Band="1" BandIndex="1">
        <Button>
            <Image Source="toolbargraphics\cut.bmp" />
        </Button>
        <Button>
            <Image Source="toolbargraphics\copy.bmp" />
        </Button>
        <Button>
            <Image Source="toolbargraphics\paste.bmp" />
        </Button>
    </ToolBar>
    <ToolBar Band="2" BandIndex="1">
        <Button>
            <Image Source="toolbargraphics\undo.bmp" />
        </Button>
        <Button>
            <Image Source="toolbargraphics\redo.bmp" />
        </Button>
    </ToolBar>
    <ToolBar Band="2" BandIndex="2">
        <Button>
            <Image Source="toolbargraphics\paint.bmp" />
        </Button>
        <Button>
```

```
<Image Source="toolbargraphics\spell.bmp" />
</Button>
<Separator/>
<Button>
    <Image Source="toolbargraphics\save.bmp" />
</Button>
<Button>
    <Image Source="toolbargraphics\open.bmp" />
</Button>
</ToolBar>
</ToolBarTray>
```

带有溢出项的ToolBar

ToolBar 控件包含的项数经常超出工具栏大小可容纳的项数。当发生此情况时，ToolBar 会显示“溢出”按钮。若要查看溢出项，用户可单击“溢出”按钮，这些项将显示在ToolBar 下方的弹出窗口中。下图显示一个具有溢出项的ToolBar：



可以通过将 `ToolBar.OverflowMode` 附加属性设置为 `OverflowMode.Always`、`OverflowMode.Never` 或 `OverflowMode.AsNeeded` 来指定何时将工具栏上的项放置在溢出面板上。以下示例指定工具栏上的最后四个按钮应始终在溢出面板上。

XAML

```
<ToolBarTray Background="White">
    <ToolBar Band="1" BandIndex="1">
        <Button>
            <Image Source="toolbargraphics\cut.bmp" />
        </Button>
        <Button>
            <Image Source="toolbargraphics\copy.bmp" />
        </Button>
        <Button>
            <Image Source="toolbargraphics\paste.bmp" />
        </Button>
        <Button>
            <Image Source="toolbargraphics\undo.bmp" />
        </Button>
        <Button>
            <Image Source="toolbargraphics\redo.bmp" />
        </Button>
        <Button>
            <Image Source="toolbargraphics\paint.bmp" />
        </Button>
    </ToolBar>
</ToolBarTray>
```

```
<Button>
    <Image Source="toolbargraphics\spell.bmp" />
</Button>
<Separator/>
<ButtonToolBar.OverflowMode="Always">
    <Image Source="toolbargraphics\save.bmp" />
</Button>
<ButtonToolBar.OverflowMode="Always">
    <Image Source="toolbargraphics\open.bmp" />
</Button>
<ButtonToolBar.OverflowMode="Always">
    <Image Source="toolbargraphics\print.bmp" />
</Button>
<ButtonToolBar.OverflowMode="Always">
    <Image Source="toolbargraphics\preview.bmp" />
</Button>
</ToolBar>
</ToolBarTray>
```

ToolBar 在其 ControlTemplate 中使用 ToolBarPanel 和 ToolBarOverflowPanel。 ToolBarPanel 负责工具栏上的项的布局。 ToolBarOverflowPanel 负责处理 ToolBar 上不适合的项的布局。 关于用于 ToolBar 的 ControlTemplate 示例，请参阅

[ToolBar 样式和模板。](#)

另请参阅

- [ToolBarPanel](#)
- [ToolBarOverflowPanel](#)
- [设置 ToolBar 上控件的样式](#)
- [WPF 控件库示例](#)

如何：设置 ToolBar 上的控件的样式

项目 • 2023/02/06

ToolBar 定义 ResourceKey 对象来指定 ToolBar 中控件的样式。若要对 ToolBar 中的控件设置样式，请将样式的 `x:key` 属性设置为 ToolBar 中定义的 ResourceKey。

ToolBar 定义了以下 ResourceKey 对象：

- ButtonStyleKey
- CheckBoxStyleKey
- ComboBoxStyleKey
- MenuStyleKey
- RadioButtonStyleKey
- SeparatorStyleKey
- TextBoxStyleKey
- ToggleButtonStyleKey

示例

下面的示例为 ToolBar 中的控件定义样式。

XAML

```
<Window.Resources>

    <!--Styles for controls in a toolbar.-->
    <Style x:Key="{x:Static ToolBar.SeparatorStyleKey}"
TargetType="Separator">
        <Setter Property="Background" Value="DarkBlue"/>
        <Setter Property="Width" Value="2"/>
    </Style>

    <Style x:Key="{x:Static ToolBar.ButtonStyleKey}" TargetType="Button">
        <Setter Property="Foreground" Value="Blue"/>
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="HorizontalAlignment" Value="Center"/>
        <Setter Property="VerticalAlignment" Value="Center"/>
    </Style>

    <Style x:Key="{x:Static ToolBar.CheckBoxStyleKey}" TargetType="CheckBox">
        <Setter Property="Foreground" Value="DarkSlateBlue"/>
    </Style>
```

```

<Setter Property="FontSize" Value="14"/>
<Setter Property="HorizontalAlignment" Value="Center"/>
<Setter Property="VerticalAlignment" Value="Center"/>
</Style>

<Style x:Key="{x:StaticToolBar.MenuStyleKey}" TargetType="Menu">
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="FontStyle" Value="Italic"/>
    <Setter Property="FontWeight" Value="Bold"/>
    <Setter Property="Background" Value="LightSteelBlue"/>
</Style>

<Style x:Key="{x:StaticToolBar.RadioButtonStyleKey}" TargetType="RadioButton">
    <Setter Property="Background" Value="LightSteelBlue"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
</Style>

<Style x:Key="{x:StaticToolBar.TextBoxStyleKey}" TargetType="TextBox">
    <Setter Property="Background" Value="DarkBlue"/>
    <Setter Property="Foreground" Value="White"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="FontStyle" Value="Italic"/>
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
    <Setter Property="Width" Value="75"/>
</Style>

<Style x:Key="{x:StaticToolBar.ComboBoxStyleKey}" TargetType="ComboBox">
    <Setter Property="Background" Value="LightSteelBlue"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="MinWidth" Value="60"/>
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
</Style>
<!--*****Styles for controls that are not in a toolbar.*****-->
<Style TargetType="Separator">
    <Setter Property="Background" Value="DarkBlue"/>
    <Setter Property="Width" Value="2"/>
</Style>

<Style TargetType="Button">
    <Setter Property="Foreground" Value="Blue"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
</Style>

<Style TargetType="CheckBox">
    <Setter Property="Foreground" Value="DarkSlateBlue"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
</Style>

```

```

</Style>

<Style TargetType="Menu">
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="FontStyle" Value="Italic"/>
    <Setter Property="FontWeight" Value="Bold"/>
    <Setter Property="Background" Value="LightSteelBlue"/>
</Style>

<Style TargetType="RadioButton">
    <Setter Property="Background" Value="LightSteelBlue"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
</Style>

<Style TargetType="TextBox">
    <Setter Property="Background" Value="DarkBlue"/>
    <Setter Property="Foreground" Value="White"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="FontStyle" Value="Italic"/>
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
    <Setter Property="Width" Value="75"/>
</Style>

<Style TargetType="ComboBox">
    <Setter Property="Background" Value="LightSteelBlue"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="MinWidth" Value="60"/>
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
</Style>
</Window.Resources>

```

XAML

```

<ToolBarTray Margin="10,10,3,3"
              Grid.Column="0" Grid.Row="2"
              Background="LightBlue">
    <ToolBar >
        <Button Content="Button 1"/>
        <Button Content="Button 2"/>
        <Separator/>
        <CheckBox Content="CheckBox 1"/>
        <CheckBox Content="CheckBox 2"/>
        <Separator/>
        <RadioButton>One</RadioButton>
        <RadioButton>Two</RadioButton>
        <Separator/>
        <ComboBox>
            <ComboBoxItem IsSelected="True">Item 1</ComboBoxItem>
            <ComboBoxItem>Item 2</ComboBoxItem>

```

```
<ComboBoxItem>Item 3</ComboBoxItem>
<ComboBoxItem>Item 4</ComboBoxItem>
</ComboBox>
<TextBox/>
<Separator/>
<Menu>
    <MenuItem Header="Menu">
        <MenuItem Header="File">
            <MenuItem Header="Copy"/>
            <MenuItem Header="Paste"/>
        </MenuItem>
    </MenuItem>
</Menu>
</ToolBar>
</ToolBarTray>
```

请参阅

- [样式设置和模板化](#)

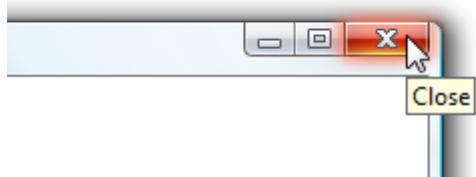
ToolTip

项目 • 2023/02/06

工具提示是一个小型的弹出窗口，在用户将鼠标指针暂停在某个元素（如 [Button](#)）上时显示。

下图显示了指向“关闭”[Button](#) 的鼠标指针，随后显示了该按钮的标识 [ToolTip](#)。

显示工具提示的“关闭”按钮



本节内容

[ToolTip 概述](#)

[操作指南主题](#)

参考

[ToolTip](#)

[ToolTipService](#)

[Popup](#)

相关章节

[Popup 概述](#)

[操作指南主题](#)

ToolTip 概述

项目 • 2022/09/27

工具提示是一个小型的弹出窗口，在用户将鼠标指针暂停在某个元素（如 [Button](#)）上时显示。本主题介绍工具提示，并讨论如何创建和自定义工具提示内容。

什么是工具提示

当用户将鼠标指针移动到具有工具提示的元素上时，将在一段指定的时间内显示一个包含工具提示内容（例如，介绍控件功能的文本内容）的窗口。如果用户将鼠标指针从控件上移开，该窗口将消失，因为工具提示内容无法接收焦点。

工具提示的内容可以包含一行或多行文本、图像、形状或其他可视内容。通过将以下属性之一设置为工具提示内容来定义控件的工具提示。

- [FrameworkContentElement.ToolTip](#)
- [FrameworkElement.ToolTip](#)

使用哪个属性取决于定义工具提示的控件继承自 [FrameworkContentElement](#) 还是 [FrameworkElement](#) 类。

创建工作提示

以下示例演示如何通过将 [Button](#) 控件的 [ToolTip](#) 属性设置为文本字符串来创建简单的工具提示。

XAML

```
<Button ToolTip="Click to submit your information"
        Click="SubmitCode" Height="20" Width="50">Submit</Button>
```

还可以将工具提示定义为 [ToolTip](#) 对象。以下示例使用 XAML 将 [ToolTip](#) 对象指定为 [TextBox](#) 元素的工具提示。请注意，该示例通过设置 [FrameworkElement.ToolTip](#) 属性来指定 [ToolTip](#)。

XAML

```
<TextBox HorizontalAlignment="Left">ToolTip with non-text content
<TextBox.ToolTip>
<ToolTip>
<DockPanel Width="50" Height="70">
    <Image Source="data\flower.jpg"/>
```

```
<TextBlock>Useful information goes here.</TextBlock>
</DockPanel>
</ToolTip>
</TextBox.ToolTip>
</TextBox>
```

以下示例使用代码来生成 [ToolTip](#) 对象。该示例创建了一个 [ToolTip](#) (tt) , 并将其与 [Button](#) 关联。

C#

```
button = new Button();
button.Content = "Hover over me.";
tt = new ToolTip();
tt.Content = "Created with C#";
button.ToolTip = tt;
cv2.Children.Add(button);
```

还可以通过将工具提示内容包含在布局元素 (如 [DockPanel](#)) 中来创建未定义为 [ToolTip](#) 对象的工具提示内容。下面的示例演示如何将 [TextBox](#) 的 [ToolTip](#) 属性设置为包含在 [DockPanel](#) 控件中的内容。

XAML

```
<TextBox>
    ToolTip with image and text
    <TextBox.ToolTip>
        <StackPanel>
            <Image Source="data\flower.jpg"/>
            <TextBlock>Useful information goes here.</TextBlock>
        </StackPanel>
    </TextBox.ToolTip>
```

使用 [ToolTipd](#) 和 [ToolTipService](#) 类的属性

可以通过设置视觉属性和应用样式来自定义工具提示内容。如果将工具提示内容定义为 [ToolTip](#) 对象，可以设置 [ToolTip](#) 对象的视觉属性。否则，必须在 [ToolTipService](#) 类上设置等效的附加属性。

有关如何设置属性以使用 [ToolTip](#) 和 [ToolTipService](#) 属性指定工具提示内容位置的示例，请参阅[定位 ToolTip](#)。

设置工具提示样式

可以通过定义自定义 Style 来设置 ToolTip 的样式。下面的示例定义一个名为 Simple 的 Style，它演示如何通过设置 Background、Foreground、FontSize 和 FontWeight 来偏移 ToolTip 的位置并更改其外观。

XAML

```
<Style TargetType="ToolTip">
    <Setter Property = "HorizontalOffset" Value="10"/>
    <Setter Property = "VerticalOffset" Value="10"/>
    <Setter Property = "Background" Value="LightBlue"/>
    <Setter Property = "Foreground" Value="Purple"/>
    <Setter Property = "FontSize" Value="14"/>
    <Setter Property = "FontWeight" Value="Bold"/>
</Style>
```

使用 ToolTipService 的 Time Interval 属性

ToolTipService 类提供以下属性用于设置工具提示显示时间：InitialShowDelay、BetweenShowDelay 和 ShowDuration。

使用 InitialShowDelay 和 ShowDuration 属性指定显示 ToolTip 之前的延迟（通常很短暂），并且还指定 ToolTip 保持可见的时长。有关详细信息，请参阅[如何：延迟 ToolTip 的显示](#)。

BetweenShowDelay 属性确定当用户在不同控件之间快速移动鼠标指针时，这些控件的工具提示是否在没有初始延迟的情况下显示。有关 BetweenShowDelay 属性的详细信息，请参阅[使用 BetweenShowDelay 属性](#)。

以下示例演示如何为工具提示设置这些属性。

XAML

```
<Ellipse Height="25" Width="50"
    Fill="Gray"
    HorizontalAlignment="Left"
    ToolTipService.InitialShowDelay="1000"
    ToolTipService.ShowDuration="7000"
    ToolTipService.BetweenShowDelay="2000">
<Ellipse.ToolTip>
    <ToolTip Placement="Right"
        PlacementRectangle="50,0,0,0"
        HorizontalOffset="10"
        VerticalOffset="20"
        HasDropShadow="false"
        Opened="whenToolTipOpens"
        Closed="whenToolTipCloses"
        >
<BulletDecorator>
```

```
<BulletDecorator.Bullet>
  <Ellipse Height="10" Width="20" Fill="Blue"/>
</BulletDecorator.Bullet>
<TextBlock>Uses the ToolTip Class</TextBlock>
</BulletDecorator>
</ToolTip>
</Ellipse.ToolTip>
</Ellipse>
```

另请参阅

- [ToolTipService](#)
- [ToolTip](#)
- [ToolTipEventArgs](#)
- [ToolTipEventHandler](#)
- [操作指南主题](#)

ToolTip 帮助主题

项目 • 2023/02/06

本节内容

[定位 ToolTip](#)

[使用 BetweenShowDelay 属性](#)

参考

[ToolTip](#)

[ToolTipService](#)

[Popup](#)

相关章节

[Popup 概述](#)

[操作指南主题](#)

如何：定位 ToolTip

项目 • 2023/02/06

此示例显示如何指定工具提示在屏幕上的位置。

示例

可以使用在 [ToolTip](#) 和 [ToolTipService](#) 类中定义的五个属性集来定位工具提示。 下表显示这两组五个属性，并根据类提供指向其参考文档的链接。

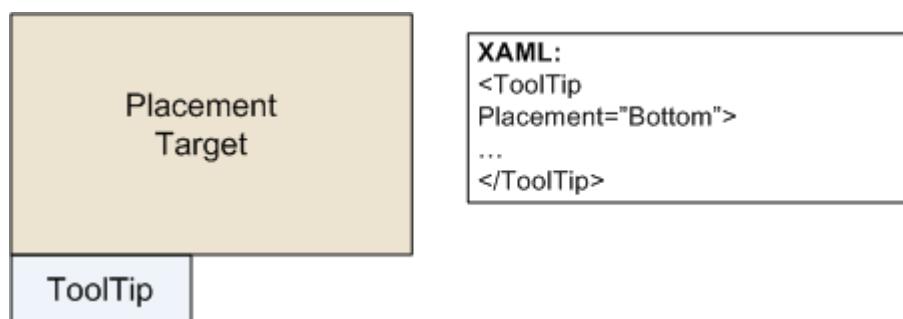
根据类对应的工具提示属性

System.Windows.Controls.ToolTip 类属性	System.Windows.Controls.ToolTipService 类属性
ToolTip.Placement	ToolTipService.Placement
ToolTip.PlacementTarget	ToolTipService.PlacementTarget
ToolTip.PlacementRectangle	ToolTipService.PlacementRectangle
ToolTip.HorizontalOffset	ToolTipService.HorizontalOffset
ToolTip.VerticalOffset	ToolTipService.VerticalOffset

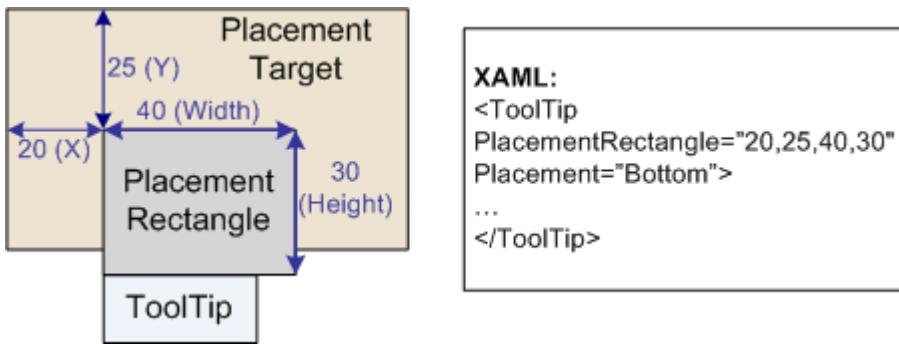
如果使用 [ToolTip](#) 对象定义工具提示的内容，则可以使用任一类的属性；但是，[ToolTipService](#) 属性优先。对于未定义为 [ToolTip](#) 对象的工具提示，请使用 [ToolTipService](#) 属性。

下图显示如何使用这些属性定位工具提示。虽然这些图中的 Extensible Application Markup Language (XAML) 示例显示了如何设置由 [ToolTip](#) 类定义的属性，但 [ToolTipService](#) 类的相应属性遵循相同的布局规则。有关 Placement 属性的可能值的详细信息，请参阅 [Popup 放置行为](#)。

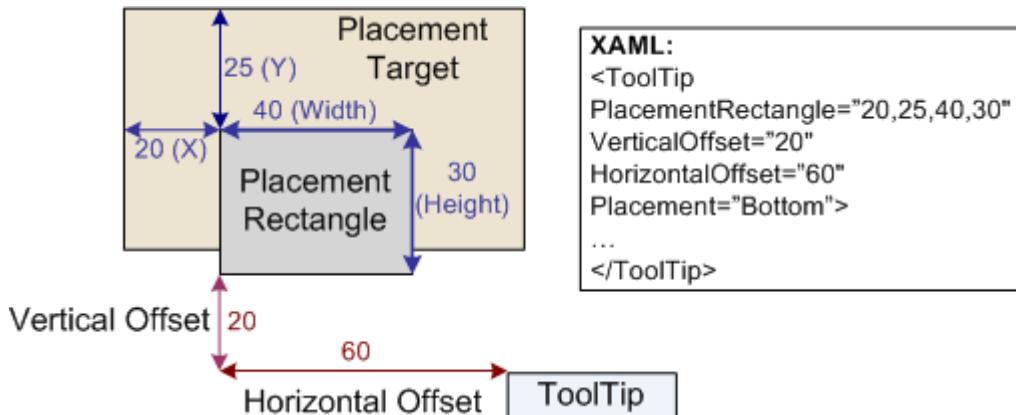
下图显示使用 Placement 属性放置工具提示：



下图显示使用 Placement 和 PlacementRectangle 属性放置工具提示：



下图显示使用 Placement、PlacementRectangle 和 Offset 属性放置工具提示：



以下示例显示如何使用 `ToolTip` 属性指定其内容为 `ToolTip` 对象的工具提示的位置。

XAML

```

<Ellipse Height="25" Width="50"
Fill="Gray"
HorizontalAlignment="Left"
ToolTipService.InitialShowDelay="1000"
ToolTipService.ShowDuration="7000"
ToolTipService.BetweenShowDelay="2000">
<Ellipse.ToolTip>
<ToolTip Placement="Right"
PlacementRectangle="50,0,0,0"
HorizontalOffset="10"
VerticalOffset="20"
HasDropShadow="false"
Opened="whenToolTipOpens"
Closed="whenToolTipCloses"
>
<BulletDecorator>
<BulletDecorator.Bullet>
<Ellipse Height="10" Width="20" Fill="Blue"/>
</BulletDecorator.Bullet>
<TextBlock>Uses the ToolTip Class</TextBlock>
</BulletDecorator>
</ToolTip>
</Ellipse.ToolTip>
</Ellipse>

```

C#

```
//Create an ellipse that will have a
//ToolTip control.
Ellipse ellipse1 = new Ellipse();
ellipse1.Height = 25;
ellipse1.Width = 50;
ellipse1.Fill = Brushes.Gray;
ellipse1.HorizontalAlignment = HorizontalAlignment.Left;

//Create a tooltip and set its position.
ToolTip tooltip = new ToolTip();
tooltip.Placement = PlacementMode.Right;
tooltip.PlacementRectangle = new Rect(50, 0, 0, 0);
tooltip.HorizontalOffset = 10;
tooltip.VerticalOffset = 20;

//Create BulletDecorator and set it
//as the tooltip content.
BulletDecorator bdec = new BulletDecorator();
Ellipse littleEllipse = new Ellipse();
littleEllipse.Height = 10;
littleEllipse.Width = 20;
littleEllipse.Fill = Brushes.Blue;
bdec.Bullet = littleEllipse;
TextBlock tipText = new TextBlock();
tipText.Text = "Uses the ToolTip class";
bdec.Child = tipText;
tooltip.Content = bdec;

//set tooltip on ellipse
ellipse1.ToolTip = tooltip;
```

以下示例显示如何使用 [ToolTipService](#) 属性指定其内容并非 [ToolTip](#) 对象的工具提示的位置。

XAML

```
<Ellipse Height="25" Width="50"
    Fill="Gray"
    HorizontalAlignment="Left"
    ToolTipService.InitialShowDelay="1000"
    ToolTipService.ShowDuration="7000"
    ToolTipService.BetweenShowDelay="2000"
    ToolTipService.Placement="Right"
    ToolTipService.PlacementRectangle="50,0,0,0"
    ToolTipService.HorizontalOffset="10"
    ToolTipService.VerticalOffset="20"
    ToolTipService.HasDropShadow="false"
    ToolTipService.ShowOnDisabled="true"
    ToolTipService.IsEnabled="true"
    ToolTipOpening="whenToolTipOpens"
```

```
    ToolTipClosing="whenToolTipCloses"
    >
<Ellipse.ToolTip>
    <BulletDecorator>
        <BulletDecorator.Bullet>
            <Ellipse Height="10" Width="20" Fill="Blue"/>
        </BulletDecorator.Bullet>
        <TextBlock>Uses the ToolTipService class</TextBlock>
    </BulletDecorator>
</Ellipse.ToolTip>
</Ellipse>
```

C#

```
//Create and Ellipse with the BulletDecorator as
//the tooltip
Ellipse ellipse2 = new Ellipse();
ellipse2.Name = "ellipse2";
this.RegisterName(ellipse2.Name, ellipse2);
ellipse2.Height = 25;
ellipse2.Width = 50;
ellipse2.Fill = Brushes.Gray;
ellipse2.HorizontalAlignment = HorizontalAlignment.Left;

//set tooltip timing
ToolTipService.SetInitialShowDelay(ellipse2, 1000);
ToolTipService.SetBetweenShowDelay(ellipse2, 2000);
ToolTipService.SetShowDuration(ellipse2, 7000);

//set tooltip placement

ToolTipService.SetPlacement(ellipse2, PlacementMode.Right);

ToolTipService.SetPlacementRectangle(ellipse2,
    new Rect(50, 0, 0, 0));

ToolTipService.SetHorizontalOffset(ellipse2, 10.0);

ToolTipService.SetVerticalOffset(ellipse2, 20.0);

ToolTipService.SetHasDropShadow(ellipse2, false);

ToolTipService.SetIsEnabled(ellipse2, true);

ToolTipService.SetShowOnDisabled(ellipse2, true);

ellipse2.AddHandler(ToolTipService.ToolTipOpeningEvent,
    new RoutedEventHandler(whenToolTipOpens));
ellipse2.AddHandler(ToolTipService.ToolTipClosingEvent,
    new RoutedEventHandler(whenToolTipCloses));
```

```
//define tooltip content
BulletDecorator bdec2 = new BulletDecorator();
Ellipse littleEllipse2 = new Ellipse();
littleEllipse2.Height = 10;
littleEllipse2.Width = 20;
littleEllipse2.Fill = Brushes.Blue;
bdec2.Bullet = littleEllipse2;
TextBlock tipText2 = new TextBlock();
tipText2.Text = "Uses the ToolTipService class";
bdec2.Child = tipText2;
ToolTipService.SetToolTip(ellipse2, bdec2);
stackPanel_1_2.Children.Add(ellipse2);
```

另请参阅

- [ToolTip](#)
- [ToolTipService](#)
- [操作指南主题](#)
- [ToolTip 概述](#)

如何：使用 BetweenShowDelay 属性

项目 • 2023/02/06

本示例演示如何使用 [BetweenShowDelay](#) 时间属性，以便在用户将鼠标指针从一个工具提示直接移动到另一个工具提示时，工具提示可快速显示（几乎没有延迟）。

示例

在下面的示例中，对于两个 [Ellipse](#) 控件的工具提示，[InitialShowDelay](#) 属性设置为 1 秒（1000 毫秒），[BetweenShowDelay](#) 设置为 2 秒（2000 毫秒）。如果显示其中一个省略号的工具提示，然后在两秒内将鼠标指针移动到另一个省略号并暂停不动，则第二个省略号的工具提示将立即显示。

在以下任一方案中，应用 [InitialShowDelay](#) 会导致第二个省略号的工具提示在等待一秒后才会出现：

- 如果移动到第二个按钮所需的时间超过两秒。
- 如果第一个省略号的工具提示在时间间隔开始时不可见。

XAML

```
<Ellipse Height="25" Width="50"
    Fill="Gray"
    HorizontalAlignment="Left"
    ToolTipService.InitialShowDelay="1000"
    ToolTipService.ShowDuration="7000"
    ToolTipService.BetweenShowDelay="2000">
<Ellipse.ToolTip>
    <ToolTip Placement="Right"
        PlacementRectangle="50,0,0,0"
        HorizontalOffset="10"
        VerticalOffset="20"
        HasDropShadow="false"
        Opened="whenToolTipOpens"
        Closed="whenToolTipCloses"
        >
        <BulletDecorator>
            <BulletDecorator.Bullet>
                <Ellipse Height="10" Width="20" Fill="Blue"/>
            </BulletDecorator.Bullet>
            <TextBlock>Uses the ToolTip Class</TextBlock>
        </BulletDecorator>
    </ToolTip>
</Ellipse.ToolTip>
</Ellipse>
```

XAML

```
<Ellipse Height="25" Width="50"
    Fill="Gray"
    HorizontalAlignment="Left"
    ToolTipService.InitialShowDelay="1000"
    ToolTipService.ShowDuration="7000"
    ToolTipService.BetweenShowDelay="2000"
    ToolTipService.Placement="Right"
    ToolTipService.PlacementRectangle="50,0,0,0"
    ToolTipService.HorizontalOffset="10"
    ToolTipService.VerticalOffset="20"
    ToolTipService.HasDropShadow="false"
    ToolTipService.ShowOnDisabled="true"
    ToolTipService.IsEnabled="true"
    ToolTipOpening="whenToolTipOpens"
    ToolTipClosing="whenToolTipCloses"
    >
<Ellipse.ToolTip>
    <BulletDecorator>
        <BulletDecorator.Bullet>
            <Ellipse Height="10" Width="20" Fill="Blue"/>
        </BulletDecorator.Bullet>
        <TextBlock>Uses the ToolTipService class</TextBlock>
    </BulletDecorator>
</Ellipse.ToolTip>
</Ellipse>
```

另请参阅

- [ToolTip](#)
- [ToolTipService](#)
- [操作指南主题](#)
- [ToolTip 概述](#)

TreeView

项目 • 2023/02/06

[TreeView](#) 控件使用可折叠的节点来显示层次结构中的信息。

下图为 [TreeView](#) 控件示例，该控件具有嵌套的 [TreeViewItem](#) 控件：

```
⊖ Employee1
    Jesper Aaberg
    ⊖ Employee Number
        12345
    ⊖ Work Days
        Monday
        Tuesday
        Thursday
⊕ Employee2
```

本节内容

[TreeView 概述](#)

[操作指南主题](#)

参考

[TreeView](#)

[TreeViewItem](#)

相关章节

[数据绑定概述](#)

[数据模板化概述](#)

TreeView 概述

项目 • 2023/02/06

TreeView 控件使用可折叠的节点提供显示层次结构中的信息的方法。本主题介绍 TreeView 和 TreeViewItem 控件，并提供简单的用法示例。

什么是 TreeView？

TreeView 是使用 TreeViewItem 控件嵌套项的 ItemsControl。以下示例创建了一个 TreeView。

XAML

```
<TreeView Name="myTreeViewEvent" >
    <TreeViewItem Header="Employee1" IsSelected="True">
        <TreeViewItem Header="Jesper Aaberg"/>
        <TreeViewItem Header="Employee Number">
            <TreeViewItem Header="12345"/>
        </TreeViewItem>
        <TreeViewItem Header="Work Days">
            <TreeViewItem Header="Monday"/>
            <TreeViewItem Header="Tuesday"/>
            <TreeViewItem Header="Thursday"/>
        </TreeViewItem>
    </TreeViewItem>
    <TreeViewItem Header="Employee2">
        <TreeViewItem Header="Dominik Paiha"/>
        <TreeViewItem Header="Employee Number">
            <TreeViewItem Header="98765"/>
        </TreeViewItem>
        <TreeViewItem Header="Work Days">
            <TreeViewItem Header="Tuesday"/>
            <TreeViewItem Header="Wednesday"/>
            <TreeViewItem Header="Friday"/>
        </TreeViewItem>
    </TreeViewItem>
</TreeView>
```

创建 TreeView

TreeView 控件包含 TreeViewItem 控件的层次结构。TreeViewItem 控件是具有 Header 和 Items 集合的 HeaderedItemsControl。

如果使用 Extensible Application Markup Language (XAML) 定义 TreeView，则可以显式定义 TreeViewItem 控件的 Header 内容以及标记其集合的项。上图演示了此方法。

还可以指定 `ItemsSource` 作为数据源，然后指定 `HeaderTemplate` 和 `ItemTemplate` 以定义 `TreeViewItem` 内容。

要定义 `TreeViewItem` 控件的布局，还可使用 `HierarchicalDataTemplate` 对象。有关详细信息和示例，请参阅[使用 `SelectedValue`、`SelectedValuePath` 和 `SelectedItem`](#)。

如果某个项不是 `TreeViewItem` 控件，当显示 `TreeView` 控件时，该项自动包含在 `TreeViewItem` 控件中。

展开和折叠 `TreeViewItem`

如果用户展开 `TreeViewItem`，则 `IsExpanded` 属性将设置为 `true`。还可以通过将 `IsExpanded` 属性设置为 `true`（展开）或 `TreeViewItem`（折叠）来展开或折叠 `false`，而无需任何直接的用户操作。当此属性发生更改时，会发生 `Expanded` 或 `Collapsed` 事件。

在 `TreeViewItem` 控件上调用 `BringIntoView` 方法时，`TreeViewItem` 及其父 `TreeViewItem` 控件将展开。如果 `TreeViewItem` 不可见或部分可见，则 `TreeView` 将滚动以使其可见。

`TreeViewItem` 选择

当用户单击 `TreeViewItem` 控件以将其选中时，将发生 `Selected` 事件，并且其 `IsSelected` 属性将设置为 `true`。`TreeViewItem` 也将成为 `TreeView` 控件的 `SelectedItem`。相反，当所选内容从 `TreeViewItem` 控件更改为其他内容时，将发生 `Unselected` 事件，并且其 `IsSelected` 属性将设置为 `false`。

`TreeView` 控件上的 `SelectedItem` 属性是只读属性，因此无法显式设置它。如果用户单击 `TreeViewItem` 控件或者当 `TreeViewItem` 控件上的 `IsSelected` 属性设置为 `true` 时，则会设置 `SelectedItem` 属性。

使用 `SelectedValuePath` 属性指定 `SelectedItem` 的 `SelectedValue`。有关详细信息，请参阅[使用 `SelectedValue`、`SelectedValuePath` 和 `SelectedItem`](#)。

可以在 `SelectedItemChanged` 事件上注册事件处理程序，以便确定所选的 `TreeViewItem` 何时更改。向事件处理程序提供的 `RoutedPropertyChangedEventArgs<T>` 指定 `OldValue`（上一个选择）和 `NewValue`（当前选择）。如果应用程序或用户未进行上一个或当前选择，则任一值都可能为 `null`。

`TreeView` 样式

针对 [TreeView](#) 控件的默认样式是将其放在包含 [ScrollViewer](#) 控件的 [StackPanel](#) 对象内部。设置 [TreeView](#) 的 [Width](#) 和 [Height](#) 属性时，这些值用于调整显示 [TreeView](#) 的 [StackPanel](#) 对象的大小。如果要显示的内容大于显示区域，[ScrollViewer](#) 将自动显示，以便用户可以滚动浏览 [TreeView](#) 内容。

若要自定义 [TreeViewItem](#) 控件的外观，请将 [Style](#) 属性设置为自定义 [Style](#)。

下面的示例演示如何使用 [Style](#) 设置 [TreeViewItem](#) 控件的 [Foreground](#) 和 [FontSize](#) 属性值。

XAML

```
<Style TargetType="{x:Type TreeViewItem}">
    <Setter Property="Foreground" Value="Blue"/>
    <Setter Property="FontSize" Value="12"/>
</Style>
```

向 [TreeView](#) 项添加图像和其他内容

可在 [TreeViewItem](#) 的 [Header](#) 内容中包含多个对象。若要将多个对象包含在 [Header](#) 内容中，请将这些对象包含在布局控件内，如 [Panel](#) 或 [StackPanel](#)。

以下示例演示如何将 [TreeViewItem](#) 的 [Header](#) 定义为 [CheckBox](#) 和 [TextBlock](#)，这两者都包含在 [DockPanel](#) 控件中。

XAML

```
<TreeViewItem>
    <TreeViewItem.Header>
        <DockPanel>
            <CheckBox/>
            <TextBlock>
                TreeViewItem Text
            </TextBlock>
        </DockPanel>
    </TreeViewItem.Header>
</TreeViewItem>
```

以下示例演示如何定义包含 [Image](#) 和 [TextBlock](#)（这两者都包含在 [DockPanel](#) 控件中）的 [DataTemplate](#)。可以使用 [DataTemplate](#) 设置 [TreeViewItem](#) 的 [HeaderTemplate](#) 或 [ItemTemplate](#)。

XAML

```
<DataTemplate x:Key="NewspaperTVItem">
    <DockPanel>
```

```
<Image Source="images\icon.jpg"/>
<TextBlock VerticalAlignment="center" Text ="{Binding Path=Name}"/>
</DockPanel>
</DataTemplate>
```

另请参阅

- [TreeView](#)
- [TreeViewItem](#)
- [操作指南主题](#)
- [WPF 内容模型](#)

TreeView 帮助主题

项目 • 2023/02/06

本节中的主题介绍如何使用 [TreeView](#) 控件在分层结构中显示信息。

本节内容

[创建简单或复杂的 TreeView](#)

[使用 SelectedValue、SelectedValuePath 和 SelectedItem](#)

[将 TreeView 绑定到深度无法确定的数据](#)

[提升 TreeView 的性能](#)

[在 TreeView 中查找 TreeViewItem](#)

参考

[TreeView](#)

[TreeViewItem](#)

相关章节

如何：创建简单或复杂的 TreeView

项目 • 2023/02/06

此示例演示如何创建简单或复杂的 TreeView 控件。

TreeView 由 TreeViewItem 控件的层次结构组成，其中可以包含简单的文本字符串，也可包含更复杂的内容，例如 Button 控件或嵌入了内容的 StackPanel。你可显式定义 TreeViewItem 内容，也可由数据源提供内容。本主题提供这些概念的示例。

示例

TreeViewItem 的 Header 属性包含 TreeView 为该项显示的内容。TreeViewItem 还可以将 TreeViewItem 控件作为其子元素，你可使用 Items 属性定义这些子元素。

以下示例演示如何通过将 Header 属性设置为文本字符串来显式定义 TreeViewItem 内容。

XAML

```
<TreeView>
    <TreeViewItem Header="Employee1">
        <TreeViewItem Header="Jesper"/>
        <TreeViewItem Header="Aaberg"/>
        <TreeViewItem Header="12345"/>
    </TreeViewItem>
    <TreeViewItem Header="Employee2">
        <TreeViewItem Header="Dominik"/>
        <TreeViewItem Header="Paiha"/>
        <TreeViewItem Header="98765"/>
    </TreeViewItem>
</TreeView>
```

以下示例演示如何通过定义作为 Button 控件的 Items 来定义 TreeViewItem 的子元素。

XAML

```
<TreeView>
    <TreeViewItem Header ="Employee1">
        <TreeViewItem.Items>
            <Button>Jesper</Button>
            <Button>Aaberg</Button>
            <Button>12345</Button>
        </TreeViewItem.Items>
    </TreeViewItem>
    <TreeViewItem Header="Employee2">
        <TreeViewItem.Items>
            <Button>Dominik</Button>
        </TreeViewItem.Items>
    </TreeViewItem>
</TreeView>
```

```
<Button>Paiha</Button>
<Button>98765</Button>
</TreeViewItem.Items>
</TreeViewItem>
</TreeView>
```

以下示例演示如何创建一个 [XmlDataProvider](#) 提供 [TreeViewItem](#) 内容且 [HierarchicalDataTemplate](#) 定义内容外观的 [TreeView](#)。

XAML

```
<XmlDataProvider x:Key="myEmployeeData" XPath="/EmployeeData">
  <x:XData>
    <EmployeeData xmlns="">
      <EmployeeInfo>
        <EmployeeInfoData>Employee1</EmployeeInfoData>
        <Item Type="FirstName">Jesper</Item>
        <Item Type="LastName">Aaberg</Item>
        <Item Type="EmployeeNumber">12345</Item>
      </EmployeeInfo>
      <EmployeeInfo>
        <EmployeeInfoData>Employee2</EmployeeInfoData>
        <Item Type="FirstName">Dominik</Item>
        <Item Type="LastName">Paiha</Item>
        <Item Type="EmployeeNumber">98765</Item>
      </EmployeeInfo>
    </EmployeeData>
  </x:XData>
</XmlDataProvider>
```

XAML

```
<HierarchicalDataTemplate DataType="EmployeeInfo"
  ItemsSource ="{Binding XPath=Item}">
  <TextBlock Text="{Binding XPath=EmployeeInfoData}" />
</HierarchicalDataTemplate>
```

XAML

```
<TreeView ItemsSource="{Binding Source={StaticResource myEmployeeData},
  XPath=EmployeeInfo}" />
```

以下示例演示如何创建一个 [TreeViewItem](#) 内容包含带有嵌入内容的 [DockPanel](#) 控件的 [TreeView](#)。

XAML

```
<TreeView>
  <TreeViewItem Header="Animals">
```

```
<TreeViewItem.Items>
<DockPanel>
    <Image Source="data\fish.png"/>
    <TextBlock Margin="5" Foreground="Brown"
        FontSize="12">Fish</TextBlock>
</DockPanel>
<DockPanel>
    <Image Source="data\dog.png"/>
    <TextBlock Margin="5" Foreground="Brown"
        FontSize="12">Dog</TextBlock>
</DockPanel>
<DockPanel>
    <Image Source="data\cat.png"/>
    <TextBlock Margin="5" Foreground="Brown"
        FontSize="12">Cat</TextBlock>
</DockPanel>
</TreeViewItem.Items>
</TreeViewItem>
</TreeView>
```

另请参阅

- [TreeView](#)
- [TreeViewItem](#)
- [TreeView 概述](#)
- [操作指南主题](#)

如何：使用 SelectedValue、 SelectedValuePath 和 SelectedItem

项目 • 2023/02/06

此示例演示如何使用 [SelectedValue](#) 和 [SelectedValuePath](#) 属性为 [TreeView](#) 的 [SelectedItem](#) 指定值。

示例

[SelectedValuePath](#) 属性提供了为 [TreeView](#) 中的 [SelectedItem](#) 指定 [SelectedValue](#) 的方法。[SelectedItem](#) 表示 [Items](#) 集合中的对象，[TreeView](#) 显示选定项的单个属性的值。[SelectedValuePath](#) 属性指定用于确定 [SelectedValue](#) 属性的值的属性的路径。本主题中的示例说明了此概念。

下面的示例演示包含员工信息的 [XmlDataProvider](#)。

XAML

```
<XmlDataProvider x:Key="myEmployeeData" XPath="/EmployeeData">
  <x:XData>
    <EmployeeData xmlns="">
      <EmployeeInfo>
        <EmployeeName>Jesper Aabergy</EmployeeName>
        <EmployeeWorkDay>Monday</EmployeeWorkDay>
        <EmployeeWorkDay>Wednesday</EmployeeWorkDay>
        <EmployeeWorkDay>Friday</EmployeeWorkDay>
        <EmployeeStartTime>8:00am</EmployeeStartTime>
        <EmployeeNumber>12345</EmployeeNumber>
      </EmployeeInfo>
      <EmployeeInfo>
        <EmployeeName>Dominik Paiha</EmployeeName>
        <EmployeeWorkDay>Monday</EmployeeWorkDay>
        <EmployeeWorkDay>Tuesday</EmployeeWorkDay>
        <EmployeeStartTime>6:30am</EmployeeStartTime>
        <EmployeeNumber>98765</EmployeeNumber>
      </EmployeeInfo>
    </EmployeeData>
  </x:XData>
</XmlDataProvider>
```

下面的示例定义了显示 [Employee](#) 的 [EmployeeName](#) 和 [EmployeeWorkDay](#) 的 [HierarchicalDataTemplate](#)。请注意，[HierarchicalDataTemplate](#) 不会将 [EmployeeNumber](#) 指定为模板的一部分。

XAML

```
<HierarchicalDataTemplate x:Key="SampleTemplate" DataType="EmployeeInfo"
    ItemsSource ="{Binding XPath=EmployeeWorkDay}">
    <TextBlock Text="{Binding XPath=EmployeeName}" />
</HierarchicalDataTemplate>
```

下面的示例演示一个 `TreeView`，它使用以前定义的 `HierarchicalDataTemplate`，并将 `SelectedValue` 属性设置为 `EmployeeNumber`。在 `TreeView` 中选择 `EmployeeName` 时，`SelectedItem` 属性将返回与所选 `EmployeeName` 对应的 `EmployeeInfo` 数据项。但是，由于此 `TreeView` 的 `SelectedValuePath` 设置为 `EmployeeNumber`，因此 `SelectedValue` 设置为 `EmployeeNumber`。

XAML

```
<TreeView ItemsSource="{Binding Source={StaticResource myEmployeeData},
    ItemTemplate={StaticResource SampleTemplate},
    XPath=EmployeeInfo}"
    Name="myTreeView"
    SelectedValuePath="EmployeeNumber"
    />

<TextBlock Margin="10">SelectedValuePath: </TextBlock>
<TextBlock Margin="10,0,0,0"
    Text="{Binding ElementName=myTreeView,
        Path=SelectedValuePath}"
    Foreground="Blue"/>

<TextBlock Margin="10">SelectedValue: </TextBlock>
<TextBlock Margin="10,0,0,0"
    Text="{Binding ElementName=myTreeView,
        Path=SelectedValue}"
    Foreground="Blue"/>
```

另请参阅

- [TreeView](#)
- [TreeViewItem](#)
- [TreeView 概述](#)
- [操作指南主题](#)

如何：将 TreeView 绑定到深度无法确定的数据

项目 • 2023/02/06

有时，你可能想要将 [TreeView](#) 绑定到深度未知的数据源。当数据本质上是递归的则可能会发生这种情况，例如文件系统（文件夹可以包含文件夹）或公司的组织结构（员工有其他员工作为其直接下属）。

数据源必须具有分层对象模型。例如，类 `Employee` 可能包含 `Employee` 对象的集合，这些对象是某位员工的直接下属。如果数据以非分层的方式表示，则必须生成数据的分层表示形式。

设置 `ItemsControl.ItemTemplate` 属性时，如果 `ItemsControl` 为每个子项生成一个 `ItemsControl`，则子项 `ItemsControl` 会使用与父项相同的 `ItemTemplate`。例如，如果在数据绑定 `TreeView` 上设置 `ItemTemplate` 属性，则生成的每个 `TreeViewItem` 都使用分配给 `TreeView` 的 `ItemTemplate` 属性的 `DataTemplate`。

使用 `HierarchicalDataTemplate` 可以在数据模板上指定 `TreeViewItem` 的 `ItemsSource`，或指定任意 `HeaderedItemsControl`。设置 `HierarchicalDataTemplate.ItemsSource` 属性时，在应用 `HierarchicalDataTemplate` 时使用该值。通过使用 `HierarchicalDataTemplate`，可以为 `TreeView` 中的每个 `TreeViewItem` 以递归方式设置 `ItemsSource`。

示例

下面的示例演示如何将 `TreeView` 绑定到分层数据，并使用 `HierarchicalDataTemplate` 指定每个 `TreeViewItem` 的 `ItemsSource`。`TreeView` 将绑定到表示公司员工的 XML 数据。每个 `Employee` 元素可以包含其他 `Employee` 元素，以指示上下级关系。由于数据是递归的，因此 `HierarchicalDataTemplate` 可以应用于每个级别。

XAML

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
<Page.Resources>
    <XmlDataProvider x:Key="myCompany" XPath="Company/Employee">
        <x:XData>
            <Company xmlns="">
                <Employee Name="Don Hall">
                    <Employee Name="Alice Ciccu">
                        <Employee Name="David Pelton">
                            <Employee Name="Vivian Atlas"/>
                        </Employee>
                    </Employee>
                </Employee>
            </Company>
        </x:XData>
    </XmlDataProvider>
</Page.Resources>
</Page>
```

```

        <Employee Name="Jeff Price"/>
        <Employee Name="Andy Jacobs"/>
    </Employee>
    <Employee Name="Bill Malone">
        <Employee Name="Maurice Taylor"/>
        <Employee Name="Sunil Uppal"/>
        <Employee Name="Qiang Wang"/>
    </Employee>
    </Employee>
</Company>
</x:XData>
</XmlDataProvider>

<!-- Bind the HierarchicalDataTemplate.ItemsSource property to the
employees under
each Employee element. --&gt;
&lt;HierarchicalDataTemplate x:Key="EmployeeTemplate"
    ItemsSource="{Binding XPath=Employee}"&gt;
    &lt;TextBlock Text="{Binding XPath=@Name}" &gt;&lt;/TextBlock&gt;
&lt;/HierarchicalDataTemplate&gt;

&lt;Style TargetType="TreeViewItem"&gt;
    &lt;Setter Property="IsExpanded" Value="True"/&gt;
&lt;/Style&gt;
&lt;/Page.Resources&gt;

&lt;Grid&gt;
    &lt;TreeView ItemsSource="{Binding Source={StaticResource myCompany}}"
        ItemTemplate="{StaticResource EmployeeTemplate}" /&gt;
&lt;/Grid&gt;
&lt;/Page&gt;
</pre>

```

另请参阅

- [数据绑定概述](#)
- [数据模板化概述](#)

如何：提高 TreeView 的性能

项目 • 2023/02/06

如果 `TreeView` 包含许多项，则加载所需的时间可能会导致用户界面出现相当长的延迟。可以通过将 `VirtualizingStackPanel.IsVirtualizing` 附加属性设置为 `true` 来缩短加载时间。当用户使用鼠标滚轮或拖动滚动条的拇指滚动 `TreeView` 时，UI 的反应也可能很慢。可以通过将 `VirtualizingStackPanel.VirtualizationMode` 附加属性设置为 `VirtualizationMode.Recycling` 来提高用户滚动时 `TreeView` 的性能。

示例

描述

以下示例创建一个 `TreeView`，它将 `VirtualizingStackPanel.IsVirtualizing` 附加属性设置为 `true`，将 `VirtualizingStackPanel.VirtualizationMode` 附加属性设置为 `VirtualizationMode.Recycling`，以便优化其性能。

代码

XAML

```
<StackPanel>
    <StackPanel.Resources>
        <src:TreeViewData x:Key="dataItems"/>

        <HierarchicalDataTemplate DataType="{x:Type src:ItemsForTreeView}"
            ItemsSource="{Binding Path=SecondLevelItems}">
            <!--Display the TopLevelName property in the first level.-->
            <TextBlock Text="{Binding Path=TopLevelName}" />

            <!--Display each string in the SecondLevelItems property in
                the second level.-->
            <HierarchicalDataTemplate.ItemTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding}" />
                </DataTemplate>
            </HierarchicalDataTemplate.ItemTemplate>

            <!--Set the foreground of the items in the second level
                to Navy.-->
            <HierarchicalDataTemplate.ItemContainerStyle>
                <Style TargetType="TreeViewItem">
```

```

        <Setter Property="Foreground" Value="Navy"/>
    </Style>
</HierarchicalDataTemplate.ItemContainerStyle>
</HierarchicalDataTemplate>
</StackPanel.Resources>

<TreeView Height="200"
    ItemsSource="{Binding Source={StaticResource dataItems}}"
    VirtualizingStackPanel.IsVirtualizing="True"
    VirtualizingStackPanel.VirtualizationMode="Recycling">
<TreeView.ItemContainerStyle>

    <!--Expand each TreeViewItem in the first level and
        set its foreground to Green.-->
<Style TargetType="TreeViewItem">
    <Setter Property="IsExpanded" Value="True"/>
    <Setter Property="Foreground" Value="Green"/>
</Style>
</TreeView.ItemContainerStyle>
</TreeView>
</StackPanel>

```

以下示例显示了上一个示例所使用的数据。

C#

```

public class TreeViewData : ObservableCollection<ItemsForTreeView>
{
    public TreeViewData()
    {
        for (int i = 0; i < 100; ++i)
        {
            ItemsForTreeView item = new ItemsForTreeView();
            item.TopLevelName = "item " + i.ToString();
            Add(item);
        }
    }
}

public class ItemsForTreeView
{
    public string TopLevelName { get; set; }
    private ObservableCollection<string> level2Items;

    public ObservableCollection<string> SecondLevelItems
    {
        get
        {
            level2Items ??= new ObservableCollection<string>();
            return level2Items;
        }
    }
}

```

```
public ItemsForTreeView()
{
    for (int i = 0; i < 10; ++i)
    {
        SecondLevelItems.Add("Second Level " + i.ToString());
    }
}
```

另请参阅

- [控件](#)

如何：在 TreeView 中查找 TreeViewItem

项目 • 2023/02/06

TreeView 控件提供了一种显示分层数据的简便方式。如果 TreeView 绑定到数据源，则可使用 SelectedItem 属性快速检索所选数据对象。通常最好使用基础数据对象，但有时可能需要以编程方式操作包含 TreeViewItem 的数据。例如，可能需要以编程方式展开 TreeViewItem，或在 TreeView 中选择不同的项。

要查找包含特定数据对象的 TreeViewItem，必须遍历 TreeView 的每个级别。也可虚拟化 TreeView 中的项以提高性能。如果项可能已被虚拟化，还必须实现 TreeViewItem 以检查它是否包含数据对象。

示例

描述

下面的示例在 TreeView 中搜索特定对象并返回包含该对象的 TreeViewItem。该示例确保实例化每个 TreeViewItem，以便可以搜索其子项。如果 TreeView 不使用虚拟化项目，此示例也适用。

① 备注

下面的示例适用于任何 TreeView（无论基础数据模型如何），并搜索每个 TreeViewItem，直到找到该对象。另一种性能更好的方法是在数据模型中搜索指定对象，跟踪其在数据层次结构中的位置，然后在 TreeView 中找到对应的 TreeViewItem。但是，具有良好性能的方法需要了解数据模型，并且无法在任何给定的 TreeView 中实现通用化。

代码

C#

```
/// <summary>
/// Recursively search for an item in this subtree.
/// </summary>
/// <param name="container">
///   The parent ItemsControl. This can be a TreeView or a TreeViewItem.
/// </param>
/// <param name="item">
///   The item to search for.
```

```

/// </param>
/// <returns>
/// The TreeViewItem that contains the specified item.
/// </returns>
private TreeViewItem GetTreeViewItem(ItemsControl container, object item)
{
    if (container != null)
    {
        if (container.DataContext == item)
        {
            return container as TreeViewItem;
        }

        // Expand the current container
        if (container is TreeViewItem && !((TreeViewItem)container).IsExpanded)
        {
            container.SetValue(TreeViewItem.IsExpandedProperty, true);
        }

        // Try to generate the ItemsPresenter and the ItemsPanel.
        // by calling ApplyTemplate. Note that in the
        // virtualizing case even if the item is marked
        // expanded we still need to do this step in order to
        // regenerate the visuals because they may have been virtualized
        // away.

        container.ApplyTemplate();
        ItemsPresenter itemsPresenter =
            (ItemsPresenter)container.Template.FindName("ItemsHost",
        container);
        if (itemsPresenter != null)
        {
            itemsPresenter.ApplyTemplate();
        }
        else
        {
            // The Tree template has not named the ItemsPresenter,
            // so walk the descendants and find the child.
            itemsPresenter = FindVisualChild<ItemsPresenter>(container);
            if (itemsPresenter == null)
            {
                container.UpdateLayout();

                itemsPresenter = FindVisualChild<ItemsPresenter>(container);
            }
        }
    }

    Panel itemsHostPanel =
    (Panel)VisualTreeHelper.GetChild(itemsPresenter, 0);

    // Ensure that the generator for this panel has been created.
    UIElementCollection children = itemsHostPanel.Children;

    MyVirtualizingStackPanel virtualizingPanel =

```

```

        itemsHostPanel as MyVirtualizingStackPanel;

    for (int i = 0, count = container.Items.Count; i < count; i++)
    {
        TreeViewItem subContainer;
        if (virtualizingPanel != null)
        {
            // Bring the item into view so
            // that the container will be generated.
            virtualizingPanel.BringIntoView(i);

            subContainer =
                (TreeViewItem)container.ItemContainerGenerator.
                ContainerFromIndex(i);
        }
        else
        {
            subContainer =
                (TreeViewItem)container.ItemContainerGenerator.
                ContainerFromIndex(i);

            // Bring the item into view to maintain the
            // same behavior as with a virtualizing panel.
            subContainer.BringIntoView();
        }

        if (subContainer != null)
        {
            // Search the next level for the object.
            TreeViewItem resultContainer = GetTreeViewItem(subContainer,
item);
            if (resultContainer != null)
            {
                return resultContainer;
            }
            else
            {
                // The object is not under this TreeViewItem
                // so collapse it.
                subContainer.IsExpanded = false;
            }
        }
    }

    return null;
}

/// <summary>
/// Search for an element of a certain type in the visual tree.
/// </summary>
/// <typeparam name="T">The type of element to find.</typeparam>
/// <param name="visual">The parent element.</param>
/// <returns></returns>
private T FindVisualChild<T>(Visual visual) where T : Visual

```

```

{
    for (int i = 0; i < VisualTreeHelper.GetChildrenCount(visual); i++)
    {
        Visual child = (Visual)VisualTreeHelper.GetChild(visual, i);
        if (child != null)
        {
            T correctlyTyped = child as T;
            if (correctlyTyped != null)
            {
                return correctlyTyped;
            }

            T descendant = FindVisualChild<T>(child);
            if (descendant != null)
            {
                return descendant;
            }
        }
    }

    return null;
}

```

前面的代码依赖于公开名为 `BringIntoView` 的方法的自定义 `VirtualizingStackPanel`。下面的代码定义自定义 `VirtualizingStackPanel`。

```

C#

public class MyVirtualizingStackPanel : VirtualizingStackPanel
{
    /// <summary>
    /// Publically expose BringIndexIntoView.
    /// </summary>
    public void BringIntoView(int index)
    {

        this.BringIndexIntoView(index);
    }
}

```

以下 XAML 展示了如何创建使用自定义 `VirtualizingStackPanel` 的 `TreeView`。

XAML

```

<TreeView VirtualizingStackPanel.IsVirtualizing="True">

<!--Use the custom class MyVirtualizingStackPanel
     as the ItemsPanel for the TreeView and
     TreeViewItem object.-->
<TreeView.ItemsPanel>
    <ItemsPanelTemplate>

```

```
<src:MyVirtualizingStackPanel/>
</ItemsPanelTemplate>
</TreeView.ItemsPanel>
<TreeView.ItemContainerStyle>
<Style TargetType="TreeViewItem">
<Setter Property="ItemsPanel">
<Setter.Value>
<ItemsPanelTemplate>
<src:MyVirtualizingStackPanel/>
</ItemsPanelTemplate>
</Setter.Value>
</Setter>
</Style>
</TreeView.ItemContainerStyle>
</TreeView>
```

另请参阅

- 提升 TreeView 的性能

WrapPanel

项目 • 2023/02/06

[WrapPanel](#) 元素将子元素按从左到右的顺序定位，将内容分到其包含框边缘的下一行。

参考

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

相关章节

[布局](#)

[演练：我的第一个 WPF 桌面应用程序](#)

[ScrollViewer 概述](#)

Viewbox

项目 • 2023/02/06

[Viewbox](#) 控件用于拉伸或缩放子元素。

本节内容

[向 Viewbox 的内容应用 Stretch 属性](#)

参考

[Viewbox](#)

[Image](#)

请参阅

- [WPF 控件库示例 ↗](#)

如何：向 Viewbox 的内容应用 Stretch 属性

项目 • 2023/02/06

示例

此示例演示如何更改 [Viewbox](#) 的 [StretchDirection](#) 和 [Stretch](#) 属性的值。

第一个示例使用 Extensible Application Markup Language (XAML) 定义 [Viewbox](#) 元素。它分配了 400 的 [MaxWidth](#) 和 [MaxHeight](#)。该示例将 [Image](#) 元素嵌套在 [Viewbox](#) 中。与 [Stretch](#) 和 [StretchDirection](#) 枚举的属性值对应的 [Button](#) 元素操作嵌套的 [Image](#) 的拉伸行为。

XAML

```
<StackPanel Margin="0,0,0,10" HorizontalAlignment="Center"
Orientation="Horizontal" DockPanel.Dock="Top">
    <Button Name="btn1" Click="stretchNone">Stretch="None" </Button>
    <Button Name="btn2" Click="stretchFill">Stretch="Fill" </Button>
    <Button Name="btn3" Click="stretchUni">Stretch="Uniform" </Button>
    <Button Name="btn4" Click="stretchUniFill">Stretch="UniformToFill"
</Button>
</StackPanel>

<StackPanel Margin="0,0,0,10" HorizontalAlignment="Center"
Orientation="Horizontal" DockPanel.Dock="Top">
    <Button Name="btn5" Click="sdUpOnly">StretchDirection="UpOnly" </Button>
    <Button Name="btn6" Click="sdDownOnly">StretchDirection="DownOnly"
</Button>
    <Button Name="btn7" Click="sdBoth">StretchDirection="Both" </Button>
</StackPanel>

<TextBlock DockPanel.Dock="Top" Name="txt1" />
<TextBlock DockPanel.Dock="Top" Name="txt2" />

<StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
    <Viewbox MaxWidth="500" MaxHeight="500" Name="vb1">
        <Image Source="tulip_farm.jpg"/>
    </Viewbox>
</StackPanel>
```

下面的代码隐藏文件处理上一个 XAML 示例定义的 [ButtonClick](#) 事件。

C#

```

private void stretchNone(object sender, RoutedEventArgs e)
{
    vb1.Stretch = System.Windows.Media.Stretch.None;
    txt1.Text = "Stretch is now set to None.";
}

private void stretchFill(object sender, RoutedEventArgs e)
{
    vb1.Stretch = System.Windows.Media.Stretch.Fill;
    txt1.Text = "Stretch is now set to Fill.";
}

private void stretchUni(object sender, RoutedEventArgs e)
{
    vb1.Stretch = System.Windows.Media.Stretch.Uniform;
    txt1.Text = "Stretch is now set to Uniform.";
}

private void stretchUniFill(object sender, RoutedEventArgs e)
{
    vb1.Stretch = System.Windows.Media.Stretch.UniformToFill;
    txt1.Text = "Stretch is now set to UniformToFill.";
}

private void sdUpOnly(object sender, RoutedEventArgs e)
{
    vb1.StretchDirection = System.Windows.Controls.StretchDirection.UpOnly;
    txt2.Text = "StretchDirection is now UpOnly.";
}

private void sdDownOnly(object sender, RoutedEventArgs e)
{
    vb1.StretchDirection =
System.Windows.Controls.StretchDirection.DownOnly;
    txt2.Text = "StretchDirection is now DownOnly.";
}

private void sdBoth(object sender, RoutedEventArgs e)
{
    vb1.StretchDirection = System.Windows.Controls.StretchDirection.Both;
    txt2.Text = "StretchDirection is now Both.";
}

```

另请参阅

- [Viewbox](#)
- [Stretch](#)
- [StretchDirection](#)

样式和模板

项目 • 2023/02/06

Windows Presentation Foundation (WPF) 样式设置和模板化是指一套功能（样式、模板、触发器和情节提要），可使应用程序、文档和用户界面 (UI) 设计者创建极具视觉表现力的应用程序，并为其产品创建标准化的特定外观。

本节内容

[样式设置和模板化](#)

[如何：查找由 ControlTemplate 生成的元素](#)

参考

[Style](#)

[ControlTemplate](#)

[DataTemplate](#)

相关章节

[高级](#)

[控件自定义](#)

[图形和多媒体](#)

WPF 中的样式和模板

项目 • 2023/03/04

Windows Presentation Foundation (WPF) 样式设置和模板化是指一套功能，这套功能使开发者和设计者能够为其产品创建极具视觉表现力的效果和一致的外观。自定义应用的外观时，需要一个强大的样式设置和模板化模型，以便维护和共享应用内部和应用之间的外观。WPF 就提供了这样的模型。

WPF 样式设置模型的另一项功能是将呈现与逻辑分离。设计者可以仅使用 XAML 处理应用外观，与此同时开发者使用 C# 或 Visual Basic 处理编程逻辑。

本概述侧重于应用的样式设置和模板化两方面，不讨论任何数据绑定概念。有关数据绑定的信息，请参阅[数据绑定概述](#)。

了解资源很重要，正是这些资源使样式和模板能够重复使用。有关资源的详细信息，请参阅[XAML 资源](#)。

示例

本概述中提供的示例代码基于下图所示的[简单照片浏览应用程序](#)。



此简单照片示例使用样式设置和模板化创建极具视觉表现力的用户体验。该示例具有两个 [TextBlock](#) 元素和一个绑定到图像列表的 [ListBox](#) 控件。

有关完整示例，请参阅[样式设置和模板化示例简介](#)。

样式

可以将 [Style](#) 视为一种将一组属性值应用到多个元素的便捷方法。可以对从 [FrameworkElement](#) 或 [FrameworkContentElement](#) 派生的任何元素（如 [Window](#) 或 [Button](#)）使用样式。

声明样式的最常见方法是在 XAML 文件的 `Resources` 部分中声明为资源。样式是一种资源，因此它们遵从适用于所有资源的相同范围规则。简而言之，声明样式的位置会影响样式的应用范围。例如，如果在应用定义 XAML 文件的根元素中声明样式，则该样式可以在应用中的任何位置使用。

例如，以下 XAML 代码为 `TextBlock` 声明了两个样式，一个自动应用于所有 `TextBlock` 元素，另一个则必须显式引用。

XAML

```
<Window.Resources>
    <!-- .... other resources .... -->

    <!--A Style that affects all TextBlocks-->
    <Style TargetType="TextBlock">
        <Setter Property="HorizontalAlignment" Value="Center" />
        <Setter Property="FontFamily" Value="Comic Sans MS"/>
        <Setter Property="FontSize" Value="14"/>
    </Style>

    <!--A Style that extends the previous TextBlock Style with an x:Key of
    TitleText-->
    <Style BasedOn="{StaticResource {x:Type TextBlock}}"
        TargetType="TextBlock"
        x:Key="TitleText">
        <Setter Property="FontSize" Value="26"/>
        <Setter Property="Foreground">
            <Setter.Value>
                <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
                    <LinearGradientBrush.GradientStops>
                        <GradientStop Offset="0.0" Color="#90DDDD" />
                        <GradientStop Offset="1.0" Color="#5BFFFF" />
                    </LinearGradientBrush.GradientStops>
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
    </Style>
</Window.Resources>
```

下面举例说明了如何使用上面声明的样式。

XAML

```
<StackPanel>
    <TextBlock Style="{StaticResource TitleText}" Name="textblock1">My
    Pictures</TextBlock>
    <TextBlock>Check out my new pictures!</TextBlock>
</StackPanel>
```



ControlTemplate

在 WPF 中，控件的 [ControlTemplate](#) 用于定义控件的外观。可以通过定义新的 [ControlTemplate](#) 并将其分配给控件来更改控件的结构和外观。在许多情况下，模板提供了足够的灵活性，从而无需自行编写自定义控件。

每个控件都有一个分配给 [Control.Template](#) 属性的默认模板。该模板将控件的视觉呈现与控件的功能关联起来。因为在 XAML 中定义了模板，所以无需编写任何代码即可更改控件的外观。每个模板都是为特定控件（例如 [Button](#)）设计的。

通常在 XAML 文件的 [Resources](#) 部分中将模板声明为资源。与所有资源一样，将应用范围规则。

控件模板比样式复杂得多。这是因为控件模板重写了整个控件的视觉外观，而样式只是将属性更改应用于现有控件。但是，控件模板是通过设置 [Control.Template](#) 属性来应用的，因此可以使用样式来定义或设置模板。

设计器通常允许创建现有模板的副本并进行修改。例如，在 Visual Studio WPF 设计器中，选择一个 [CheckBox](#) 控件，然后右键单击并选择“编辑模板”>“创建副本”。此命令会生成一个用于定义模板的样式。

```
XAML

<Style x:Key="CheckBoxStyle1" TargetType="{x:Type CheckBox}">
    <Setter Property="FocusVisualStyle" Value="{StaticResource FocusVisual1}" />
    <Setter Property="Background" Value="{StaticResource OptionMark.Static.Background1}" />
    <Setter Property="BorderBrush" Value="{StaticResource OptionMark.Static.Border1}" />
    <Setter Property="Foreground" Value="{DynamicResource {x:Static SystemColors.ControlTextBrushKey}}" />
    <Setter Property="BorderThickness" Value="1" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type CheckBox}">
                <Grid x:Name="templateRoot" Background="Transparent" SnapsToDevicePixels="True">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="Auto" />
                        <ColumnDefinition Width="*" />
                
```

```

        </Grid.ColumnDefinitions>
        <Border x:Name="checkBoxBorder" Background="{TemplateBinding Background}" BorderThickness="{TemplateBinding BorderThickness}" BorderBrush="{TemplateBinding BorderBrush}" HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}" Margin="1" VerticalAlignment="{TemplateBinding VerticalContentAlignment}">
            <Grid x:Name="markGrid">
                <Path x:Name="optionMark" Data="F1 M 9.97498,1.22334L 4.6983,9.09834L 4.52164,9.09834L 0,5.19331L 1.27664,3.52165L 4.255,6.08833L 8.33331,1.52588e-005L 9.97498,1.22334 Z " Fill="{StaticResource OptionMark.Static.Glyph1}" Margin="1" Opacity="0" Stretch="None"/>
                <Rectangle x:Name="indeterminateMark" Fill="{StaticResource OptionMark.Static.Glyph1}" Margin="2" Opacity="0"/>
            </Grid>
        </Border>
        <ContentPresenter x:Name="contentPresenter" Grid.Column="1" Focusable="False" HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}" Margin="{TemplateBinding Padding}" RecognizesAccessKey="True" SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}" VerticalAlignment="{TemplateBinding VerticalContentAlignment}"/>
    </Grid>
    <ControlTemplate.Triggers>
        <Trigger Property="HasContent" Value="true">
            <Setter Property="FocusVisualStyle" Value="{StaticResource OptionMarkFocusVisualStyle}"/>
            <Setter Property="Padding" Value="4,-1,0,0"/>
        </Trigger>
    ... content removed to save space ...

```

通过编辑模板副本可以很好地了解模板的工作原理。与其新建一个空白模板，不如编辑副本并更改视觉呈现的某些方面来得简单。

有关示例，请参阅[为控件创建模板](#)。

TemplateBinding

你可能已经注意到，上一部分中定义的模板资源使用了 [TemplateBinding 标记扩展](#)。对于模板方案来说，`TemplateBinding` 是绑定的优化形式，类似于使用 `{Binding RelativeSource={RelativeSource TemplatedParent}}` 构造的绑定。`TemplateBinding` 可用于将模板的各个部分绑定到控件的各个属性。例如，每个控件都有一个 `BorderThickness` 属性。可使用 `TemplateBinding` 管理此控件设置影响模板中的哪个元素。

ContentControl 和 ItemsControl

如果在 `ContentControl` 的 `ControlTemplate` 中声明了 `ContentPresenter`，`ContentPresenter` 将自动绑定到 `ContentTemplate` 和 `Content` 属性。同样，

ItemsControl 的 ControlTemplate 中的 ItemsPresenter 将自动绑定到 ItemTemplate 和 Items 属性。

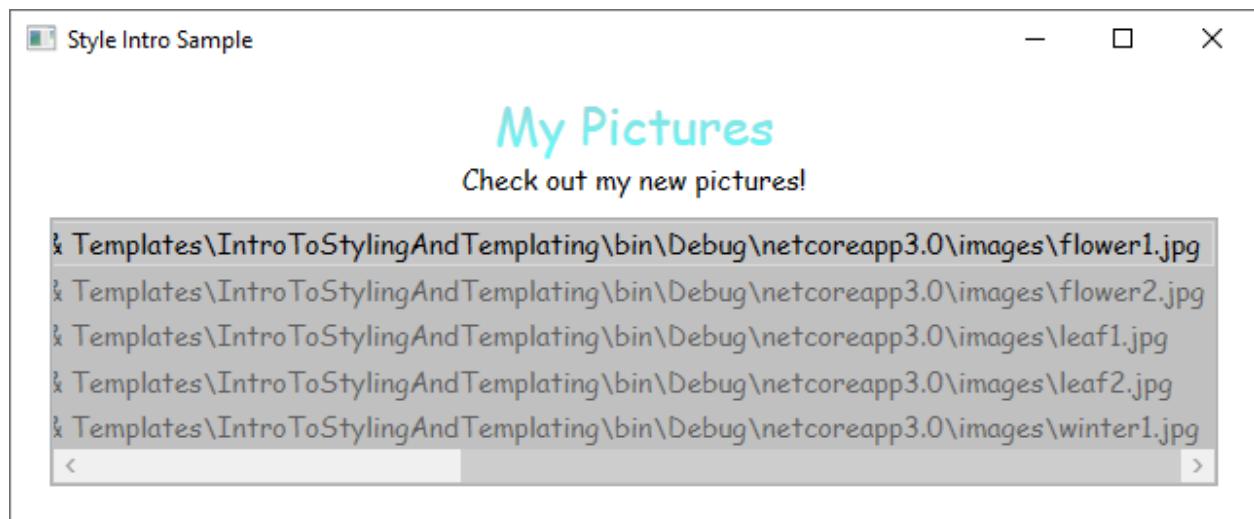
DataTemplate

在此示例应用中，有一个绑定到照片列表的 ListBox 控件。

XAML

```
<ListBox ItemsSource="{Binding Source={StaticResource MyPhotos}}"
         Background="Silver" Width="600" Margin="10" SelectedIndex="0"/>
```

此 ListBox 当前如下所示。



大多数控件都具有某些类型的内容，并且该内容通常来自要绑定到的数据。在此示例中，数据是照片列表。在 WPF 中，使用 DataTemplate 定义数据的视觉表示形式。基本上，放入 DataTemplate 的内容决定了数据在呈现的应用中的外观。

在示例应用中，每个自定义 Photo 对象都有一个字符串类型的 Source 属性，用于指定图像的文件路径。当前，照片对象显示为文件路径。

C#

```
public class Photo
{
    public Photo(string path)
    {
        Source = path;
    }

    public string Source { get; }

    public override string ToString() => Source;
}
```

若要使照片显示为图像，请将 [DataTemplate](#) 创建为资源。

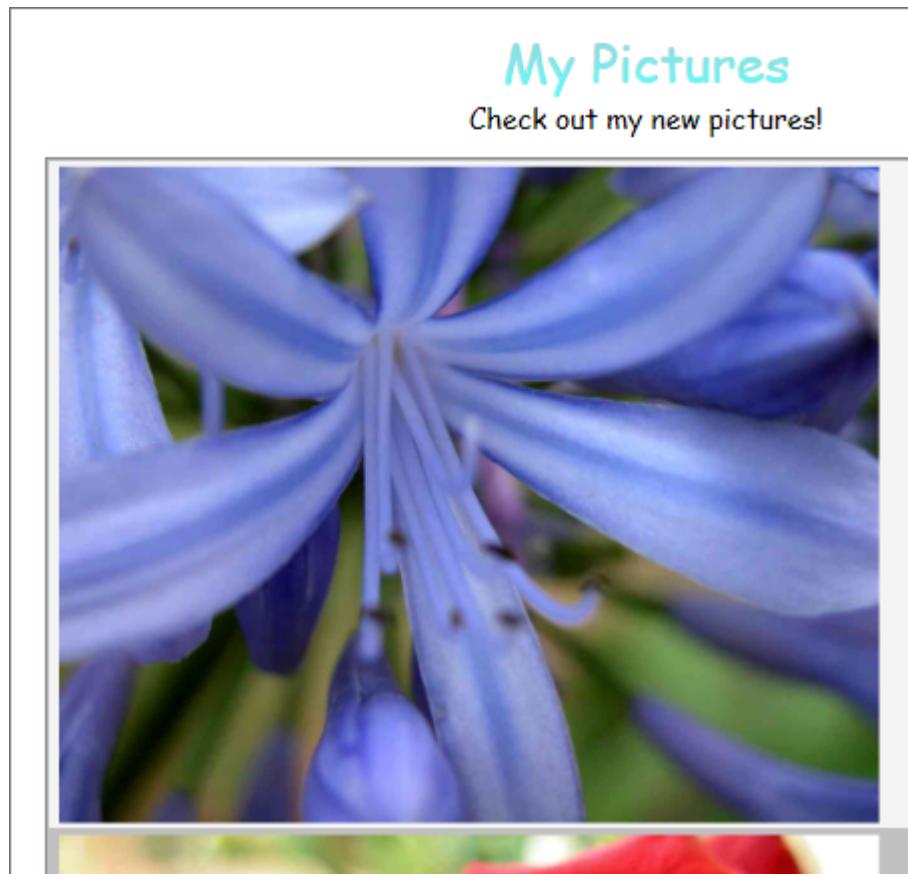
XAML

```
<Window.Resources>
    <!-- .... other resources .... -->

    <!--DataTemplate to display Photos as images
    instead of text strings of Paths-->
    <DataTemplate DataType="{x:Type local:Photo}">
        <Border Margin="3">
            <Image Source="{Binding Source}" />
        </Border>
    </DataTemplate>
</Window.Resources>
```

请注意，[DataType](#) 属性与 [Style](#) 的 [TargetType](#) 属性相似。如果 [DataTemplate](#) 在 [resources](#) 部分中，则在将 [DataType](#) 属性指定为某个类型并且省略 [x:Key](#) 时，只要该类型出现，就会应用 [DataTemplate](#)。始终可以选择为 [DataTemplate](#) 分配 [x:Key](#)，然后将其设置为采用 [DataTemplate](#) 类型的属性（例如 [ItemTemplate](#) 属性或 [ContentTemplate](#) 属性）的 [StaticResource](#)。

实质上，上述示例中的 [DataTemplate](#) 定义每当存在 [Photo](#) 对象时，它都应显示为 [Border](#) 内的 [Image](#)。有了这个 [DataTemplate](#)，我们的应用现在如下所示。



数据模板化模型提供其他功能。例如，如果要使用 `HeaderedItemsControl` 类型（例如 `Menu` 或 `TreeView`）显示包含其他集合的集合数据，可以使用 `HierarchicalDataTemplate`。另一个数据模板化功能是 `DataTemplateSelector`，该功能允许基于自定义逻辑选择要使用的 `DataTemplate`。有关详细信息，请参阅[数据模板化概述](#)，该概述提供不同数据模板化功能的更多深入讨论。

触发器

触发器在属性值发生更改或引发事件时设置属性或启动操作，例如动画。`Style`、`ControlTemplate` 和 `DataTemplate` 都具有可包含一组触发器的 `Triggers` 属性。触发器分为几种类型。

PropertyTrigger

根据属性的值设置属性值或启动操作的 `Trigger` 称为属性触发器。

若要演示如何使用属性触发器，可以使每个 `ListBoxItem` 在未选中时部分透明。以下样式将 `ListBoxItem` 的 `Opacity` 值设置为 `0.5`。但是，当 `IsSelected` 属性为 `true` 时，`Opacity` 设置为 `1.0`。

XAML

```
<Window.Resources>
    <!-- .... other resources .... -->

    <Style TargetType="ListBoxItem">
        <Setter Property="Opacity" Value="0.5" />
        <Setter Property="MaxHeight" Value="75" />
        <Style.Triggers>
            <Trigger Property="IsSelected" Value="True">
                <Trigger.Setters>
                    <Setter Property="Opacity" Value="1.0" />
                </Trigger.Setters>
            </Trigger>
        </Style.Triggers>
    </Style>
</Window.Resources>
```

此示例使用 `Trigger` 设置属性值，但请注意，`Trigger` 类还有 `EnterActions` 和 `ExitActions` 属性，这些属性可使触发器执行操作。

请注意，`ListBoxItem` 的 `MaxHeight` 属性设置为 `75`。在下图中，第三项是选中的项。



EventTrigger 和情节提要

另一个触发器类型是 [EventTrigger](#)，用于根据某个事件的发生启动一组操作。例如，以下 [EventTrigger](#) 对象指定当鼠标指针进入 [ListBoxItem](#) 时，[MaxHeight](#) 属性在 [0.2](#) 秒的时间内动画化为值 [90](#)。当鼠标离开该项时，属性在 [1](#) 秒的时间内返回到原始值。请注意，不需要为 [MouseLeave](#) 动画指定 [To](#) 值。这是因为动画能够跟踪原始值。

XAML

```
<Style.Triggers>
    <Trigger Property="IsSelected" Value="True">
        <Trigger.Setters>
            <Setter Property="Opacity" Value="1.0" />
        </Trigger.Setters>
    </Trigger>
    <EventTrigger RoutedEvent="Mouse.MouseEnter">
        <EventTrigger.Actions>
            <BeginStoryboard>
                <Storyboard>
                    <DoubleAnimation
                        Duration="0:0:0.2"
                        Storyboard.TargetProperty="MaxHeight"
                        To="90" />
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger.Actions>
    </EventTrigger>
    <EventTrigger RoutedEvent="Mouse.MouseLeave">
        <EventTrigger.Actions>
            <BeginStoryboard>
                <Storyboard>
                    <DoubleAnimation
                        Duration="0:0:1"
                        Storyboard.TargetProperty="MaxHeight" />
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger.Actions>
    </EventTrigger>
</Style.Triggers>
```

有关详细信息，请参阅[情节提要概述](#)。

在下图中，鼠标指向第三项。



MultiTrigger、DataTrigger 和 MultiDataTrigger

除了 [Trigger](#) 和 [EventTrigger](#)，还有其他类型的触发器。[MultiTrigger](#) 允许基于多个条件设置属性值。当条件的属性为数据绑定时，使用 [DataTrigger](#) 和 [MultiDataTrigger](#)。

视觉状态

控件始终处于特定的状态。例如，当鼠标在控件的表面上移动时，该控件被视为处于公用状态 `MouseOver`。没有特定状态的控件被视为处于公用 `Normal` 状态。状态分为多个组，前面提到的状态属于 `CommonStates` 状态组。大多数控件都有两个状态组：

`CommonStates` 和 `FocusStates`。在应用于控件的每个状态组中，控件始终处于每个组的一种状态，例如 `CommonStates.MouseOver` 和 `FocusStates.Unfocused`。但是，控件不能处于同一组中的两种不同状态，例如 `CommonStates.Normal` 和 `CommonStates.Disabled`。下面是大多数控件可以识别和使用的状态表。

VisualStyle 名称	VisualStyleGroup 名称	描述
普通	<code>CommonStates</code>	默认状态。
<code>MouseOver</code>	<code>CommonStates</code>	鼠标指针悬停在控件上方。
已按下	<code>CommonStates</code>	已按下控件。
已禁用	<code>CommonStates</code>	已禁用控件。
已设定焦点	<code>FocusStates</code>	控件有焦点。
失去焦点	<code>FocusStates</code>	控件没有焦点。

通过在控件模板的根元素上定义 `System.Windows.VisualStateManager`，可以在控件进入特定状态时触发动画。`VisualStateManager` 声明要监视的 `VisualStateGroup` 和 `VisualState` 的组合。当控件进入受监视状态时，将启动 `VisualStateManager` 定义的动画。

例如，以下 XAML 代码监视 `CommonStates.MouseOver` 状态，以对名为 `backgroundElement` 的元素的填充颜色进行动画处理。当控件恢复为 `CommonStates.Normal` 状态时，将还原名为 `backgroundElement` 的元素的填充颜色。

XAML

```
<ControlTemplate x:Key="roundbutton" TargetType="Button">
    <Grid>
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup Name="CommonStates">
                <VisualState Name="Normal">
                    <ColorAnimation
                        Storyboard.TargetName="backgroundElement"
                        Storyboard.TargetProperty="(Shape.Fill).(
                            SolidColorBrush.Color)"
                        To="{TemplateBinding Background}"
                        Duration="0:0:0.3"/>
                </VisualState>
                <VisualState Name="MouseOver">
                    <ColorAnimation
                        Storyboard.TargetName="backgroundElement"
                        Storyboard.TargetProperty="(Shape.Fill).(
                            SolidColorBrush.Color)"
                        To="Yellow"
                        Duration="0:0:0.3"/>
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
    ...

```

有关情节提要的详细信息，请参阅[情节提要概述](#)。

共享资源和主题

典型的 WPF 应用可能具有多个 UI 资源，它们可应用于整个应用。这组资源统称为应用的主题。WPF 支持使用封装为 `ResourceDictionary` 类的资源字典将 UI 资源打包为主题。

WPF 主题通过使用 WPF 公开的样式设置和模板化机制来定义，该机制用于自定义任何元素的视觉对象。

WPF 主题资源存储在嵌入的资源字典中。这些资源必须嵌入到已签名的程序集内，并且可以嵌入到与代码本身相同的程序集内或并行程序集内。对于包含 WPF 控件的程序集 PresentationFramework.dll，主题资源位于一系列并行程序集内。

该主题成为在搜索元素样式时最后查看的位置。通常，搜索先沿着元素树搜索适当的资源，然后在应用资源集合中查找，最后查询系统。这样一来，应用开发者便有机会在到达主题之前在树或应用级别上为任何对象重新定义样式。

可以将资源字典定义为单独的文件，这些文件支持跨多个应用重复使用主题。还可以通过定义多个资源字典来创建可交换的主题，这些资源字典以不同的值提供相同类型的资源。在应用级别上重新定义这些样式或其他资源是设计应用外观的推荐方法。

若要跨应用共享一组资源（包括样式和模板），可创建 XAML 文件，并定义包含对 `shared.xaml` 文件的引用的 [ResourceDictionary](#)。

XAML

```
<ResourceDictionary.MergedDictionaries>
  <ResourceDictionary Source="Shared.xaml" />
</ResourceDictionary.MergedDictionaries>
```

它是 `shared.xaml` 的共享，用于定义包含一组样式和画笔资源的 [ResourceDictionary](#)，从而使应用中的控件具有一致的外观。

有关详细信息，请参阅[合并资源字典](#)。

如果要为自定义控件创建主题，请参阅[控件创作概述](#)的**在主题级别定义资源**部分。

请参阅

- [WPF 中的 Pack URI](#)
- [如何：查找由 ControlTemplate 生成的元素](#)
- [查找由 DataTemplate 生成的元素](#)

创建控件模板

项目 · 2022/09/27

使用 Windows Presentation Foundation (WPF) , 可以使用自己的可重用模板自定义现有控件的可视结构和行为。可以对应用程序、窗口和页面全局应用模板，也可以将模板直接应用于控件。需要新建控件的大多数场景均可改为为现有控件创建新模板。

本文将介绍如何为 [Button 控件](#) 创建新的 [ControlTemplate](#)。

何时创建 ControlTemplate

控件有许多属性，例如 [Background](#)、[Foreground](#) 和 [FontFamily](#)。这些属性控制控件外观的不同方面，但可通过设置这些属性进行的更改有限。例如，可以从 [CheckBox](#) 中将 [Foreground](#) 属性设置为蓝色，并将 [FontStyle](#) 设置为斜体。要自定义设置控件中其他属性无法实现的控件外观时，则创建 [ControlTemplate](#)。

在多数用户界面中，按钮的总体外观相同：即一个包含某些文本的矩形。若想要创建一个圆形的按钮，可以创建一个继承自该按钮或重新创建该按钮功能的新控件。此外，新用户控件还会提供圆形视觉对象。

通过自定义现有控件的可视布局，可以避免创建新控件。借助圆形按钮，可创建具有所需可视布局的 [ControlTemplate](#)。

另一方面，如果你需要具有新功能、其他属性和新设置的控件，可创建新的 [UserControl](#)。

先决条件

创建新的 WPF 应用程序，在 `MainWindow.xaml`（或选择的其他窗口）的 `<Window>` 元素中设置以下属性：

属性	值
Title	Template Intro Sample
SizeToContent	WidthAndHeight
MinWidth	250

将 `<Window>` 元素的内容设置为以下 XAML：

XAML

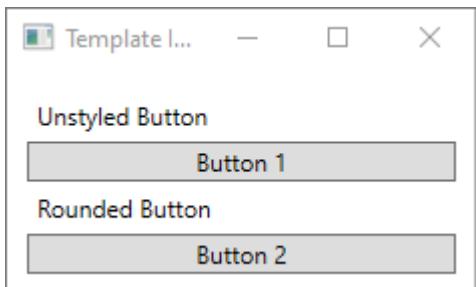
```
<StackPanel Margin="10">
    <Label>Unstyled Button</Label>
    <Button>Button 1</Button>
    <Label>Rounded Button</Label>
    <Button>Button 2</Button>
</StackPanel>
```

最后，MainWindow.xaml 文件应如下所示：

XAML

```
<Window x:Class="IntroToStylingAndTemplating.Window1"
        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:local="clr-namespace:IntroToStylingAndTemplating"
        mc:Ignorable="d"
        Title="Template Intro Sample" SizeToContent="WidthAndHeight"
        MinWidth="250">
    <StackPanel Margin="10">
        <Label>Unstyled Button</Label>
        <Button>Button 1</Button>
        <Label>Rounded Button</Label>
        <Button>Button 2</Button>
    </StackPanel>
</Window>
```

如果你运行应用程序，它将如下所示：



创建 ControlTemplate

声明 [ControlTemplate](#) 的最常见方法是在 XAML 文件的 `Resources` 部分中声明为资源。模板是资源，因此它们遵从适用于所有资源的相同范围规则。简言之，声明模板的位置会影响模板的应用范围。例如，如果在应用程序定义 XAML 文件的根元素中声明模板，则该模板可以在应用程序中的任何位置使用。如果在窗口中定义模板，则仅该窗口中的控件可以使用该模板。

首先，将 `Window.Resources` 元素添加到 MainWindow.xaml 文件：

XAML

```
<Window x:Class="IntroToStylingAndTemplating.Window2"
    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:local="clr-namespace:IntroToStylingAndTemplating"
    mc:Ignorable="d"
    Title="Template Intro Sample" SizeToContent="WidthAndHeight"
    MinWidth="250">
    <Window.Resources>

    </Window.Resources>
    <StackPanel Margin="10">
        <Label>Unstyled Button</Label>
        <Button>Button 1</Button>
        <Label>Rounded Button</Label>
        <Button>Button 2</Button>
    </StackPanel>
</Window>
```

使用以下属性集创建新的 `<ControlTemplate>` :

属性	值
x:Key	roundbutton
TargetType	Button

此控制模板很简单：

- 控件的根元素 Grid
- 用于绘制按钮圆形外观的 Ellipse
- 用于显示用户指定的按钮内容的 ContentPresenter

XAML

```
<ControlTemplate x:Key="roundbutton" TargetType="Button">
    <Grid>
        <Ellipse Fill="{TemplateBinding Background}" Stroke="
{TemplateBinding Foreground}" />
        <ContentPresenter HorizontalAlignment="Center"
VerticalAlignment="Center" />
    </Grid>
</ControlTemplate>
```

TemplateBinding

创建新的 [ControlTemplate](#) 时，可能仍然想要使用公共属性更改控件外观。[TemplateBinding](#) 标记扩展将 [ControlTemplate](#) 中元素的属性绑定到由控件定义的公共属性。 使用 [TemplateBinding](#) 时，可让控件属性用作模板参数。 换言之，设置控件属性后，该值将传递到包含 [TemplateBinding](#) 的元素。

椭圆形

请注意，[<Ellipse>](#) 元素的 [Fill](#) 和 [Stroke](#) 属性绑定到了控件的 [Foreground](#) 和 [Background](#) 属性。

ContentPresenter

此外，还将 [<ContentPresenter>](#) 元素添加到了模板。此模板专为按钮设计，因此请注意该按钮继承自 [ContentControl](#)。此按钮会显示该元素的内容。可以在该按钮中设置任何内容，例如纯文本，甚至其他控件。以下两个按钮均有效：

XAML

```
<Button>My Text</Button>

<!-- and -->

<Button>
    <CheckBox>Checkbox in a button</CheckBox>
</Button>
```

在前面的两个示例中，将文本和复选框设置为 [Button.Content](#) 属性。设置为内容的任何内容都可通过 [<ContentPresenter>](#) 显示，这是模板的功能。

若将 [ControlTemplate](#) 应用到 [ContentControl](#) 类型（例如 [Button](#)），将在元素树中搜索 [ContentPresenter](#)。若找到了 [ContentPresenter](#)，模板会自动将控件的 [Content](#) 属性绑定到 [ContentPresenter](#)。

使用模板

找到本文开头声明的按钮。

XAML

```
<StackPanel Margin="10">
    <Label>Unstyled Button</Label>
    <Button>Button 1</Button>
    <Label>Rounded Button</Label>
```

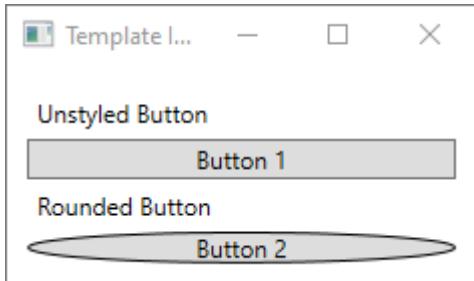
```
<Button>Button 2</Button>
</StackPanel>
```

将第二个按钮的 `Template` 属性设置为 `roundbutton` 资源：

XAML

```
<StackPanel Margin="10">
    <Label>Unstyled Button</Label>
    <Button>Button 1</Button>
    <Label>Rounded Button</Label>
    <Button Template="{StaticResource roundbutton}">Button 2</Button>
</StackPanel>
```

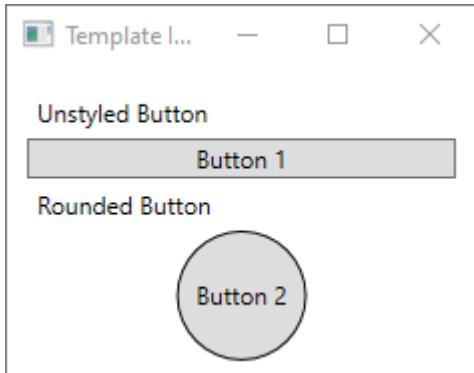
若运行项目并查看结果，将看到此按钮具有圆形背景。



你可能已注意到，此按钮不是一个圆形，而是倾斜的。由于 `<Ellipse>` 元素的工作方式，它始终会扩展并填充可用空间。将此按钮的 `width` 和 `height` 属性更改为同一个值，以使圆形均衡：

XAML

```
<StackPanel Margin="10">
    <Label>Unstyled Button</Label>
    <Button>Button 1</Button>
    <Label>Rounded Button</Label>
    <Button Template="{StaticResource roundbutton}" Width="65"
           Height="65">Button 2</Button>
</StackPanel>
```



添加触发器

即使已应用模板的按钮看上去与众不同，但它的行为与任何其他按钮相同。若按下此按钮，将触发 [Click](#) 事件。不过，你可能已注意到，当你将鼠标移到此按钮上方时，此按钮的视觉对象不会改变。这些视觉对象交互均由模板定义。

通过 WPF 提供的动态事件和属性系统，你可以监视特定属性是否是某个值，必要时还可重新设置模板样式。在此示例中，你将监视按钮的 [IsMouseOver](#) 属性。当鼠标位于控件上方时，使用新颜色设置 [<Ellipse>](#) 的样式。此触发器类型称为 [PropertyTrigger](#)。

必须为 [<Ellipse>](#) 添加一个可引用的名称，以便于触发器起作用。将其命名为 “backgroundElement”。

XAML

```
<Ellipse x:Name="backgroundElement" Fill="{TemplateBinding Background}"  
Stroke="{TemplateBinding Foreground}" />
```

接下来，将新的 [Trigger](#) 添加到 [ControlTemplate.Triggers](#) 集合。此触发器将监视 [IsMouseOver](#) 事件是否为值 `true`。

XAML

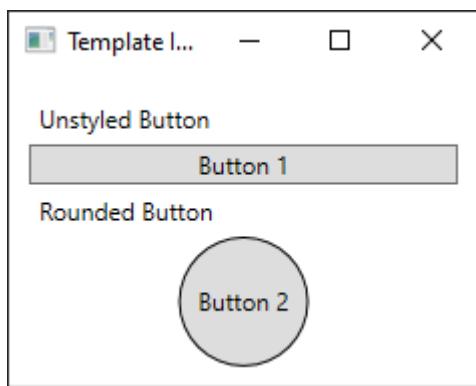
```
<ControlTemplate x:Key="roundbutton" TargetType="Button">  
    <Grid>  
        <Ellipse x:Name="backgroundElement" Fill="{TemplateBinding Background}"  
        Stroke="{TemplateBinding Foreground}" />  
        <ContentPresenter HorizontalAlignment="Center"  
        VerticalAlignment="Center" />  
    </Grid>  
    <ControlTemplate.Triggers>  
        <Trigger Property="IsMouseOver" Value="true">  
            </Trigger>  
    </ControlTemplate.Triggers>  
</ControlTemplate>
```

接下来，将 [<Setter>](#) 添加到 [<Trigger>](#)，后者会将 [<Ellipse>](#) 的 [Fill](#) 属性更改为一种新颜色。

XAML

```
<Trigger Property="IsMouseOver" Value="true">  
    <Setter Property="Fill" TargetName="backgroundElement"  
    Value="AliceBlue"/>  
</Trigger>
```

运行该项目。请注意，当你将鼠标移到按钮上方时，`<Ellipse>` 的颜色会改变。



使用 VisualState

视觉状态由控件定义和触发。例如，当鼠标移到控件上方时，将触发 `CommonStates.MouseOver` 状态。可以基于控件的当前状态对属性更改进行动画处理。在上一个部分中，当 `IsMouseOver` 属性为 `true` 时，使用 `<PropertyTrigger>` 将按钮的前景色更改为 `AliceBlue`。可改为创建一个视觉状态，来对此颜色的更改进行动画处理，以实现平稳过渡。有关 `VisualStates` 的详细信息，请参阅 [WPF 中的样式和模板](#)。

若要将 `<PropertyTrigger>` 转换为动画效果的可视状态，首先要从模板删除 `<ControlTemplate.Triggers>` 元素。

XAML

```
<ControlTemplate x:Key="roundbutton" TargetType="Button">
    <Grid>
        <Ellipse x:Name="backgroundElement" Fill="{TemplateBinding Background}" Stroke="{TemplateBinding Foreground}" />
        <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center" />
    </Grid>
</ControlTemplate>
```

接下来，在控件模板的 `<Grid>` 根中，添加 `<VisualStateManager.VisualStateGroups>`，其中包含 `CommonStates` 的 `<VisualStateGroup>`。定义两种状态：`Normal` 和 `MouseOver`。

XAML

```
<ControlTemplate x:Key="roundbutton" TargetType="Button">
    <Grid>
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup Name="CommonStates">
                <VisualState Name="Normal">
                    </VisualState>
```

```
<VisualState Name="MouseOver">
</VisualState>
</VisualStateManager.VisualStateGroups>
<Ellipse x:Name="backgroundElement" Fill="{TemplateBinding Background}" Stroke="{TemplateBinding Foreground}" />
<ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center" />
</Grid>
</ControlTemplate>
```

触发 `<VisualState>` 时，将应用该状态中定义的任何动画。为每种状态创建动画。动画位于 `<Storyboard>` 元素中。有关情节提要的详细信息，请参阅[情节提要概述](#)。

- 普通

此状态对椭圆填充进行动画处理，将其还原为控件的 `Background` 颜色。

XAML

```
<Storyboard>
<ColorAnimation Storyboard.TargetName="backgroundElement"
Storyboard.TargetProperty="(Shape.Fill).SolidColorBrush.Color)"
To="{TemplateBinding Background}"
Duration="0:0:0.3"/>
</Storyboard>
```

- MouseOver

此状态对椭圆 `Background` 颜色进行动画处理，将其更改为新颜色 `Yellow`。

XAML

```
<Storyboard>
<ColorAnimation Storyboard.TargetName="backgroundElement"
Storyboard.TargetProperty="(Shape.Fill).SolidColorBrush.Color)"
To="Yellow"
Duration="0:0:0.3"/>
</Storyboard>
```

现在，`<ControlTemplate>` 应如下所示。

XAML

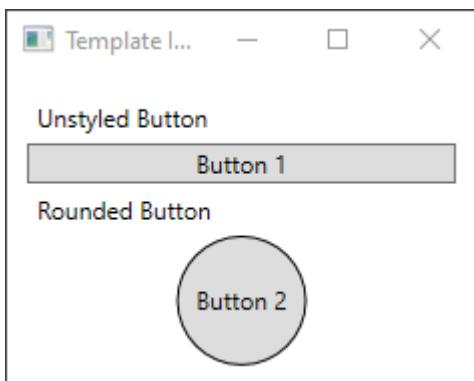
```
<ControlTemplate x:Key="roundbutton" TargetType="Button">
<Grid>
<VisualStateManager.VisualStateGroups>
```

```

<VisualStateGroup Name="CommonStates">
    <VisualState Name="Normal">
        <Storyboard>
            <ColorAnimation
                Storyboard.TargetName="backgroundElement"
                Storyboard.TargetProperty="(Shape.Fill).(
                    SolidColorBrush.Color)"
                To="{TemplateBinding Background}"
                Duration="0:0:0.3"/>
        </Storyboard>
    </VisualState>
    <VisualState Name="MouseOver">
        <Storyboard>
            <ColorAnimation
                Storyboard.TargetName="backgroundElement"
                Storyboard.TargetProperty="(Shape.Fill).(
                    SolidColorBrush.Color)"
                To="Yellow"
                Duration="0:0:0.3"/>
        </Storyboard>
    </VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Ellipse Name="backgroundElement" Fill="{TemplateBinding
Background}" Stroke="{TemplateBinding Foreground}" />
<ContentPresenter x:Name="contentPresenter"
HorizontalContentAlignment="Center" VerticalAlignment="Center" />
</Grid>
</ControlTemplate>

```

运行该项目。请注意，当你将鼠标移到按钮上方时，`<Ellipse>` 的颜色会进行动画处理。



后续步骤

- WPF 中的样式和模板
- XAML 资源概述

如何：查找由 ControlTemplate 生成的元素

项目 • 2023/02/06

此示例演示如何查找由 [ControlTemplate](#) 生成的元素。

示例

下面的示例显示了一个样式，该样式为 [Button](#) 类创建一个简单的 [ControlTemplate](#)：

XAML

```
<Style TargetType="{x:Type Button}">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Button}">
                <Grid Margin="5" Name="grid">
                    <Ellipse Stroke="DarkBlue" StrokeThickness="2">
                        <Ellipse.Fill>
                            <RadialGradientBrush Center="0.3,0.2" RadiusX="0.5"
RadiusY="0.5">
                                <GradientStop Color="Azure" Offset="0.1" />
                                <GradientStop Color="CornflowerBlue" Offset="1.1" />
                            </RadialGradientBrush>
                        </Ellipse.Fill>
                    </Ellipse>
                    <ContentPresenter Name="content" Margin="10"
                           HorizontalAlignment="Center"
                           VerticalAlignment="Center"/>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

若要在应用模板后在模板中查找元素，可调用 [Template](#) 的 [FindName](#) 方法。以下示例创建一个消息框，显示控件模板内的 [Grid](#) 的实际宽度值：

C#

```
// Finding the grid that is generated by the ControlTemplate of the Button
Grid gridInTemplate = (Grid)myButton1.Template.FindName("grid", myButton1);

// Do something to the ControlTemplate-generated grid
MessageBox.Show("The actual width of the grid in the ControlTemplate: "
+ gridInTemplate.GetValue(Grid.ActualWidthProperty).ToString());
```

另请参阅

- [查找由 DataTemplate 生成的元素](#)
- [样式设置和模板化](#)
- [WPF XAML 名称范围](#)
- [WPF 中的树](#)

控件自定义

项目 • 2023/02/06

此类别涵盖用于创建功能齐全的 Windows Presentation Foundation (WPF) 控件的各种基类、接口及其他元素和概念。

本节内容

- [控件创作概述](#)
- [可样式化控件的设计准则](#)
- [装饰器](#)
- [控件样式和模板](#)
- [WPF 自定义控件的 UI 自动化](#)

请参阅

- [样式设置和模板化](#)

控件创作概述

项目 · 2022/09/27

Windows Presentation Foundation (WPF) 控件模型的扩展性极大地减少了创建新控件的需要。但在某些情况下，仍可能需要创建自定义控件。本主题讨论可最大限度减少在 Windows Presentation Foundation (WPF) 中创建自定义控件以及其他控件创作模型的需要的功能。本主题还演示如何创建新控件。

编写新控件的替代方法

以前，如果要通过现有控件获取自定义体验，只能更改控件的标准属性，例如背景色、边框宽度和字号。如果希望在这些预定义参数的基础之上扩展控件的外观或行为，则需要创建新控件，常用的方法是继承现有控件并重写负责绘制该控件的方法。虽然这仍是一种可选方法，但也可以利用 WPF 的丰富内容模型、样式、模板和触发器来自定义现有的控件。下表提供了一些示例，演示如何在不创建新控件的情况下使用这些功能来实现一致的自定义体验。

- 丰富内容。** 很多标准 WPF 控件都支持丰富内容。例如，[Button](#) 的内容属性为 [Object](#) 类型，因此从理论上讲，任何内容都可以显示在 [Button](#) 上。若要使按钮显示图像和文本，可以将图像和 [TextBlock](#) 添加到 [StackPanel](#) 中，然后将 [StackPanel](#) 分配给 [Content](#) 属性。由于这些控件可以显示 WPF 视觉元素和任意数据，因此，降低了创建新控件或修改现有控件来支持复杂可视化效果的需求。有关 [Button](#) 的内容模型和 WPF 中其他内容模型的详细信息，请参阅 [WPF 内容模型](#)。
- 样式。** [Style](#) 是表示控件属性的值的集合。使用样式可创建所需控件外观和行为的可重用表示形式，而无需编写新控件。例如，假设需要所有 [TextBlock](#) 控件的字体都为红色 Arial 字体，并且字号为 14。可以创建一个样式作为资源，并相应设置适当属性。这样，添加到应用程序中的每个 [TextBlock](#) 都将具有相同的外观。
- 数据模板。** 通过 [DataTemplate](#)，可以自定义数据在控件上的显示方式。例如，[DataTemplate](#) 可用于指定数据在 [ListBox](#) 中的显示方式。有关这种情况的示例，请参阅[数据模块化概述](#)。除了自定义数据的外观，[DataTemplate](#) 还可以包含 UI 元素，这大大增加了自定义 UI 的灵活性。例如，使用 [DataTemplate](#) 可以创建 [ComboBox](#)，其中每一项都包含一个复选框。
- 控件模板。** WPF 中的许多控件使用 [ControlTemplate](#) 定义控件的结构和外观，因为它可以将控件的外观和功能区分开来。重新定义 [ControlTemplate](#) 可以极大地更改控件的外观。例如，假设需要一个看起来像交通信号灯的控件。此控件具有简单的用户界面和功能。该控件有三个圆圈，一次只有一个圆圈亮起。经过一番考虑后，用户可能意识到 [RadioButton](#) 提供了一次只选择一项的功能，但是 [RadioButton](#) 的默认外观完全不像交通信号灯上的灯。由于 [RadioButton](#) 使用控件模板定义外观，

因此很容易重新定义 `ControlTemplate` 以符合该控件的要求，从而使用单选按钮来制作交通信号灯。

① 备注

尽管 `RadioButton` 可以使用 `DataTemplate`，但在本例中，光有 `DataTemplate` 还不够。`DataTemplate` 定义控件内容的外观。对于 `RadioButton`，指示是否选择 `RadioButton` 的圆圈右侧显示出来的全部都是该控件的内容。在交通信号灯的示例中，单选按钮只需要是可“点亮”的圆圈。由于交通信号灯的外观要求与 `RadioButton` 的默认外观存在很大差异，因此，有必要重新定义 `ControlTemplate`。一般情况下，`DataTemplate` 用于定义控件的内容（或数据），`ControlTemplate` 用于定义控件的构成方式。

- **触发器。** 使用 `Trigger` 可以动态更改控件的外观和行为，无需创建新控件。例如，假设应用程序中有多个 `ListBox` 控件，但需要每个 `ListBox` 中的项在选中时都显示为红色粗体。用户首先想到的可能是创建一个从 `ListBox` 继承的类，然后重写 `OnSelectionChanged` 方法，以更改选中项的外观，不过，更好的方法是向 `ListBoxItem` 的样式添加一个更改选中项外观的触发器。触发器可以更改属性值或根据属性值执行操作。`EventTrigger` 使用户可以在事件发生时执行操作。

有关样式、模板和触发器的详细信息，请参阅[样式设置和模板化](#)。

一般情况下，如果控件可以镜像现有控件的功能，但希望该控件具有不同的外观，则应先考虑是否可以使用本节中讨论的某些方法来更改现有控件的外观。

控件创作模型

通过丰富内容模型、样式、模板和触发器，最大程度地减少创建新控件的需要。但是，如果确实需要创建新控件，则理解 WPF 中的不同控件创作模型就显得非常重要。WPF 提供三个用于创建控件的常规模型，每个模型都提供不同的功能集和灵活度。三个模型的基类是 `UserControl`、`Control` 和 `FrameworkElement`。

从 `UserControl` 派生

在 WPF 中创建控件的最简单方法是从 `UserControl` 派生。生成继承自 `UserControl` 的控件时，会将现有组件添加到 `UserControl`，命名这些组件，然后在 WPF 中引用事件处理程序。

如果生成无误，`UserControl` 可以利用丰富内容、样式和触发器的优势。但是，如果控件继承自 `UserControl`，则使用该控件的用户将无法使用 `DataTemplate` 或 `ControlTemplate`

来自定义其外观。因此，有必要从 [Control](#) 类或其派生类（[UserControl](#) 除外）之一中派生，以创建支持模板的自定义控件。

从 [UserControl](#) 派生的优点

如果符合以下所有情况，请考虑从 [UserControl](#) 派生：

- 希望采用与生成应用程序相似的方法生成控件。
- 控件仅包含现有组件。
- 无需支持复杂的自定义项。

从 [Control](#) 派生

从 [Control](#) 类派生是大多数现有 WPF 控件使用的模型。创建从 [Control](#) 类派生的控件时，可使用模板定义它的外观。通过这种方式，可以将运算逻辑从视觉表示形式中分离出来。通过使用命令和绑定（而不是事件）并尽可能避免引用 [ControlTemplate](#) 中的元素，也可确保分离 UI 和逻辑。如果控件的 UI 和逻辑正确分离，该控件的用户即可重新定义控件的 [ControlTemplate](#)，从而自定义其外观。尽管生成自定义 [Control](#) 不像生成 [UserControl](#) 那样简单，但自定义 [Control](#) 还是提供了最大的灵活性。

从 [Control](#) 派生的优点

如果符合以下任一情况，请考虑从 [Control](#) 派生，而不要使用 [UserControl](#) 类：

- 希望建立控件的外观能通过 [ControlTemplate](#) 进行自定义。
- 希望建立控件支持不同的主题。

从 [FrameworkElement](#) 派生

从 [UserControl](#) 或 [Control](#) 派生的控件依赖于组合现有元素。在很多情况下，这是一种可接受的解决方案，因为从 [FrameworkElement](#) 继承的任何对象都可以位于 [ControlTemplate](#) 中。但是，某些时候，简单的元素组合不能满足控件的外观需要。对于这些情况，使组件基于 [FrameworkElement](#) 才是正确的选择。

生成基于 [FrameworkElement](#) 的组件有两种标准方法：直接呈现和自定义元素组合。直接呈现涉及的操作包括：重写 [FrameworkElement](#) 的 [OnRender](#) 方法，并提供显式定义组件视觉对象的 [DrawingContext](#) 操作。这是由 [Image](#) 和 [Border](#) 使用的方法。自定义元素组合涉及的操作包括：使用 [Visual](#) 类型的对象组合组件的外观。有关示例，请参阅[使](#)

用 DrawingVisual 对象。 Track 是 WPF 中使用自定义元素组合的控件示例。在同一控件中，也可以混合使用直接呈现和自定义元素组合。

从 FrameworkElement 派生的优点

如果符合以下任何情况，请考虑从 FrameworkElement 派生：

- 希望对控件的外观进行精确控制，而不仅仅是简单的元素组合提供的效果。
- 希望通过定义自己的呈现逻辑来定义控件的外观。
- 希望以一种 UserControl 和 Control 之外的新颖方式组合现有元素。

控件创作基础知识

如前所述，WPF 的最强大功能之一在于，它能够在不需要创建自定义控件的情况下，不只是通过设置控件的基本属性来更改其外观和行为。样式设置、数据绑定和触发器功能通过 WPF 属性系统和 WPF 事件系统实现。以下各部分介绍应遵循的一些做法（与创建自定义控件时所用的模型无关），以便自定义控件的用户可以像使用 WPF 附带的控件一样使用这些功能。

使用依赖属性

当属性为依赖属性时，可以执行以下操作：

- 在样式中设置该属性。
- 将该属性绑定到数据源。
- 使用动态资源作为该属性的值。
- 对该属性进行动画处理。

如果希望控件的属性支持以上任一功能，应将该属性实现为依赖属性。下面的示例通过执行以下操作定义一个名为 Value 的依赖属性：

- 将一个名为 ValueProperty 的 DependencyProperty 标识符定义为

```
public static readonly
```

 字段。
- 通过调用 DependencyProperty.Register 向属性系统注册该属性名，以指定以下内容：
 - 属性的名称。

- 属性的类型。
- 拥有属性的类型。
- 属性的元数据。 元数据包含属性的默认值、[CoerceValueCallback](#) 和 [PropertyChangedCallback](#)。
- 通过实现该属性的 `get` 和 `set` 访问器，定义一个名为 `Value`（与用来注册该依赖属性的名称相同）的 CLR 包装器属性。 请注意，`get` 和 `set` 访问器仅分别调用 [GetValue](#) 和 [SetValue](#)。 建议依赖属性的访问器不要包含其他逻辑，因为客户端和 WPF 可绕过这两个访问器直接调用 [GetValue](#) 和 [SetValue](#)。 例如，如果属性绑定到数据源，则不会调用该属性的 `set` 访问器。 不要向 `get` 和 `set` 访问器添加其他逻辑，应使用 [ValidateValueCallback](#)、[CoerceValueCallback](#) 和 [PropertyChangedCallback](#) 委托在值更改时进行响应或检查该值。 有关这些回叫的详细信息，请参阅[依赖属性回叫和验证](#)。
- 为 [CoerceValueCallback](#) 定义方法，名为 `CoerceValue`。`CoerceValue` 确保 `Value` 大于或等于 `MinValue` 且小于或等于 `MaxValue`。
- 为 [PropertyChangedCallback](#) 定义方法，名为 `OnValueChanged`。`OnValueChanged` 创建一个 [RoutedPropertyChangedEventArgs<T>](#) 对象，并准备引发 `ValueChanged` 路由事件。 路由事件在下一节中讨论。

C#

```

/// <summary>
/// Identifies the Value dependency property.
/// </summary>
public static readonly DependencyProperty ValueProperty =
    DependencyProperty.Register(
        "Value", typeof(decimal), typeof(NumericUpDown),
        new FrameworkPropertyMetadata(MinValue, new
    PropertyChangedCallback(OnValueChanged),
        new
    CoerceValueCallback(CoerceValue)));

/// <summary>
/// Gets or sets the value assigned to the control.
/// </summary>
public decimal Value
{
    get { return (decimal)GetValue(ValueProperty); }
    set { SetValue(ValueProperty, value); }
}

private static object CoerceValue(DependencyObject element, object value)
{
    decimal newValue = (decimal)value;
    NumericUpDown control = (NumericUpDown)element;
}

```

```

        newValue = Math.Max(MinValue, Math.Min(MaxValue, newValue));

        return newValue;
    }

    private static void OnValueChanged(DependencyObject obj,
    DependencyPropertyChangedEventArgs args)
    {
        NumericUpDown control = (NumericUpDown)obj;

        RoutedPropertyChangedEventArgs<decimal> e = new
        RoutedPropertyChangedEventArgs<decimal>(
            (decimal)args.OldValue, (decimal)args.NewValue, ValueChangedEvent);
        control.OnValueChanged(e);
    }
}

```

有关详细信息，请参阅[自定义依赖属性](#)。

使用路由事件

就像依赖属性以附加功能扩展 CLR 属性的概念一样，路由事件扩展标准 CLR 事件的概念。在创建新的 WPF 控件时，将事件实现为路由事件也是一种好方法，因为路由事件支持以下行为：

- 事件可以在多个控件的父级上进行处理。如果事件是浮升事件，元素树中的单个父级可订阅该事件。然后，应用程序作者可以使用一个处理程序来响应多个控件的该事件。例如，如果控件属于 [ListBox](#) 中的每个项（因为它包含在 [DataTemplate](#) 中），应用程序开发人员可以为该控件的 [ListBox](#) 事件定义相应的事件处理程序。每当其中任何控件发生该事件时，都会调用该事件处理程序。
- 路由事件可在 [EventSetter](#) 中使用，应用程序开发人员通过它可以在样式内指定事件的处理程序。
- 路由事件可在 [EventTrigger](#) 中使用，这对于使用 XAML 对属性进行动画处理很有用。有关详细信息，请参阅[动画概述](#)。

下面的示例通过执行以下操作定义了一个路由事件：

- 将一个名为 `ValueChangedEvent` 的 [RoutedEvent](#) 标识符定义为
`public static readonly` 字段。
- 可通过调用 [EventManager.RegisterRoutedEventArgs](#) 方法来注册路由事件。该示例在调用 [RegisterRoutedEventArgs](#) 时指定以下信息：
 - 事件名称是 `ValueChanged`。

- 路由策略为 `Bubble`，这意味着首先调用源（引发事件的对象）上的事件处理程序，然后从最近的父元素上的事件处理程序开始，相继调用源的各个父元素上的事件处理程序。
- 事件处理程序的类型是 `RoutedPropertyChangedEventHandler<T>`（使用 `Decimal` 类型构造）。
- 该事件的所属类型为 `NumericUpDown`。
- 声明一个名为 `ValueChanged` 的公共事件，并包含事件访问器声明。该示例调用 `add` 访问器声明中的 `AddHandler` 和 `remove` 访问器声明中的 `RemoveHandler` 来使用 WPF 事件服务。
- 创建一个名为 `OnValueChanged` 的受保护的虚拟方法，该方法会引发 `ValueChanged` 事件。

C#

```

/// <summary>
/// Identifies the ValueChanged routed event.
/// </summary>
public static readonly RoutedEvent ValueChangedEvent =
EventManager.RegisterRoutedEvent(
    "ValueChanged", RoutingStrategy.Bubble,
    typeof(RoutedPropertyChangedEventHandler<decimal>),
    typeof(NumericUpDown));

/// <summary>
/// Occurs when the Value property changes.
/// </summary>
public event RoutedPropertyChangedEventHandler<decimal> ValueChanged
{
    add { AddHandler(ValueChangedEvent, value); }
    remove { RemoveHandler(ValueChangedEvent, value); }
}

/// <summary>
/// Raises the ValueChanged event.
/// </summary>
/// <param name="args">Arguments associated with the ValueChanged event.
</param>
protected virtual void
OnValueChanged(RoutedEventArgs<decimal> args)
{
    RaiseEvent(args);
}

```

有关详细信息，请参阅[路由事件概述](#)和[创建自定义路由事件](#)。

使用绑定

若要将控件的 UI 与其逻辑分离，请考虑使用数据绑定。如果使用 [ControlTemplate](#) 定义控件的外观，这一点尤其重要。使用数据绑定时，或许可以避免在代码中引用 UI 的特定部分。最好避免引用 [ControlTemplate](#) 中的元素，因为当代码引用 [ControlTemplate](#) 中的元素并且 [ControlTemplate](#) 发生更改时，需要将引用的元素包含在新的 [ControlTemplate](#) 中。

下面的示例更新 [NumericUpDown](#) 控件的 [TextBlock](#)，向它分配一个名称，然后在代码中按名称引用该文本框。

XAML

```
<Border BorderThickness="1" BorderBrush="Gray" Margin="2"
        Grid.RowSpan="2" VerticalAlignment="Center"
        HorizontalAlignment="Stretch">
    <TextBlock Name="valueText" Width="60" TextAlignment="Right" Padding="5"/>
</Border>
```

C#

```
private void UpdateTextBlock()
{
    valueText.Text = Value.ToString();
}
```

下面的示例使用绑定来达到相同的目的。

XAML

```
<Border BorderThickness="1" BorderBrush="Gray" Margin="2"
        Grid.RowSpan="2" VerticalAlignment="Center"
        HorizontalAlignment="Stretch">

    <!--Bind the TextBlock to the Value property-->
    <TextBlock
        Width="60" TextAlignment="Right" Padding="5"
        Text="{Binding RelativeSource={RelativeSource FindAncestor,
            AncestorType={x:Type local:NumericUpDown}},
            Path=Value}"/>

</Border>
```

有关数据绑定的详细信息，请参阅[数据绑定概述](#)。

设计器的设计

若要在适用于 Visual Studio 的 WPF 设计器中获得对自定义 WPF 控件的支持（例如，使用“属性”窗口编辑属性），请遵循以下准则。有关针对 WPF 设计器进行开发的详细信息，请参阅在 [Visual Studio 中设计 XAML](#)。

依赖项属性

确保实现 CLR `get` 和 `set` 访问器，如前面的“使用依赖属性”中所述。设计器可以使用包装器来检测某个依赖属性是否存在，但与 WPF 和控件客户端一样，在获取或设置属性时不需要使用设计器来调用访问器。

附加属性

应使用以下准则在自定义控件上实现附加属性：

- 具有一个使用 [RegisterAttached](#) 方法创建的
`public static readonly DependencyProperty`，其形式为 `PropertyNameProperty`。
传递到 [RegisterAttached](#) 的属性名称必须与 `PropertyName` 匹配。
- 实现一对名为 `Set 属性名称` 和 `Get 属性名称` 的 `public static` CLR 方法。这两种方法都应接受从 `DependencyProperty` 派生的类作为其第一个参数。`Set 属性名称` 方法还接受其类型与属性的注册数据类型匹配的参数。`Get 属性名称` 方法应返回相同类型的值。如果缺少 `Set 属性名称` 方法，则该属性标记为只读。
- `Set PropertyName` 和 `Get PropertyName` 必须分别直接路由到目标依赖对象的 `GetValue` 和 `SetValue` 方法。通过调用方法包装器或直接调用目标依赖对象，设计器可以访问附加属性。

有关附加属性的详细信息，请参阅[附加属性概述](#)。

定义和使用共享资源

可以将控件包含在应用程序所在的程序集中，也可以将控件打包到可在多个应用程序中使用的单独程序集中。大多数情况下，不管使用什么方法，本主题中讨论的信息都适用。但有一处差异值得注意。将控件放入应用程序所在的程序集中时，可以随意向 `App.xaml` 文件添加全局资源。但只包含控件的程序集没有与之关联的 `Application` 对象，因此 `App.xaml` 文件不可用。

当应用程序查找资源时，它会按以下顺序在三个级别进行查找：

1. 元素级别。

系统从引用该资源的元素开始搜索，接着搜索逻辑父级的资源，依此类推，直至到达根元素。

2. 应用程序级别。

由 [Application](#) 对象定义的资源。

3. 主题级别。

主题级别的字典存储在名为“Themes”的子文件夹中。 Themes 文件夹中的文件与主题对应。例如，可能有 Aero.NormalColor.xaml、Luna.NormalColor.xaml、Royale.NormalColor.xaml 等。可能还有一个名为 generic.xaml 的文件。当系统在主题级别查找资源时，它会先在特定于主题的文件中查找相应资源，然后在 generic.xaml 中进行查找。

当控件位于独立于应用程序的程序集中时，必须将全局资源放在元素级别或主题级别。这两种方法都各有优点。

在元素级别定义资源

可以通过创建自定义资源字典并将其与控件的资源字典合并，在元素级别定义共享资源。采用此方法时，可以为资源文件指定任意名称，并且资源文件可以与控件位于同一文件夹中。元素级别的资源还可以使用简单字符串作为键。下面的示例创建一个名为 Dictionary1.xaml 的 [LinearGradientBrush](#) 文件。

XAML

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <LinearGradientBrush
        x:Key="myBrush"
        StartPoint="0,0" EndPoint="1,1">
        <GradientStop Color="Red" Offset="0.25" />
        <GradientStop Color="Blue" Offset="0.75" />
    </LinearGradientBrush>
</ResourceDictionary>
```

定义字典后，需要将其与控件的资源字典合并。可以使用 XAML 或代码执行此操作。

下面的示例通过使用 XAML 合并资源字典。

XAML

```
<UserControl.Resources>
    <ResourceDictionary>
```

```
<ResourceDictionary.MergedDictionaries>
    <ResourceDictionary Source="Dictionary1.xaml"/>
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</UserControl.Resources>
```

这种方法的缺点是每次引用它时都会创建一个 [ResourceDictionary](#) 对象。例如，如果库中有 10 个自定义控件，并且通过使用 XAML 来合并每个控件的共享资源字典，则会创建 10 个完全相同的 [ResourceDictionary](#) 对象。通过创建一个在代码中合并资源并返回生成的 [ResourceDictionary](#) 的静态类，可以避免出现这种情况。

下面的示例创建了一个返回共享 [ResourceDictionary](#) 的类。

C#

```
internal static class SharedDictionaryManager
{
    internal static ResourceDictionary SharedDictionary
    {
        get
        {
            if (_sharedDictionary == null)
            {
                System.Uri resourceLocater =
                    new
System.Uri("/ElementResourcesCustomControlLibrary;component/Dictionary1.xaml",
",
                    System.UriKind.Relative);

                _sharedDictionary =
                    (ResourceDictionary)Application.LoadComponent(resourceLocater);
            }

            return _sharedDictionary;
        }
    }

    private static ResourceDictionary _sharedDictionary;
}
```

下面的示例先在一个自定义控件的构造函数中将共享资源与该控件的资源合并，然后再调用 [InitializeComponent](#)。由于 [SharedDictionaryManager.SharedDictionary](#) 为静态属性，因此 [ResourceDictionary](#) 仅创建一次。因为资源字典在调用 [InitializeComponent](#) 前已合并，所以控件可以在它的 XAML 文件中使用资源。

C#

```
public NumericUpDown()
{
    this.Resources.MergedDictionaries.Add(SharedDictionaryManager.SharedDictionary);
    InitializeComponent();
}
```

在主题级别定义资源

通过 WPF 可以为不同的 Windows 主题创建资源。作为控件作者，可以为特定主题定义资源，以根据所使用的主题更改控件的外观。例如，Windows 经典主题（Windows 2000 的默认主题）中 [Button](#) 的外观不同于 Windows Luna 主题（Windows XP 的默认主题）中 [Button](#) 的外观，因为 [Button](#) 针对每种主题使用的 [ControlTemplate](#) 不同。

特定于主题的资源以特定文件名保留在资源字典中。这些文件必须位于一个名为 [Themes](#) 的文件夹中，此文件夹是包含该控件的文件夹的子文件夹。下表列出了资源字典文件以及与每个文件关联的主题：

资源字典文件名	Windows 主题
Classic.xaml	Windows XP 中的经典 Windows 9x/2000 外观
Luna.NormalColor.xaml	Windows XP 上的默认蓝色主题
Luna.Homestead.xaml	Windows XP 上的橄榄色主题
Luna.Metallic.xaml	Windows XP 上的银色主题
Royale.NormalColor.xaml	Windows XP Media Center Edition 上的默认主题
Aero.NormalColor.xaml	Windows Vista 上的默认主题

无需为每一种主题都定义资源。如果没有为特定主题定义资源，控件将在 [classic.xaml](#) 中检查资源。如果在与当前主题对应的文件或 [Classic.xaml](#) 中没有定义资源，控件将使用常规资源，该资源位于名为 [generic.xaml](#) 的资源字典文件中。[generic.xaml](#) 文件与特定于主题的资源字典文件位于同一文件夹中。尽管 [generic.xaml](#) 不与特定的 Windows 主题对应，但它仍是一个主题级别字典。

带有主题和 UI 自动化支持的 [C#](#) 或 [Visual Basic](#) [NumericUpDown](#) 自定义控件示例包含两个用于 [NumericUpDown](#) 控件的资源字典：一个在 [generic.xaml](#) 中，另一个在 [Luna.NormalColor.xaml](#) 中。

将 [ControlTemplate](#) 放入任何特定于主题的资源字典文件中时，都必须为控件创建静态构造函数，并对 [DefaultStyleKey](#) 调用 [OverrideMetadata\(Type, PropertyMetadata\)](#) 方法，

如下例所示。

C#

```
static NumericUpDown()
{
    DefaultStyleKeyProperty.OverrideMetadata(typeof(NumericUpDown),
        new FrameworkPropertyMetadata(typeof(NumericUpDown)));
}
```

定义和引用主题资源的键

在元素级别定义资源时，可以指定一个字符串作为它的键，然后通过该字符串访问该资源。在主题级别定义资源时，必须使用 [ComponentResourceKey](#) 作为键。以下示例定义 generic.xaml 中的资源。

XAML

```
<LinearGradientBrush
    x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type
local:Painter},
                                ResourceId=MyEllipseBrush}"
    StartPoint="0,0" EndPoint="1,0">
    <GradientStop Color="Blue" Offset="0" />
    <GradientStop Color="Red" Offset="0.5" />
    <GradientStop Color="Green" Offset="1"/>
</LinearGradientBrush>
```

以下示例通过指定 [ComponentResourceKey](#) 作为键来引用该资源。

XAML

```
<RepeatButton
    Grid.Column="1" Grid.Row="0"
    Background="{StaticResource {ComponentResourceKey
        TypeInTargetAssembly={x:Type local:NumericUpDown},
        ResourceId=ButtonBrush}}">
    Up
</RepeatButton>
<RepeatButton
    Grid.Column="1" Grid.Row="1"
    Background="{StaticResource {ComponentResourceKey
        TypeInTargetAssembly={x:Type local:NumericUpDown},
        ResourceId=ButtonBrush}}">
    Down
</RepeatButton>
```

指定主题资源的位置

若要找到控件的资源，承载应用程序需要知道相应程序集包含特定于控件的资源。可以通过向包含此控件的程序集添加 [ThemeInfoAttribute](#) 来达到此目的。

[ThemeInfoAttribute](#) 具有一个 [GenericDictionaryLocation](#) 属性和一个 [ThemeDictionaryLocation](#) 属性，前者指定常规资源的位置，后者指定特定于主题的资源的位置。

下面的示例将 [GenericDictionaryLocation](#) 和 [ThemeDictionaryLocation](#) 属性设置为 [SourceAssembly](#)，以指定常规资源和特定于主题的资源与控件位于同一程序集中。

C#

```
[assembly: ThemeInfo(ResourceDictionaryLocation.SourceAssembly,
    ResourceDictionaryLocation.SourceAssembly)]
```

另请参阅

- 在 [Visual Studio 中设计 XAML](#)
- [WPF 中的 Pack URI](#)
- [控件自定义](#)

创建具有可自定义外观的控件

项目 • 2022/09/27

Windows Presentation Foundation (WPF) 使你能够创建可自定义其外观的控件。例如，可以通过创建新的 [CheckBox](#) 来更改 [ControlTemplate](#) 的外观（可超过设置属性的功能）。下图显示了一个使用默认 [ControlTemplate](#) 的 [CheckBox](#)，以及一个使用自定义 [ControlTemplate](#) 的 [CheckBox](#)。

CheckBox1 使用默认控件模板的 CheckBox

CheckBox2



使用自定义控件模板的 CheckBox

如果在创建控件时遵循部件和状态模型，则控件的外观可自定义。Blend for Visual Studio 等设计器工具支持部件和状态模型，因此在遵循此模型时，控件可在这些类型的应用程序中进行自定义。本主题讨论部件和状态模型，以及如何在创建自己的控件时遵循它。本主题使用自定义控件 [NumericUpDown](#) 的示例来说明此模型的理念。

[NumericUpDown](#) 控件显示一个数值，用户可通过单击控件的按钮来增大或减小该值。下图显示了本主题中讨论的 [NumericUpDown](#) 控件。



自定义 NumericUpDown 控件

本主题包含以下各节：

- [先决条件](#)
- [部件和状态模型](#)
- 在 [ControlTemplate](#) 中定义控件的视觉结构和视觉行为
- 在代码中使用 [ControlTemplate](#) 的部件
- 提供控件协定
- 完整示例

先决条件

本主题假定你知道如何为现有控件创建新 [ControlTemplate](#) 控件，熟悉控件协定中的元素，并了解在[创建控件模板](#)中讨论的概念。

① 备注

若要创建可以自定义其外观的控件，必须创建从 `Control` 类或其子类之一（`UserControl` 除外）继承的控件。继承自 `UserControl` 的控件是可快速创建的控件，但它不使用 `ControlTemplate`，你无法自定义其外观。

部件和状态模型

部件和状态模型指定如何定义控件的视觉结构和视觉行为。 若要遵循部件和状态模型，应执行以下操作：

- 在控件的 `ControlTemplate` 中定义视觉结构和视觉行为。
- 当控件的逻辑与控件模板的部件交互时，请遵循某些最佳做法。
- 提供控件协定以指定 `ControlTemplate` 中应包含的内容。

在控件的 `ControlTemplate` 中定义视觉结构和视觉行为时，应用程序作者可以通过创建新 `ControlTemplate`（而不是编写代码）来更改控件的视觉结构和视觉行为。 必须提供控件协定，告知应用程序作者应在 `ControlTemplate` 中定义哪些 `FrameworkElement` 对象和状态。 与 `ControlTemplate` 中的部件交互时，应遵循一些最佳做法，以便控件可正确处理不完整的 `ControlTemplate`。 如果遵循这三个原则，应用程序作者能够如同针对 WPF 附带的控件一样，轻松地针对控件创建 `ControlTemplate`。 以下部分详细说明了其中每个建议。

在 `ControlTemplate` 中定义控件的视觉结构和视觉行为

使用部件和状态模型创建自定义控件时，可在其 `ControlTemplate` 中定义控件的视觉结构和视觉行为，而不是在其逻辑中。 控件的视觉结构是构成控件的 `FrameworkElement` 对象的组合。 视觉行为是控件处于特定状态时的显示方式。 有关创建指定控件的视觉结构和视觉行为的 `ControlTemplate` 的详细信息，请参阅[创建控件模板](#)。

在 `NumericUpDown` 控件示例中，视觉结构包括两个 `RepeatButton` 控件和一个 `TextBlock`。 如果在 `NumericUpDown` 控件的代码中（例如在其构造函数中）添加这些控件，则这些控件的位置不可更改。 应在 `ControlTemplate` 中定义控件的视觉结构和视觉行为，而不是在其代码中定义。 随后应用程序开发人员自定义按钮和 `TextBlock` 的位置，并指定当 `Value` 为负数时发生的行为，因为可以替换 `ControlTemplate`。

以下示例演示 `NumericUpDown` 控件的可视结构，其中包括 `RepeatButton`（用于增大 `Value`）、`RepeatButton`（用于减小 `Value`）和 `TextBlock`（用于显示 `Value`）。

XAML

```
<ControlTemplate TargetType="src:NumericUpDown">
    <Grid Margin="3"
        Background="{TemplateBinding Background}">
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition/>
                <RowDefinition/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>

            <Border BorderThickness="1" BorderBrush="Gray"
                Margin="7,2,2,2" Grid.RowSpan="2"
                Background="#E0FFFFFF"
                VerticalAlignment="Center"
                HorizontalAlignment="Stretch">

                <!--Bind the TextBlock to the Value property-->
                <TextBlock Name="TextBlock"
                    Width="60" TextAlignment="Right" Padding="5"
                    Text="{Binding RelativeSource={RelativeSource
FindAncestor,
                    AncestorType={x:Type src:NumericUpDown}},Path=Value}"/>
            </Border>

            <RepeatButton Content="Up" Margin="2,5,5,0"
                Name="UpButton"
                Grid.Column="1" Grid.Row="0"/>
            <RepeatButton Content="Down" Margin="2,0,5,5"
                Name="DownButton"
                Grid.Column="1" Grid.Row="1"/>

            <Rectangle Name="FocusVisual" Grid.ColumnSpan="2" Grid.RowSpan="2"
                Stroke="Black" StrokeThickness="1"
                Visibility="Collapsed"/>
        </Grid>
    </Grid>
</ControlTemplate>
```

`NumericUpDown` 控件的视觉行为是值在为负数时为红色字体。如果在 `Value` 为负数时在代码中更改 `TextBlock` 的 `Foreground`，则 `NumericUpDown` 会始终显示红色负值。通过将 `VisualState` 对象添加到 `ControlTemplate`，在 `ControlTemplate` 中指定控件的视觉行为。

以下示例演示 `Positive` 和 `Negative` 状态的 `VisualState` 对象。`Positive` 和 `Negative` 相互排斥（控件始终恰好位于这两个控件中的一个），因此该示例将 `VisualState` 对象置于单个 `VisualStateGroup` 中。当控件进入 `Negative` 状态时，`TextBlock` 的 `Foreground` 会变为红色。当控件处于 `Positive` 状态时，`Foreground` 会恢复为其原始值。在[创建控件模板](#)中进一步讨论了如何在 `ControlTemplate` 中定义 `VisualState` 对象。

① 备注

请务必对 `ControlTemplate` 的根 `FrameworkElement` 设置 `VisualStateManager.VisualStateGroups` 附加属性。

XAML

```
<ControlTemplate TargetType="local:NumericUpDown">
    <Grid Margin="3"
        Background="{TemplateBinding Background}">

        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup Name="ValueStates">

                <!--Make the Value property red when it is negative.-->
                <VisualState Name="Negative">
                    <Storyboard>
                        <ColorAnimation To="Red"
                            Storyboard.TargetName="TextBlock"
                            Storyboard.TargetProperty="(Foreground).(Color)"/>
                    </Storyboard>

                </VisualState>

                <!--Return the TextBlock's Foreground to its
                    original color.-->
                <VisualState Name="Positive"/>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
    </Grid>
</ControlTemplate>
```

在代码中使用 `ControlTemplate` 的部件

`ControlTemplate` 作者可能会有意或错误地省略 `FrameworkElement` 或 `VisualState` 对象，但控件的逻辑可能需要这些部件才能正常运行。部件和状态模型指定，控件应可还原到缺少 `FrameworkElement` 或 `VisualState` 对象的 `ControlTemplate`。如果 `ControlTemplate` 中缺少 `FrameworkElement`、`VisualState` 或 `VisualStateGroup`，控件不应引发异常或报告错误。本部分介绍有关与 `FrameworkElement` 对象交互和管理状态的建议做法。

预计缺少 FrameworkElement 对象

在 `ControlTemplate` 中定义 `FrameworkElement` 对象时，控件的逻辑可能需要与其中一些对象进行交互。例如，`NumericUpDown` 控件订阅按钮的 `Click` 事件以增大或减小 `Value`，并将 `TextBlock` 的 `Text` 属性设置为 `Value`。如果自定义 `ControlTemplate` 省略 `TextBlock` 或按钮，则控件失去一些功能是可以接受的，但应确保控件不会导致错误。例如，如果 `ControlTemplate` 不包含用于更改 `Value` 的按钮，则 `NumericUpDown` 会失去该功能，但使用 `ControlTemplate` 的应用程序会继续运行。

以下做法可确保控件正确响应缺少 `FrameworkElement` 对象：

1. 为代码中需要引用的每个 `FrameworkElement` 设置 `x:Name` 属性。
2. 为需要与之交互的每个 `FrameworkElement` 定义私有属性。
3. 订阅和取消订阅控件在 `FrameworkElement` 属性 `set` 访问器中处理的任何事件。
4. 设置步骤 2 中在 `OnApplyTemplate` 方法中定义的 `FrameworkElement` 属性。这是 `ControlTemplate` 中的 `FrameworkElement` 可供控件使用的最早时间。使用 `FrameworkElement` 的 `x:Name` 从 `ControlTemplate` 获取它。
5. 访问其成员之前，检查 `FrameworkElement` 是否不是 `null`。如果是 `null`，不要报告错误。

以下示例演示 `NumericUpDown` 控件如何按照前面列表中的建议与 `FrameworkElement` 对象交互。

在 `ControlTemplate` 中定义 `NumericUpDown` 控件的视觉结构的示例中，增大 `Value` 的 `RepeatButton` 将其 `x:Name` 属性设置为 `UpButton`。下面的示例声明一个名为 `UpButtonElement` 的属性，它表示在 `ControlTemplate` 中声明的 `RepeatButton`。`set` 访问器会在 `UpDownElement` 不是 `null` 时首先取消订阅按钮的 `Click` 事件，然后设置该属性，再然后订阅 `Click` 事件。还为另一个 `RepeatButton` 定义了一个名为 `DownButtonElement` 的属性，但未在此处显示。

C#

```
private RepeatButton upButtonElement;

private RepeatButton UpButtonElement
{
    get
    {
        return upButtonElement;
    }

    set
```

```

    {
        if (upButtonElement != null)
        {
            upButtonElement.Click -=
                new RoutedEventHandler(upButtonElement_Click);
        }
        upButtonElement = value;

        if (upButtonElement != null)
        {
            upButtonElement.Click +=
                new RoutedEventHandler(upButtonElement_Click);
        }
    }
}

```

以下示例演示 `NumericUpDown` 控件的 `OnApplyTemplate`。该示例使用 `GetTemplateChild` 方法从 `ControlTemplate` 获取 `FrameworkElement` 对象。请注意，该示例可防范 `GetTemplateChild` 找到具有指定名称，但不属于预期类型的 `FrameworkElement` 的情况。另一种最佳做法是忽略具有指定 `x:Name`，但属于错误类型的元素。

C#

```

public override void OnApplyTemplate()
{
    UpButtonElement = GetTemplateChild("UpButton") as RepeatButton;
    DownButtonElement = GetTemplateChild("DownButton") as RepeatButton;
    //TextElement = GetTemplateChild("TextBlock") as TextBlock;

    UpdateStates(false);
}

```

按照前面示例中演示的做法操作，可确保控件在 `ControlTemplate` 缺少 `FrameworkElement` 时继续运行。

使用 `VisualStateManager` 管理状态

`VisualStateManager` 会跟踪控件的状态，并执行在状态之间转换所需的逻辑。将 `VisualState` 对象添加到 `ControlTemplate` 时，会将它们添加到 `VisualStateGroup`，并将 `VisualStateGroup` 添加到 `VisualStateManager.VisualStateGroups` 附加属性，以便 `VisualStateManager` 有权访问它们。

以下示例重复前面显示与控件的 `Positive` 和 `Negative` 状态对应的 `VisualState` 对象的示例。`Negative` `VisualState` 中的 `Storyboard` 会将 `TextBlock` 的 `Foreground` 转变为红色。当 `NumericUpDown` 控件处于 `Negative` 状态时，`Negative` 状态下的情节提要会开始。随后 `Negative` 状态下的 `Storyboard` 会在控件恢复为 `Positive` 状态时停止。

`Positive` `VisualState` 不需要包含 `Storyboard`，因为当 `Negative` 的 `Storyboard` 停止时，`Foreground` 会恢复为其原始颜色。

XAML

```
<ControlTemplate TargetType="local:NumericUpDown">
    <Grid Margin="3"
        Background="{TemplateBinding Background}">

        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup Name="ValueStates">

                <!--Make the Value property red when it is negative.-->
                <VisualState Name="Negative">
                    <Storyboard>
                        <ColorAnimation To="Red"
                            Storyboard.TargetName="TextBlock"
                            Storyboard.TargetProperty="(Foreground).(Color)"/>
                    </Storyboard>
                </VisualState>

                <!--Return the TextBlock's Foreground to its
                    original color.-->
                <VisualState Name="Positive"/>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
    </Grid>
</ControlTemplate>
```

请注意，为 `TextBlock` 提供了名称，但 `TextBlock` 不在 `NumericUpDown` 的控件协定中，因为控件的逻辑从不引用 `TextBlock`。在 `ControlTemplate` 中引用的元素具有名称，但不需要是控件协定的一部分，因为控件的新 `ControlTemplate` 可能不需要引用该元素。例如，为 `NumericUpDown` 创建新 `ControlTemplate` 的用户可能会决定不通过更改 `Foreground` 来指示 `Value` 是负值。在这种情况下，代码和 `ControlTemplate` 都不会按名称引用 `TextBlock`。

控件的逻辑负责更改控件的状态。以下示例演示 `NumericUpDown` 控件在 `Value` 为 0 或更大时调用 `GoToState` 方法以进入 `Positive` 状态，在 `Value` 小于 0 时进入 `Negative` 状态。

C#

```
if (Value >= 0)
{
    VisualStateManager.GoToState(this, "Positive", useTransitions);
}
else
{
```

```
        VisualStateManager.GoToState(this, "Negative", useTransitions);
    }
```

`GoToState` 方法会执行所需逻辑以便相应地启动和停止情节提要。当控件调用 `GoToState` 以更改其状态时，`VisualStateManager` 会执行以下操作：

- 如果控件即将进入的 `VisualStyle` 具有 `Storyboard`，则情节提要会开始。那么，如果控件即将退出的 `VisualStyle` 具有 `Storyboard`，则情节提要会结束。
- 如果控件已处于指定的状态，则 `GoToState` 不执行任何操作并返回 `true`。
- 如果指定的状态在 `control` 的 `ControlTemplate` 中不存在于，则 `GoToState` 不执行任何操作并返回 `false`。

使用 `VisualStateManager` 的最佳做法

建议执行以下操作来维护控件的状态：

- 使用属性跟踪其状态。
- 创建帮助程序方法以在状态之间转换。

`NumericUpDown` 控件使用其 `Value` 属性跟踪它是处于 `Positive` 还是 `Negative` 状态。

`NumericUpDown` 控件还定义了 `Focused` 和 `UnFocused` 状态，会跟踪 `IsFocused` 属性。如果使用不是天然对应于控件属性的状态，则可以定义私有属性以跟踪状态。

更新所有状态的单个方法可集中对 `VisualStateManager` 的调用并使代码可管理。下面的示例演示 `NumericUpDown` 控件的帮助程序方法 `UpdateStates`。当 `Value` 大于或等于 0 时，`Control` 处于 `Positive` 状态。当 `Value` 小于 0 时，控件处于 `Negative` 状态。当 `IsFocused` 为 `true` 时，控件处于 `Focused` 状态；否则，它处于 `Unfocused` 状态。无论状态发生何种更改，控件都可以在每次需要更改其状态时调用 `UpdateStates`。

C#

```
private void UpdateStates(bool useTransitions)
{
    if (Value >= 0)
    {
        VisualStateManager.GoToState(this, "Positive", useTransitions);
    }
    else
    {
        VisualStateManager.GoToState(this, "Negative", useTransitions);
    }

    if (IsFocused)
```

```
{  
    VisualStateManager.GoToState(this, "Focused", useTransitions);  
}  
else  
{  
    VisualStateManager.GoToState(this, "Unfocused", useTransitions);  
}  
}
```

如果在控件已处于该状态时将状态名称传递给 `GoToState`，则 `GoToState` 不执行任何操作，因此无需检查控件的当前状态。例如，如果 `Value` 从一个负数更改为另一个负数，则 `Negative` 状态的情节提要不会中断，用户不会看到控件发生更改。

在调用 `GoToState` 时，`VisualStateManager` 使用 `VisualStateGroup` 对象确定要退出的状态。对于在其 `ControlTemplate` 中定义的每个 `VisualStateGroup`，控件始终处于一个状态，并且只有当它进入同一个 `VisualStateGroup` 中的另一个状态时，才会离开一个状态。例如，`NumericUpDown` 控件的 `ControlTemplate` 在一个 `VisualStateGroup` 中定义了 `Positive` 和 `Negative` `VisualState` 对象，在另一个中定义了 `Focused` 和 `Unfocused` `VisualState` 对象。（可以在本主题的[完整示例](#)部分中定义的 `Focused` 和 `Unfocused` `VisualState`）当控件从 `Positive` 状态转换为 `Negative` 状态或进行相反转换时，控件仍保持 `Focused` 或 `Unfocused` 状态。

控件的状态可能会在三种典型情况下进行更改：

- 当 `ControlTemplate` 应用于 `Control` 时。
- 当属性更改时。
- 当事件发生时。

以下示例演示如何在这些情况下更新 `NumericUpDown` 控件的状态。

应在 `OnApplyTemplate` 方法中更新控件的状态，以便在应用 `ControlTemplate` 时，控件可显示正确的状态。以下示例在 `OnApplyTemplate` 中调用 `UpdateStates` 以确保控件处于合适的状态。例如，假设你创建了 `NumericUpDown` 控件，然后将其 `Foreground` 设置为绿色，并将 `Value` 设置为 -5。如果在将 `ControlTemplate` 应用于 `NumericUpDown` 控件时未调用 `UpdateStates`，则控件不处于 `Negative` 状态，并且值为绿色而不是红色。必须调用 `UpdateStates` 以将控件置于 `Negative` 状态。

C#

```
public override void OnApplyTemplate()  
{  
    UpButtonElement = GetTemplateChild("UpButton") as RepeatButton;  
    DownButtonElement = GetTemplateChild("DownButton") as RepeatButton;  
    //TextElement = GetTemplateChild("TextBlock") as TextBlock;
```

```
        UpdateStates(false);  
    }  
}
```

属性更改时，通常需要更新控件的状态。以下示例演示整个 `ValueChangedCallback` 方法。由于在 `Value` 更改时调用了 `ValueChangedCallback`，因此如果 `Value` 从正值更改为负值，或进行相反更改，则方法会调用 `UpdateStates`。在 `Value` 更改，但仍保持正值或负值时调用 `UpdateStates` 是可接受的，因为在这种情况下，控件不会更改状态。

C#

```
private static void ValueChangedCallback(DependencyObject obj,  
    DependencyPropertyChangedEventArgs args)  
{  
    NumericUpDown ctl = (NumericUpDown)obj;  
    int newValue = (int)args.NewValue;  
  
    // Call UpdateStates because the Value might have caused the  
    // control to change ValueStates.  
    ctl.UpdateStates(true);  
  
    // Call OnValueChanged to raise the ValueChanged event.  
    ctl.OnValueChanged(  
        new ValueChangedEventArgs(NumericUpDown.ValueChangedEvent,  
            newValue));  
}  
}
```

在事件发生时，也可能需要更新状态。下面的示例演示 `NumericUpDown` 对 `Control` 调用 `UpdateStates` 以处理 `GotFocus` 事件。

C#

```
protected override void OnGotFocus(RoutedEventArgs e)  
{  
    base.OnGotFocus(e);  
    UpdateStates(true);  
}  
}
```

`VisualStateManager` 可帮助管理控件的状态。使用 `VisualStateManager` 可确保控件在状态之间正确转换。如果遵循本部分中介绍的有关使用 `VisualStateManager` 的建议，则控件的代码会保持可读且可维护。

提供控件协定

可提供控件协定，以便 `ControlTemplate` 作者了解要放入模板中的内容。控件协定具有三个元素：

- 控件逻辑使用的可视元素。
- 控件状态和每种状态所属的组。
- 以可视方式影响控件的公共属性。

创建新 [ControlTemplate](#) 的用户需要了解控件逻辑使用的 [FrameworkElement](#) 对象、每个对象的类型以及其名称。 [ControlTemplate](#) 作者还需要了解控件可以处于的每个可能状态的名称，以及状态所处的 [VisualStyleGroup](#)。

回到 [NumericUpDown](#) 示例，该控件期望 [ControlTemplate](#) 具有以下 [FrameworkElement](#) 对象：

- 一个名为 `UpButton` 的 [RepeatButton](#)。
- 一个名为 `DownButton` 的 [RepeatButton](#)

该控件可以处于以下状态：

- 在 [ValueStates](#) [VisualStyleGroup](#) 中
 - `Positive`
 - `Negative`
- 在 [FocusStates](#) [VisualStyleGroup](#) 中
 - `Focused`
 - `Unfocused`

若要指定控件所需的 [FrameworkElement](#) 对象，可使用 [TemplatePartAttribute](#)，它指定预期元素的名称和类型。 若要指定控件的可能状态，可使用 [TemplateVisualStateAttribute](#)，它指定状态的名称及其所属的 [VisualStyleGroup](#)。 将 [TemplatePartAttribute](#) 和 [TemplateVisualStateAttribute](#) 放置在控件的类定义中。

影响控件外观的任何公共属性也是控件协定的一部分。

以下示例为 [NumericUpDown](#) 控件指定 [FrameworkElement](#) 对象和状态。

C#

```
[TemplatePart(Name = "UpButtonElement", Type = typeof(RepeatButton))]
[TemplatePart(Name = "DownButtonElement", Type = typeof(RepeatButton))]
[TemplateVisualState(Name = "Positive", GroupName = "ValueStates")]
[TemplateVisualState(Name = "Negative", GroupName = "ValueStates")]
[TemplateVisualState(Name = "Focused", GroupName = "FocusedStates")]
[TemplateVisualState(Name = "Unfocused", GroupName = "FocusedStates")]
```

```

public class NumericUpDown : Control
{
    public static readonly DependencyProperty BackgroundProperty;
    public static readonly DependencyProperty BorderBrushProperty;
    public static readonly DependencyProperty BorderThicknessProperty;
    public static readonly DependencyProperty FontFamilyProperty;
    public static readonly DependencyProperty FontSizeProperty;
    public static readonly DependencyProperty FontStretchProperty;
    public static readonly DependencyProperty FontStyleProperty;
    public static readonly DependencyProperty FontWeightProperty;
    public static readonly DependencyProperty ForegroundProperty;
    public static readonly DependencyProperty
HorizontalContentAlignmentProperty;
    public static readonly DependencyProperty PaddingProperty;
    public static readonly DependencyProperty TextAlignProperty;
    public static readonly DependencyProperty TextDecorationsProperty;
    public static readonly DependencyProperty TextWrappingProperty;
    public static readonly DependencyProperty
VerticalContentAlignmentProperty;

    public Brush Background { get; set; }
    public Brush BorderBrush { get; set; }
    public Thickness BorderThickness { get; set; }
    public FontFamily FontFamily { get; set; }
    public double FontSize { get; set; }
    public FontStretch FontStretch { get; set; }
    public FontStyle FontStyle { get; set; }
    public FontWeight FontWeight { get; set; }
    public Brush Foreground { get; set; }
    public HorizontalAlignment HorizontalContentAlignment { get; set; }
    public Thickness Padding { get; set; }
    public TextAlignment TextAlign { get; set; }
    public TextDecorationCollection TextDecorations { get; set; }
    public TextWrapping TextWrapping { get; set; }
    public VerticalAlignment VerticalContentAlignment { get; set; }
}

```

完整的示例

以下示例是 `NumericUpDown` 控件的整个 `ControlTemplate`。

XAML

```

<!--This is the contents of the themes/generic.xaml file.-->
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:VSMCustomControl">

    <Style TargetType="{x:Type local:NumericUpDown}">
        <Setter Property="Template">

```

```

<Setter.Value>
    <ControlTemplate TargetType="local:NumericUpDown">
        <Grid Margin="3"
            Background="{TemplateBinding Background}">

            <VisualStateManager.VisualStateGroups>

                <VisualStateGroup Name="ValueStates">

                    <!--Make the Value property red when it is negative.-->
                    <VisualState Name="Negative">
                        <Storyboard>
                            <ColorAnimation To="Red"
                                Storyboard.TargetName="TextBlock"
                                Storyboard.TargetProperty="(Foreground).(Color)"/>
                        </Storyboard>
                    </VisualState>

                    <!--Return the control to its initial state by
                        return the TextBlock's Foreground to its
                        original color.-->
                    <VisualState Name="Positive"/>
                </VisualStateGroup>

                <VisualStateGroup Name="FocusStates">

                    <!--Add a focus rectangle to highlight the entire control
                        when it has focus.-->
                    <VisualState Name="Focused">
                        <Storyboard>
                            <ObjectAnimationUsingKeyFrames
                                Storyboard.TargetName="FocusVisual"
                                Storyboard.TargetProperty="Visibility" Duration="0">
                                <DiscreteObjectKeyFrame KeyTime="0">
                                    <DiscreteObjectKeyFrame.Value>
                                        <Visibility>Visible</Visibility>
                                    </DiscreteObjectKeyFrame.Value>
                                </DiscreteObjectKeyFrame>
                            </ObjectAnimationUsingKeyFrames>
                        </Storyboard>
                    </VisualState>

                    <!--Return the control to its initial state by
                        hiding the focus rectangle.-->
                    <VisualState Name="Unfocused"/>
                </VisualStateGroup>

            </VisualStateManager.VisualStateGroups>

        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition/>

```

```

        <RowDefinition/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>

    <Border BorderThickness="1" BorderBrush="Gray"
        Margin="7,2,2,2" Grid.RowSpan="2"
        Background="#E0FFFFFF"
        VerticalAlignment="Center"
        HorizontalAlignment="Stretch">
        <!--Bind the TextBlock to the Value property-->
        <TextBlock Name="TextBlock"
            Width="60" TextAlignment="Right" Padding="5"
            Text="{Binding RelativeSource={RelativeSource
FindAncestor,
                AncestorType={x:Type local:NumericUpDown}},

                Path=Value}"/>
    </Border>

    <RepeatButton Content="Up" Margin="2,5,5,0"
        Name="UpButton"
        Grid.Column="1" Grid.Row="0"/>
    <RepeatButton Content="Down" Margin="2,0,5,5"
        Name="DownButton"
        Grid.Column="1" Grid.Row="1"/>

    <Rectangle Name="FocusVisual" Grid.ColumnSpan="2"
Grid.RowSpan="2"
        Stroke="Black" StrokeThickness="1"
        Visibility="Collapsed"/>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</ResourceDictionary>

```

以下示例演示 `NumericUpDown` 的逻辑。

C#

```

using System.Windows;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Input;
using System.Windows.Media;

namespace VSMCustomControl
{

```

```
[TemplatePart(Name = "UpButtonElement", Type = typeof(RepeatButton))]
[TemplatePart(Name = "DownButtonElement", Type = typeof(RepeatButton))]
[TemplateVisualState(Name = "Positive", GroupName = "ValueStates")]
[TemplateVisualState(Name = "Negative", GroupName = "ValueStates")]
[TemplateVisualState(Name = "Focused", GroupName = "FocusedStates")]
[TemplateVisualState(Name = "Unfocused", GroupName = "FocusedStates")]
public class NumericUpDown : Control
{
    public NumericUpDown()
    {
        DefaultStyleKey = typeof(NumericUpDown);
        this.IsTabStop = true;
    }

    public static readonly DependencyProperty ValueProperty =
        DependencyProperty.Register(
            "Value", typeof(int), typeof(NumericUpDown),
            new PropertyMetadata(
                new PropertyChangedCallback(ValueChangedCallback)));
}

public int Value
{
    get
    {
        return (int)GetValue(ValueProperty);
    }

    set
    {
        SetValue(ValueProperty, value);
    }
}

private static void ValueChangedCallback(DependencyObject obj,
    DependencyPropertyChangedEventArgs args)
{
    NumericUpDown ctl = (NumericUpDown)obj;
    int newValue = (int)args.NewValue;

    // Call UpdateStates because the Value might have caused the
    // control to change ValueStates.
    ctl.UpdateStates(true);

    // Call OnValueChanged to raise the ValueChanged event.
    ctl.OnValueChanged(
        new ValueChangedEventArgs(NumericUpDown.ValueChangedEvent,
            newValue));
}

public static readonly RoutedEvent ValueChangedEvent =
    EventManager.RegisterRoutedEvent("ValueChanged",
        RoutingStrategy.Direct,
        typeof(ValueChangedEventHandler),
        typeof(NumericUpDown));
```

```
public event ValueChangedEventHandler ValueChanged
{
    add { AddHandler(ValueChangedEvent, value); }
    remove { RemoveHandler(ValueChangedEvent, value); }
}

protected virtual void OnValueChanged(ValueChangedEventArgs e)
{
    // Raise the ValueChanged event so applications can be alerted
    // when Value changes.
    RaiseEvent(e);
}

private void UpdateStates(bool useTransitions)
{
    if (Value >= 0)
    {
        VisualStateManager.GoToState(this, "Positive",
useTransitions);
    }
    else
    {
        VisualStateManager.GoToState(this, "Negative",
useTransitions);
    }

    if (IsFocused)
    {
        VisualStateManager.GoToState(this, "Focused",
useTransitions);
    }
    else
    {
        VisualStateManager.GoToState(this, "Unfocused",
useTransitions);
    }
}

public override void OnApplyTemplate()
{
    UpButtonElement = GetTemplateChild("UpButton") as RepeatButton;
    DownButtonElement = GetTemplateChild("DownButton") as
RepeatButton;
    //TextElement = GetTemplateChild("TextBlock") as TextBlock;

    UpdateStates(false);
}

private RepeatButton downButtonElement;

private RepeatButton DownButtonElement
{
    get
    {
        return downButtonElement;
    }
}
```

```
        }

        set
        {
            if (downButtonElement != null)
            {
                downButtonElement.Click -=
                    new RoutedEventHandler(downButtonElement_Click);
            }
            downButtonElement = value;

            if (downButtonElement != null)
            {
                downButtonElement.Click +=
                    new RoutedEventHandler(downButtonElement_Click);
            }
        }
    }

void downButtonElement_Click(object sender, RoutedEventArgs e)
{
    Value--;
}

private RepeatButton upButtonElement;

private RepeatButton UpButtonElement
{
    get
    {
        return upButtonElement;
    }

    set
    {
        if (upButtonElement != null)
        {
            upButtonElement.Click -=
                new RoutedEventHandler(upButtonElement_Click);
        }
        upButtonElement = value;

        if (upButtonElement != null)
        {
            upButtonElement.Click +=
                new RoutedEventHandler(upButtonElement_Click);
        }
    }
}

void upButtonElement_Click(object sender, RoutedEventArgs e)
{
    Value++;
}
```

```
protected override void OnMouseLeftButtonDown(MouseEventArgs e)
{
    base.OnMouseLeftButtonDown(e);
    Focus();
}

protected override void OnGotFocus(RoutedEventArgs e)
{
    base.OnGotFocus(e);
    UpdateStates(true);
}

protected override void OnLostFocus(RoutedEventArgs e)
{
    base.OnLostFocus(e);
    UpdateStates(true);
}

public delegate void ValueChangedEventHandler(object sender,
ValueChangedEventArgs e);

public class ValueChangedEventArgs : RoutedEventArgs
{
    private int _value;

    public ValueChangedEventArgs(RoutedEventArgs id, int num)
    {
        _value = num;
        RoutedEvent = id;
    }

    public int Value
    {
        get { return _value; }
    }
}
}
```

另请参阅

- [创建控件模板](#)
- [控件自定义](#)

可样式化控件的设计准则

项目 · 2022/09/28

本文档概述在设计可方便地样式化和模板化的控件时需要考虑的一组最佳做法。在为内置的 WPF 控件集处理主题控件样式时，我们通过大量试验和错误总结出了这组最佳做法。我们已经认识到，成功的样式设置不只是设计完善的对象模型的功能，也是样式本身的功能。本文档面向控件作者，而不是样式作者。

术语

“样式设置和模板化”是一组技术，控件作者可以通过该组技术将控件的可视化特性延迟到控件的样式和模板。这组技术包括：

- 样式（包括属性资源库、触发器和演示图板）。
- 资源。
- 控件模板。
- 数据模板。

有关样式设置和模板化简介，请参阅[样式设置和模板化](#)。

准备工作：了解控件

在开始阅读这些准则之前，请务必了解并定义了控件的常见用法。样式设置公开一组通常不受约束的可能性。旨在由许多开发人员在许多应用程序中广泛使用的控件面临着如下挑战：可以使用样式设置对控件的可视化外观进行广泛更改。实际上，带样式的控件甚至可能并非控件作者的本意。由于样式设置在本质上可以提供无限的灵活性，因此可以使用“常见用法”这一概念来帮助你限制自己的决定。

若要了解控件的常见用法，最好考虑控件的价值主张。你的控件能够在表中提供哪些无法由其他控件提供的内容？常见用法并不表示任何特定的可视化外观，而是表示控件的基本原理和一组有关其用法的合理预期。了解到这一点，就可以对控件在一般情况下的撰写模型和样式定义行为进行一些假设。例如，对于 [ComboBox](#)，如果你了解其常见用法，并不意味着非常清楚特定的 [ComboBox](#) 是否有圆角，但是将由此清楚 [ComboBox](#) 可能需要一个弹出窗口和某种用来切换其开关状态的方法。

一般性指导

- **不严格实施模板协定。** 控件的模板协定可能包含元素、命令、绑定和触发器，甚至还可以包含必需的或为了使控件正常工作而应当使用的属性设置。
 - 最大限度地减少协定。
 - 围绕如下预期进行设计：在设计时（即，在使用设计工具时），控件模板通常处于不完整状态。WPF 不提供“正在撰写”状态的基础结构，因此，控件必须围绕这样的状态可能有效这一预期来生成。
 - 在没有遵循模板协定的任何方面时，不引发异常。按照这一原则，当面板的子级太多或太少时，面板不应引发异常。
- **将外围功能分解成模板帮助程序元素。** 每个控件都应当将重点放在其核心功能和真正的价值主张上，而且每个控件都应当由控件的常见用法定义。为此，请使用模板中的撰写和帮助程序元素实现外围行为和可视化（即，那些不构成控件核心功能的行为和可视化）。帮助程序元素分为三类：
 - **独立**帮助程序类型是以“匿名方式”用在模板中的可重用的公共控件或基元，这意味着帮助程序元素和带样式的控件无法互相识别。在技术上，任何元素都可以是匿名类型，但是在此上下文中，该术语描述了那些封装专用功能以实现目标方案的类型。
 - **基于类型的**帮助程序元素是封装专用功能的新类型。通常，这些元素在设计上比通用控件或基元的功能范围要窄。与独立帮助程序元素不同的是，基于类型的帮助程序元素能够识别它们的使用上下文，而且通常必须与包含它们所属模板的控件共享数据。
 - **命名的**帮助程序元素是控件应当能够在其模板中根据名称找到的常用控件或基元。这些元素在模板中具有一个已知的名称，这使得控件能够找到这些元素并以编程方式与之交互。在任何模板中，都只能有一个具有给定名称的元素。

下表显示了由目前的控件样式使用的部分帮助程序元素列表：

元素	类型	使用者
ContentPresenter	基于类型的	Button、CheckBox、RadioButton、Frame 等（所有 ContentControl 类型）
ItemsPresenter	基于类型的	ListBox、ComboBox、Menu 等（所有 ItemsControl 类型）
ToolBarOverflowPanel	名为	ToolBar
Popup	独立	ComboBox、ToolBar、Menu、ToolTip 等等
RepeatButton	名为	Slider、ScrollBar 等等

元素	类型	使用者
ScrollBar	名为	ScrollViewer
ScrollViewer	独立	ListBox、ComboBox、Menu、Frame 等等
TabPanel	独立	TabControl
TextBox	名为	ComboBox
TickBar	基于类 型的	Slider

- **最大限度地减少帮助程序元素所必需的、特定于用户的绑定或属性设置。** 通常，帮助程序元素需要某些绑定或属性设置才能在控件模板中正确工作。帮助程序元素和模板化控件应当尽可能多地生成这些设置。在设置属性或者建立绑定时，注意不要重写由用户设置的值。具体的最佳做法如下所示：
 - 命名的帮助程序元素应当由父级标识，而且父级应当针对帮助程序元素建立任何必需的设置。
 - 基于类型的帮助程序元素应当直接针对自身建立任何必需的设置。这样做可能需要帮助程序元素查找它在使用时的信息上下文，包括其 `TemplatedParent`（它在使用时的模板的控件类型）。例如，当用于 `ContentControl` 派生类型时，`ContentPresenter` 会自动将其 `TemplatedParent` 的 `Content` 属性绑定到其 `Content` 属性。
 - 独立帮助程序元素不能按这种方式进行优化，这是因为按照定义，帮助程序元素和父级不能相互识别。
- **使用 Name 属性标记模板中的元素。** 如果控件需要在样式中查找某个元素才能以编程方式访问它，则该控件应当使用 `Name` 属性和 `FindName` 范例来进行查找。控件不应在未找到所需元素时引发异常，而是应在不提示的情况下禁用需要该元素的功能。
- **使用最佳做法来表示样式中的控件状态和行为。** 下面按顺序列出了用来表示样式中的控件状态更改和行为的最佳做法。应使用列表上的第一项来实现你的方案。
 1. 属性绑定。示例：`ComboBox.IsDropDownOpen` 和 `ToggleButton.IsChecked` 之间的绑定。
 2. 触发的属性更改或属性动画。示例：`Button` 的悬停状态。
 3. 命令。示例：`ScrollBar` 中的 `LineUpCommand` / `LineDownCommand`。
 4. 独立帮助程序元素。示例：`TabControl` 中的 `TabPanel`。

5. 基于类型的帮助程序类型。示例：[Button](#) 中的 [ContentPresenter](#)，[Slider](#) 中的 [TickBar](#)。

6. 命名的帮助程序元素。示例：[ComboBox](#) 中的 [TextBox](#)。

7. 命名的帮助程序类型中的冒泡事件。如果侦听样式元素中的冒泡事件，应当要求生成该事件的元素能够进行唯一标识。示例：[ToolBar](#) 中的 [Thumb](#)。

8. 自定义 [OnRender](#) 行为。示例：[Button](#) 中的 [ButtonChrome](#)。

- **慎用样式触发器（与模板触发器相对）。** 影响模板中元素上的属性的触发器必须在模板中声明。影响控件上的属性的触发器（没有 [TargetName](#)）可以在样式中声明，除非你知道更改模板还可能会损坏触发器。
- **与现有的样式设置模式保持一致。** 一个问题常常有多种解决办法。注意尽可能与现有的控件样式设置模式保持一致。这对于派生自同一基类型（例如，[ContentControl](#)、[ItemsControl](#)、[RangeBase](#) 等）的控件尤其重要。
- **在不重新模板化的情况下公开属性来启用常见自定义项方案。** WPF 不支持可插入/可自定义的部件，因此控件用户只能使用两种自定义方法：直接设置属性或者使用样式设置属性。请记住，比较合适的做法是，设置数量有限的属性，使其面向极其常见的高优先级自定义项方案，否则的话，这些方案需要重新模板化。下面是有关何时以及如何启用自定义项方案的最佳方法：
 - 极其常见的自定义项应当作为属性在控件上公开并由模板使用。
 - 不太常见（尽管并非极少见）的自定义项应当作为附加属性公开并由模板使用。
 - 需要对已知但是极少见的自定义项重新模板化，这一点也是可接受的。

主题注意事项

- **主题样式应尝试在所有主题中具有一致的属性语义，但不能保证能够实现这一点。** 作为控件文档的一部分，控件应当具有一个描述其属性语义（即控件属性的“含义”）的文档。例如，[ComboBox](#) 控件应当定义 [Background](#) 属性在 [ComboBox](#) 中的含义。控件的默认样式应当尝试遵循在其文档中的所有主题中定义的语义。另一方面，控件用户应当注意属性语义可能因主题而异。在某些情况下，给定的属性在由特定主题所需的可视化约束下可能无法表示。（例如，对于许多控件来说，传统主题没有可以向其应用 [Thickness](#) 的边框。）
- **主题样式不需要在所有主题中具有一致的触发器语义。** 由控件样式通过触发器或动画公开的行为可能因主题而异。控件用户应当注意到，控件不必使用同一个机制在所有主题中实现特定的行为。例如，一个主题可以使用动画来表示悬停行为，而另一个主题则可以使用触发器。这可能会导致自定义控件上的行为保留出现不一致。

(更改后台属性（例如，如果使用触发器表示该状态）可能会影响控件的悬停状态。但是，如果使用动画实现悬停状态，则更改为背景可能会无法弥补地中断动画，因此状态转换。）

- **主题样式不需要在所有主题中具有一致的“布局”语义。** 例如，默认的样式不需要保证控件将在所有主题中占用同样的大小，也不需要保证控件将在所有主题中具有同样的内容边距/空白。

请参阅

- [样式设置和模板化](#)
- [控件创作概述](#)

装饰器

项目 • 2023/02/06

本部分提供有关装饰器和 Windows Presentation Foundation (WPF) 装饰器框架的信息。

本节内容

[装饰器概述](#)

[操作指南主题](#)

参考

[AdornedElementPlaceholder](#)

[Adorner](#)

[AdornerDecorator](#)

[AdornerHitTestResult](#)

[AdornerLayer](#)

相关章节

装饰器概述

项目 · 2022/09/27

装饰器是一种特殊类型的 [FrameworkElement](#)，用于向用户提供视觉提示。 装饰器有很多用途，可用来向元素添加功能句柄，或者提供有关某个控件的状态信息。

关于装饰器

[Adorner](#) 是绑定到 [UIElement](#) 的自定义 [FrameworkElement](#)。 装饰器在 [AdornerLayer](#) 中呈现，它是始终位于装饰元素或装饰元素集合之上的呈现表面。 装饰器的呈现独立于装饰器绑定到的 [UIElement](#) 的呈现。 装饰器通常使用位于装饰元素左上部的标准 2D 坐标原点，相对于其绑定到的元素进行定位。

装饰器的常见应用包括：

- 向 [UIElement](#) 添加功能句柄，使用户能够以某种方式操作元素（调整大小、旋转、重新定位等）。
- 提供视觉反馈以指示各种状态，或者响应各种事件。
- 在 [UIElement](#) 上叠加视觉装饰。
- 以视觉方式遮盖或覆盖 [UIElement](#) 的一部分或全部。

Windows Presentation Foundation (WPF) 为装饰视觉元素提供了一个基本框架。 下表列出了装饰对象时使用的主要类型及其用途。 下面是几个用法示例：

类	说明
Adorner	一个抽象基类，从中继承所有的具体装饰器实现。
AdornerLayer	一个类，表示一个或多个装饰元素的装饰器的呈现层。
AdornerDecorator	一个类，使装饰器层与元素集合相关联。

实现自定义装饰器

Windows Presentation Foundation (WPF) 提供的装饰器框架主要用于支持创建自定义装饰器。 通过实现从抽象 [Adorner](#) 类继承的类来创建自定义装饰器。

① 备注

[Adorner](#) 的父级是呈现 [Adorner](#) 的 [AdornerLayer](#)，而不是被装饰的元素。

下面的示例展示了实现简单装饰器的类。示例装饰器只是用圆圈装饰 [UIElement](#) 的角。

C#

```
// Adorners must subclass the abstract base class Adorner.
public class SimpleCircleAdorner : Adorner
{
    // Be sure to call the base class constructor.
    public SimpleCircleAdorner(UIElement adornedElement)
        : base(adornedElement)
    {
    }

    // A common way to implement an adorner's rendering behavior is to
    // override the OnRender
    // method, which is called by the layout system as part of a rendering
    // pass.
    protected override void OnRender(DrawingContext drawingContext)
    {
        Rect adornedElementRect = new Rect(this.AdornedElement.DesiredSize);

        // Some arbitrary drawing implements.
        SolidColorBrush renderBrush = new SolidColorBrush(Colors.Green);
        renderBrush.Opacity = 0.2;
        Pen renderPen = new Pen(new SolidColorBrush(Colors.Navy), 1.5);
        double renderRadius = 5.0;

        // Draw a circle at each corner.
        drawingContext.DrawEllipse(renderBrush, renderPen,
adornedElementRect.TopLeft, renderRadius, renderRadius);
        drawingContext.DrawEllipse(renderBrush, renderPen,
adornedElementRect.TopRight, renderRadius, renderRadius);
        drawingContext.DrawEllipse(renderBrush, renderPen,
adornedElementRect.BottomLeft, renderRadius, renderRadius);
        drawingContext.DrawEllipse(renderBrush, renderPen,
adornedElementRect.BottomRight, renderRadius, renderRadius);
    }
}
```

下图显示了应用于 [TextBox](#) 的 SimpleCircleAdorner：



装饰器的呈现行为

请务必注意，装饰器不包括任何继承呈现行为，确保装饰器呈现是装饰器实施者的责任。实现呈现行为的一种常见方法是覆盖 [OnRender](#) 方法并使用一个或多个 [DrawingContext](#) 对象根据需要呈现装饰器的视觉效果（如上例所示）。

① 备注

放置在装饰器层中的任何内容将呈现在设置的其他任何样式的顶部。换言之，装饰器始终以可见的方式位于顶部，无法使用 z 顺序重写。

事件和命中测试

装饰器像任何其他 [FrameworkElement](#) 一样接收输入事件。因为装饰器的 z 顺序总是高于它所装饰的元素，所以装饰器接收可能用于底层装饰元素的输入事件（例如 [Drop](#) 或 [MouseMove](#)）。装饰器可以侦听某些输入事件，并通过重新引发这些事件将它们传递到基础装饰元素。

要对装饰器下的元素启用直通命中测试，请将装饰器上的命中测试 [IsHitTestVisible](#) 属性设置为 `false`。有关命中测试的详细信息，请参阅[视觉层中的命中测试](#)。

装饰单个 UIElement

若要将装饰器绑定到特定的 [UIElement](#)，请按照以下步骤操作：

1. 调用静态方法 [GetAdornerLayer](#) 以获取要装饰的 [UIElement](#) 的 [AdornerLayer](#) 对象。
[GetAdornerLayer](#) 从指定的 [UIElement](#) 开始向上遍历可视化树，并返回它找到的第一个装饰层。（如果未发现装饰器层，该方法将返回 `null`。）
2. 调用 [Add](#) 方法将装饰器绑定到目标 [UIElement](#)。

以下示例将 [SimpleCircleAdorner](#)（如上所示）绑定到名为 `myTextBox` 的 [TextBox](#)：

C#

```
myAdornerLayer = AdornerLayer.GetAdornerLayer(myTextBox);
myAdornerLayer.Add(new SimpleCircleAdorner(myTextBox));
```

① 备注

目前不支持使用 Extensible Application Markup Language (XAML) 将装饰器绑定到另一个元素。

装饰面板的子级

若要将装饰器绑定到 [Panel](#) 的子元素，请按照以下步骤操作：

1. 调用 `static` 方法 [GetAdornerLayer](#) 为要装饰其子元素的元素查找装饰层。

2. 依次枚举父元素的子级并调用 [Add](#) 方法，以将装饰器绑定到每个子元素。

以下示例将 [SimpleCircleAdorner](#) (如上所示) 绑定到名为 `myStackPanel` 的 [StackPanel](#) 的子级：

C#

```
foreach (UIElement toAdorn in myStackPanel.Children)
    myAdornerLayer.Add(new SimpleCircleAdorner(toAdorn));
```

另请参阅

- [AdornerHitTestResult](#)
- [WPF 中的形状和基本图形概述](#)
- [使用图像、图形和视觉对象进行绘制](#)
- [Drawing 对象概述](#)
- [操作指南主题](#)

装饰器帮助主题

项目 • 2023/02/06

以下示例演示如何使用 Windows Presentation Foundation (WPF) 装饰器框架完成常见任务。

本节内容

[实现装饰器](#)

[将装饰器绑定到元素](#)

[装饰面板的子元素](#)

[从元素中删除装饰器](#)

[从元素中删除所有装饰器](#)

参考

[AdornedElementPlaceholder](#)

[Adorner](#)

[AdornerDecorator](#)

[AdornerHitTestResult](#)

[AdornerLayer](#)

相关章节

如何：实现装饰器

项目 • 2023/02/06

此示例演示了一个最小装饰器实现。

实现者须知

请务必注意，装饰器不包括任何继承呈现行为，确保装饰器呈现是装饰器实施者的责任。实现呈现行为的常用方法是重写 [OnRender](#) 方法，并使用一个或多个 [DrawingContext](#) 对象根据需要呈现装饰器的视觉效果（如以下示例所示）。

示例

描述

可通过实现从抽象 [Adorner](#) 类中继承的类创建自定义装饰器。示例装饰器通过替代 [OnRender](#) 方法使用圆圈简单装饰 [UIElement](#) 的角。

代码

C#

```
// Adorners must subclass the abstract base class Adorner.
public class SimpleCircleAdorner : Adorner
{
    // Be sure to call the base class constructor.
    public SimpleCircleAdorner(UIElement adornedElement)
        : base(adornedElement)
    {
    }

    // A common way to implement an adorner's rendering behavior is to
    // override the OnRender
    // method, which is called by the layout system as part of a rendering
    // pass.
    protected override void OnRender(DrawingContext drawingContext)
    {
        Rect adornedElementRect = new Rect(this.AdornedElement.DesiredSize);

        // Some arbitrary drawing implements.
        SolidColorBrush renderBrush = new SolidColorBrush(Colors.Green);
        renderBrush.Opacity = 0.2;
        Pen renderPen = new Pen(new SolidColorBrush(Colors.Navy), 1.5);
        double renderRadius = 5.0;
    }
}
```

```
// Draw a circle at each corner.  
drawingContext.DrawEllipse(renderBrush, renderPen,  
adornedElementRect.TopLeft, renderRadius, renderRadius);  
    drawingContext.DrawEllipse(renderBrush, renderPen,  
adornedElementRect.TopRight, renderRadius, renderRadius);  
    drawingContext.DrawEllipse(renderBrush, renderPen,  
adornedElementRect.BottomLeft, renderRadius, renderRadius);  
    drawingContext.DrawEllipse(renderBrush, renderPen,  
adornedElementRect.BottomRight, renderRadius, renderRadius);  
}  
}
```

另请参阅

- [装饰器概述](#)

如何：将装饰器绑定到元素

项目 • 2023/02/06

此示例演示如何以编程方式将装饰器绑定到指定的 [UIElement](#)。

示例

若要将装饰器绑定到特定的 [UIElement](#)，请按照以下步骤操作：

1. 调用 `static` 方法 [GetAdornerLayer](#) 以获取要装饰的 [UIElement](#) 的 [AdornerLayer](#) 对象。 [GetAdornerLayer](#) 从指定的 [UIElement](#) 开始沿着可视化树向上行进，返回它发现的第一个装饰器层。（如果未发现装饰器层，该方法将返回 `null`。）
2. 调用 [Add](#) 方法将装饰器绑定到目标 [UIElement](#)。

以下示例将 [SimpleCircleAdorner](#)（如上所示）绑定到名为 `myTextBox` 的 [TextBox](#)。

C#

```
myAdornerLayer = AdornerLayer.GetAdornerLayer(myTextBox);
myAdornerLayer.Add(new SimpleCircleAdorner(myTextBox));
```

① 备注

目前不支持使用 Extensible Application Markup Language (XAML) 将装饰器绑定到另一个元素。

另请参阅

- [装饰器概述](#)

如何：装饰面板的子级

项目 • 2023/02/06

此示例演示如何以编程方式将装饰器绑定到指定 [Panel](#) 的子元素。

示例

若要将装饰器绑定到 [Panel](#) 的子元素，请按照以下步骤操作：

1. 声明一个新 [AdornerLayer](#) 对象并调用 `static GetAdornerLayer` 方法来查找要装饰其子级的元素的装饰器层。
2. 依次枚举父元素的子级并调用 [Add](#) 方法，以将装饰器绑定到每个子元素。

以下示例将 [SimpleCircleAdorner](#)（如上所示）绑定到名为 `myStackPanel` 的 [StackPanel](#) 的子级。

C#

```
foreach (UIElement toAdorn in myStackPanel.Children)  
    myAdornerLayer.Add(new SimpleCircleAdorner(toAdorn));
```

① 备注

目前不支持使用 Extensible Application Markup Language (XAML) 将装饰器绑定到另一个元素。

另请参阅

- [装饰器概述](#)

如何：从元素移除装饰器

项目 • 2023/02/06

此示例演示如何以编程方式从指定的 [UIElement](#) 中移除特定装饰器。

检索 UIElement 上的装饰器

此详细代码示例移除由 [GetAdorners](#) 返回的装饰器数组中的第一个装饰器。此示例恰好检索名为 myTextBox 的 [UIElement](#) 上的装饰器。如果在对 [GetAdorners](#) 的调用中指定的元素没有装饰器，则返回 `null`。此代码显式检查 NULL 数组，最适合 NULL 数组预期相对常见的应用程序。

C#

```
Adorner[] toRemoveArray = myAdornerLayer.GetAdorners(myTextBox);
Adorner toRemove;
if (toRemoveArray != null)
{
    toRemove = toRemoveArray[0];
    myAdornerLayer.Remove(toRemove);
}
```

示例

此精简代码示例在功能上等同于上面所示的详细示例。此代码不会显式检查 NULL 数组，因此可能会引发 [NullReferenceException](#) 异常。此代码最适合 NULL 数组预期很少见的应用程序。

C#

```
try { myAdornerLayer.Remove((myAdornerLayer.GetAdorners(myTextBox))[0]); }
catch { }
```

另请参阅

- [装饰器概述](#)

如何：从元素移除所有装饰器

项目 • 2023/02/06

此示例演示如何以编程方式从指定的 [UIElement](#) 中移除所有装饰器。

检索 [UIElement](#) 上的装饰器

此详细代码示例移除由 [GetAdorners](#) 返回的装饰器数组中的所有装饰器。此示例恰好检索名为 myTextBox 的 [UIElement](#) 上的装饰器。如果在对 [GetAdorners](#) 的调用中指定的元素没有装饰器，则返回 `null`。此代码显式检查 NULL 数组，最适合预期 NULL 数组相对常见的应用程序。

C#

```
Adorner[] toRemoveArray = myAdornerLayer.GetAdorners(myTextBox);
if (toRemoveArray != null)
{
    for (int x = 0; x < toRemoveArray.Length; x++)
    {
        myAdornerLayer.Remove(toRemoveArray[x]);
    }
}
```

代码示例

此精简代码示例在功能上等效于上面所示的详细示例。此代码不会显式检查 NULL 数组，因此可能会引发 [NullReferenceException](#) 异常。此代码最适合预期 NULL 数组很少见的应用程序。

C#

```
try { foreach (Adorner toRemove in myAdornerLayer.GetAdorners(myTextBox))
myAdornerLayer.Remove(toRemove); } catch { }
```

另请参阅

- [装饰器概述](#)

Control 样式和模板

项目 • 2023/02/06

Windows Presentation Foundation (WPF) 中的控件具有 [ControlTemplate](#)，它包含该控件的可视化树。可以通过修改某个控件的 [ControlTemplate](#) 来更改该控件的结构和外观。不能仅替换控件的可视化树的一部分；若要更改控件的可视化树，必须将该控件的 [Template](#) 属性设置为新的完整 [ControlTemplate](#)。

桌面主题确定使用的资源字典。Visual Studio 安装中包含桌面主题的资源字典。包含主题的文件夹通常位于 C:\Program Files (x86)\Microsoft Visual Studio\2019\<visual studio edition>\DesignTools\SystemThemes\wpf，其中 <visual studio edition> 表示 Visual Studio 的版本。

下表描述了资源字典文件名及其相应的桌面主题。

主题文件	桌面主题
Classic.xaml	Windows XP 操作系统上的经典 Windows 外观 (Windows 95、Windows 98 和 Windows 2000)。
Luna.NormalColor.xaml	Windows XP 上的默认蓝色主题。
Luna.Homestead.xaml	Windows XP 上的橄榄色主题。
Luna.Metallic.xaml	Windows XP 上的银色主题。
Royale.NormalColor.xaml	Windows XP Media Center Edition 操作系统上的默认主题。
Aero.NormalColor.xaml	Windows Vista 操作系统上的默认主题。

本节内容

- [Button 样式和模板](#)
- [Calendar 样式和模板](#)
- [CheckBox 样式和模板](#)
- [ComboBox 样式和模板](#)
- [ContextMenu 样式和模板](#)
- [DataGrid 样式和模板](#)
- [DatePicker 样式和模板](#)
- [DocumentViewer 样式和模板](#)
- [Expander 样式和模板](#)
- [Frame 样式和模板](#)
- [GroupBox 样式和模板](#)

[Label 样式和模板](#)

[ListBox 样式和模板](#)

[ListView 样式和模板](#)

[Menu 样式和模板](#)

[NavigationWindow 样式和模板](#)

[PasswordBox 样式和模板](#)

[ProgressBar 样式和模板](#)

[RadioButton 样式和模板](#)

[RepeatButton 样式和模板](#)

[ScrollBar 样式和模板](#)

[ScrollViewer 样式和模板](#)

[Slider 样式和模板](#)

[StatusBar 样式和模板](#)

[TabControl 样式和模板](#)

[TextBox 样式和模板](#)

[Thumb 样式和模板](#)

[ToggleButton 样式和模板](#)

[ToolBar 样式和模板](#)

[ToolTip 样式和模板](#)

[TreeView 样式和模板](#)

[Window 样式和模板](#)

参考

[System.Windows.Controls](#)

[ControlTemplate](#)

相关章节

[控件创作概述](#)

[样式设置和模板化](#)

Button 样式和模板

项目 • 2023/02/06

本主题介绍 [Button 控件](#) 的样式和模板。可以修改默认 [ControlTemplate](#)，为控件提供独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

Button 部件

[Button](#) 控件没有任何命名的部件。

Button 状态

下表列出了 [Button](#) 控件的视觉对象状态。

VisualState 名称 名称	VisualStateGroup	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在控件上方。
已按下	CommonStates	已按下控件。
已禁用	CommonStates	已禁用控件。
已设定焦点	FocusStates	控件有焦点。
失去焦点	FocusStates	控件没有焦点。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性为 <code>true</code> ，控件具有焦点。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性为 <code>true</code> ，控件不具有焦点。

Button ControlTemplate 示例

以下示例演示如何定义 [Button](#) 控件的 [ControlTemplate](#)。

XAML

```

<!-- FocusVisual -->

<Style x:Key="ButtonFocusVisual">
    <Setter Property="Control.Template">
        <Setter.Value>
            <ControlTemplate>
                <Border>
                    <Rectangle Margin="2"
                               StrokeThickness="1"
                               Stroke="#60000000"
                               StrokeDashArray="1 2" />
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<!-- Button -->
<Style TargetType="Button">
    <Setter Property="SnapsToDevicePixels"
           Value="true" />
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="FocusVisualStyle"
           Value="{StaticResource ButtonFocusVisual}" />
    <Setter Property="MinHeight"
           Value="23" />
    <Setter Property="MinWidth"
           Value="75" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="Button">
                <Border TextBlock.Foreground="{TemplateBinding Foreground}"
                        x:Name="Border"
                        CornerRadius="2"
                        BorderThickness="1">
                    <Border.BorderBrush>
                        <LinearGradientBrush StartPoint="0,0"
                                            EndPoint="0,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStopCollection>
                                    <GradientStop Color="{DynamicResource BorderLightColor}"
                                                  Offset="0.0" />
                                    <GradientStop Color="{DynamicResource BorderDarkColor}"
                                                  Offset="1.0" />
                                </GradientStopCollection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Border.BorderBrush>
                    <Border.Background>
                        <LinearGradientBrush EndPoint="0.5,1"
                                            StartPoint="0.5,0">
                            <GradientStop Color="{DynamicResource ControlLightColor}" />
                        </LinearGradientBrush>
                    </Border.Background>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```

                Offset="0" />
            <GradientStop Color="{DynamicResource ControlMediumColor}" Offset="1" />
        </LinearGradientBrush>
    </Border.Background>
    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="CommonStates">
            <VisualStateGroup.Transitions>
                <VisualTransition GeneratedDuration="0:0:0.5" />
                <VisualTransition GeneratedDuration="0"
                    To="Pressed" />
            </VisualStateGroup.Transitions>
            <VisualState x:Name="Normal" />
            <VisualState x:Name="MouseOver">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="

(Panel.Background).(GradientBrush.GradientStops)[1].(GradientStop.Color)" Storyboard.TargetName="Border">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource

ControlMouseOverColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
            <VisualState x:Name="Pressed">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="

(Panel.Background).(GradientBrush.GradientStops)[1].(GradientStop.Color)" Storyboard.TargetName="Border">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource

ControlPressedColor}" />
                    </ColorAnimationUsingKeyFrames>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="

(Border.BorderBrush).(GradientBrush.GradientStops)[0].(GradientStop.Color)" Storyboard.TargetName="Border">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource

PressedBorderDarkColor}" />
                    </ColorAnimationUsingKeyFrames>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="

(Border.BorderBrush).(GradientBrush.GradientStops)[1].(GradientStop.Color)" Storyboard.TargetName="Border">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource

PressedBorderLightColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>

```

```

        </Storyboard>
    </VisualState>
    <VisualState x:Name="Disabled">
        <Storyboard>
            <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="

(Panel.Background). (GradientBrush.GradientStops)[1].(GradientStop.Color)">

Storyboard.TargetName="Border">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource

DisabledControlDarkColor}" />
            </ColorAnimationUsingKeyFrames>
            <ColorAnimationUsingKeyFrames
                Storyboard.TargetProperty="(TextBlock.Foreground). (SolidColorBrush.Color)">

Storyboard.TargetName="Border">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource

DisabledForegroundColor}" />
            </ColorAnimationUsingKeyFrames>
            <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="

(Border.BorderBrush). (GradientBrush.GradientStops)[1].(GradientStop.Color)">

Storyboard.TargetName="Border">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource

DisabledBorderDarkColor}" />
            </ColorAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<ContentPresenter Margin="2"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"
    RecognizesAccessKey="True" />
</Border>
<ControlTemplate.Triggers>
    <Trigger Property="IsDefault"
        Value="true">

        <Setter TargetName="Border"
            Property="BorderBrush">
            <Setter.Value>
                <LinearGradientBrush StartPoint="0,0"
                    EndPoint="0,1">
                    <GradientBrush.GradientStops>
                        <GradientStopCollection>
                            <GradientStop Color="{DynamicResource

DefaultBorderBrushLightBrush}"
                                Offset="0.0" />
                            <GradientStop Color="{DynamicResource

```

```

DefaultBorderBrushDarkColor}" 
    Offset="1.0" />
  </GradientStopCollection>
  </GradientBrush.GradientStops>
</LinearGradientBrush>

  </Setter.Value>
</Setter>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

```

```
<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

Calendar 样式和模板

项目 • 2023/02/06

本主题介绍 [Calendar](#) 控件的样式和模板。 可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。 有关详细信息，请参阅[为控件创建模板](#)。

日历部件

下表列出了 [Calendar](#) 控件的已命名部件。

组成部分	类型	描述
PART_CalendarItem	CalendarItem	Calendar 上当前显示的月份或年份。
PART_Root	Panel	包含 CalendarItem 的面板。

日历状态

下表列出了 [Calendar](#) 控件的视觉对象状态。

VisualState 名称	VisualStateGroup 名称	描述
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

CalendarItem 部件

下表列出了 [CalendarItem](#) 控件的已命名部件。

组成部分	类型	描述
PART_Root	FrameworkElement	控件的根。
PART_PreviousButton	Button	单击日历时显示上一页的按钮。
PART_NextButton	Button	单击日历时显示下一页的按钮。

组成部分	类型	描述
PART_HeaderButton	Button	允许在月模式、年份模式和十年模式之间切换的按钮。
PART_MonthView	Grid	在月份模式下托管内容。
PART_YearView	Grid	在年份或十年模式下托管内容。
PART_DisabledVisual	FrameworkElement	禁用状态的覆盖。
DayTitleTemplate	DataTemplate	描述视觉对象结构的 DataTemplate。

CalendarItem 状态

下表列出了 [CalendarItem](#) 控件的视觉对象状态。

VisualState 名称 名称	VisualStateGroup	描述
正常状态	CommonStates	默认状态。
已禁用状态	CommonStates	<code>IsEnabled</code> 属性为 <code>false</code> 时日历的状态。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

CalendarDayButton 部件

[CalendarDayButton](#) 控件没有任何已命名的部件。

CalendarDayButton 状态

下表列出了 [CalendarDayButton](#) 控件的视觉对象状态。

VisualState 名称	VisualStateGroup 名称	描述
普通	CommonStates	默认状态。
已禁用	CommonStates	禁用 CalendarDayButton 。
MouseOver	CommonStates	鼠标指针悬停在 CalendarDayButton 上方。
已按下	CommonStates	按下 CalendarDayButton 。
选定	SelectionStates	按钮处于选中状态。
未选定	SelectionStates	按钮处于未选中状态。
CalendarButtonFocused	CalendarButtonFocusStates	按钮具有焦点。
CalendarButtonUnfocused	CalendarButtonFocusStates	按钮没有焦点。
已设定焦点	FocusStates	按钮具有焦点。
失去焦点	FocusStates	按钮没有焦点。
活动	ActiveStates	按钮处于活动状态。
非活动	ActiveStates	按钮处于非活动状态。
RegularDay	DayStates	按钮不表示 DateTime.Today 。
今天	DayStates	按钮表示 DateTime.Today 。
NormalDay	BlackoutDayStates	按钮表示可以选择的一天。
BlackoutDay	BlackoutDayStates	按钮表示无法选择的一天。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

CalendarButton 部件

[CalendarButton](#) 控件没有任何已命名的部件。

CalendarButton 状态

下表列出了 [CalendarButton](#) 控件的视觉对象状态。

VisualState 名称	VisualStateGroup 名称	描述
普通	CommonStates	默认状态。
已禁用	CommonStates	禁用 CalendarButton 。
MouseOver	CommonStates	鼠标指针悬停在 CalendarButton 上方。
已按下	CommonStates	按下 CalendarButton 。
选定	SelectionStates	按钮处于选中状态。
未选定	SelectionStates	按钮处于未选中状态。
CalendarButtonFocused	CalendarButtonFocusStates	按钮具有焦点。
CalendarButtonUnfocused	CalendarButtonFocusStates	按钮没有焦点。
已设定焦点	FocusStates	按钮具有焦点。
失去焦点	FocusStates	按钮没有焦点。
活动	ActiveStates	按钮处于活动状态。
非活动	ActiveStates	按钮处于非活动状态。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

日历 ControlTemplate 示例

下例演示如何定义 [Calendar](#) 控件的 [ControlTemplate](#) 及关联类型。

XAML

```

<!--Style for the days of a month.-->
<Style TargetType="CalendarDayButton"
      x:Key="CalendarDayButtonStyle">
    <Setter Property="MinWidth"
           Value="5" />
    <Setter Property="MinHeight"
           Value="5" />
    <Setter Property="FontSize"
           Value="10" />
    <Setter Property="HorizontalContentAlignment"
           Value="Center" />
    <Setter Property="VerticalContentAlignment"
           Value="Center" />
    <Setter Property="Template">
      <Setter.Value>
        <ControlTemplate TargetType="CalendarDayButton">
          <Grid>
            <VisualStateManager.VisualStateGroups>
              <VisualStateGroup Name="CommonStates">
                <VisualStateGroup.Transitions>
                  <VisualTransition GeneratedDuration="0:0:0.1" />
                </VisualStateGroup.Transitions>
                <VisualState Name="Normal" />
                <VisualState Name="MouseOver">
                  <Storyboard>
                    <DoubleAnimation
                      Storyboard.TargetName="HighlightBackground"
                      Storyboard.TargetProperty="Opacity"
                      To="0.5"
                      Duration="0" />
                  </Storyboard>
                </VisualState>
                <VisualState Name="Pressed">
                  <Storyboard>
                    <DoubleAnimation
                      Storyboard.TargetName="HighlightBackground"
                      Storyboard.TargetProperty="Opacity"
                      To="0.5"
                      Duration="0" />
                  </Storyboard>
                </VisualState>
                <VisualState Name="Disabled">
                  <Storyboard>
                    <DoubleAnimation
                      Storyboard.TargetName="HighlightBackground"
                      Storyboard.TargetProperty="Opacity"
                      To="0"
                      Duration="0" />
                    <DoubleAnimation Storyboard.TargetName="NormalText"
                      Storyboard.TargetProperty="Opacity"
                      To=".35"
                      Duration="0" />
                  </Storyboard>
                </VisualState>
              </VisualStateGroup>
            </VisualStateManager>
          </Grid>
        </ControlTemplate>
      </Setter.Value>
    </Setter>
  </Style>

```

```

        </VisualStateGroup>
        <VisualStateGroup Name="SelectionStates">
            <VisualStateGroup.Transitions>
                <VisualTransition GeneratedDuration="0" />
            </VisualStateGroup.Transitions>
            <VisualState Name="Unselected" />
            <VisualState Name="Selected">
                <Storyboard>
                    <DoubleAnimation
                        Storyboard.TargetName="SelectedBackground"
                        Storyboard.TargetProperty="Opacity"
                        To=".75"
                        Duration="0" />
                </Storyboard>
            </VisualState>
        </VisualStateGroup>
        <VisualStateGroup Name="CalendarButtonFocusStates">
            <VisualStateGroup.Transitions>
                <VisualTransition GeneratedDuration="0" />
            </VisualStateGroup.Transitions>
            <VisualState Name="CalendarButtonFocused">
                <Storyboard>
                    <ObjectAnimationUsingKeyFrames
                        Storyboard.TargetName="DayButtonFocusVisual"
                        Storyboard.TargetProperty="Visibility"
                        Duration="0">
                        <DiscreteObjectKeyFrame KeyTime="0">
                            <DiscreteObjectKeyFrame.Value>
                                <Visibility>Visible</Visibility>
                            </DiscreteObjectKeyFrame.Value>
                        </DiscreteObjectKeyFrame>
                    </ObjectAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
            <VisualState Name="CalendarButtonUnfocused">
                <Storyboard>
                    <ObjectAnimationUsingKeyFrames
                        Storyboard.TargetName="DayButtonFocusVisual"
                        Storyboard.TargetProperty="Visibility"
                        Duration="0">
                        <DiscreteObjectKeyFrame KeyTime="0">
                            <DiscreteObjectKeyFrame.Value>
                                <Visibility>Collapsed</Visibility>
                            </DiscreteObjectKeyFrame.Value>
                        </DiscreteObjectKeyFrame>
                    </ObjectAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
        </VisualStateGroup>
        <VisualStateGroup Name="ActiveStates">
            <VisualStateGroup.Transitions>
                <VisualTransition GeneratedDuration="0" />
            </VisualStateGroup.Transitions>

```

```

<VisualState Name="Active" />
<VisualState Name="Inactive">
    <Storyboard>
        <ColorAnimation Duration="0"
            Storyboard.TargetName="NormalText"
            Storyboard.TargetProperty="

(TextElement.Foreground).
            (SolidColorBrush.Color)"
            To="#FF777777" />
    </Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup Name="DayStates">
    <VisualStateGroup.Transitions>
        <VisualTransition GeneratedDuration="0" />
    </VisualStateGroup.Transitions>
    <VisualState Name="RegularDay" />
    <VisualState Name="Today">
        <Storyboard>
            <DoubleAnimation Storyboard.TargetName="TodayBackground"
                Storyboard.TargetProperty="Opacity"
                To="1"
                Duration="0" />
            <ColorAnimation Duration="0"
                Storyboard.TargetName="NormalText"
                Storyboard.TargetProperty="

(TextElement.Foreground).
                (SolidColorBrush.Color)"
                To="#FFFFFF" />
        </Storyboard>
    </VisualState>
</VisualStateGroup>
<VisualStateGroup Name="BlackoutDayStates">
    <VisualStateGroup.Transitions>
        <VisualTransition GeneratedDuration="0" />
    </VisualStateGroup.Transitions>
    <VisualState Name="NormalDay" />
    <VisualState Name="BlackoutDay">
        <Storyboard>
            <DoubleAnimation Duration="0"
                Storyboard.TargetName="Blackout"
                Storyboard.TargetProperty="Opacity"
                To=".2" />
        </Storyboard>
    </VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Rectangle x:Name="TodayBackground"
    RadiusX="1"
    RadiusY="1"
    Opacity="0">
    <Rectangle.Fill>
        <SolidColorBrush Color="{DynamicResource
SelectedBackgroundColor}" />
    </Rectangle.Fill>

```

```
</Rectangle>
<Rectangle x:Name="SelectedBackground"
           RadiusX="1"
           RadiusY="1"
           Opacity="0">
    <Rectangle.Fill>
        <SolidColorBrush Color="{DynamicResource
SelectedBackgroundColor}" />
    </Rectangle.Fill>
</Rectangle>
<Border Background="{TemplateBinding Background}"
        BorderThickness="{TemplateBinding BorderThickness}"
        BorderBrush="{TemplateBinding BorderBrush}" />
<Rectangle x:Name="HighlightBackground"
           RadiusX="1"
           RadiusY="1"
           Opacity="0">
    <Rectangle.Fill>
        <SolidColorBrush Color="{DynamicResource
ControlMouseOverColor}" />
    </Rectangle.Fill>
</Rectangle>
<ContentPresenter x:Name="NormalText"
                  HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}"
                  VerticalAlignment="{TemplateBinding
VerticalContentAlignment}"
                  Margin="5,1,5,1">
    <TextElement.Foreground>
        <SolidColorBrush Color="#FF333333" />
    </TextElement.Foreground>
</ContentPresenter>
<Path x:Name="Blackout"
      Opacity="0"
      Margin="3"
      HorizontalAlignment="Stretch"
      VerticalAlignment="Stretch"
      RenderTransformOrigin="0.5,0.5"
      Fill="#FF000000"
      Stretch="Fill"
      Data="M8.1772461,11.029181 L10.433105,
            11.029181 L11.700684,12.801641 L12.973633,
            11.029181 L15.191895,11.029181 L12.844727,
            13.999395 L15.21875,17.060919 L12.962891,
            17.060919 L11.673828,15.256231 L10.352539,
            17.060919 L8.1396484,17.060919 L10.519043,
            14.042364 z" />
<Rectangle x:Name="DayButtonFocusVisual"
           Visibility="Collapsed"
           IsHitTestVisible="false"
           RadiusX="1"
           RadiusY="1">
    <Rectangle.Stroke>
        <SolidColorBrush Color="{DynamicResource
SelectedBackgroundColor}" />
```

```

        </Rectangle.Stroke>
    </Rectangle>
</Grid>
</ControlTemplate>
<Setter.Value>
</Setter>
</Style>

<!--Style for the months of a year and years of a decade.--&gt;
&lt;Style TargetType="CalendarButton"
      x:Key="CalendarButtonStyle"&gt;
    &lt;Setter Property="MinWidth"
           Value="40" /&gt;
    &lt;Setter Property="MinHeight"
           Value="42" /&gt;
    &lt;Setter Property="FontSize"
           Value="10" /&gt;
    &lt;Setter Property="HorizontalContentAlignment"
           Value="Center" /&gt;
    &lt;Setter Property="VerticalContentAlignment"
           Value="Center" /&gt;
    &lt;Setter Property="Template"&gt;
        &lt;Setter.Value&gt;
            &lt;ControlTemplate TargetType="CalendarButton"&gt;
                &lt;Grid&gt;
                    &lt;VisualStateManager.VisualStateGroups&gt;
                        &lt;VisualStateGroup Name="CommonStates"&gt;
                            &lt;VisualStateGroup.Transitions&gt;
                                &lt;VisualTransition GeneratedDuration="0:0:0.1" /&gt;
                            &lt;/VisualStateGroup.Transitions&gt;
                            &lt;VisualState Name="Normal" /&gt;
                            &lt;VisualState Name="MouseOver"&gt;
                                &lt;Storyboard&gt;
                                    &lt;DoubleAnimation Storyboard.TargetName="Background"
                                                       Storyboard.TargetProperty="Opacity"
                                                       To=".5"
                                                       Duration="0" /&gt;
                                &lt;/Storyboard&gt;
                            &lt;/VisualState&gt;
                            &lt;VisualState Name="Pressed"&gt;
                                &lt;Storyboard&gt;
                                    &lt;DoubleAnimation Storyboard.TargetName="Background"
                                                       Storyboard.TargetProperty="Opacity"
                                                       To=".5"
                                                       Duration="0" /&gt;
                                &lt;/Storyboard&gt;
                            &lt;/VisualState&gt;
                        &lt;/VisualStateGroup&gt;
                        &lt;VisualStateGroup Name="SelectionStates"&gt;
                            &lt;VisualStateGroup.Transitions&gt;
                                &lt;VisualTransition GeneratedDuration="0" /&gt;
                            &lt;/VisualStateGroup.Transitions&gt;
                            &lt;VisualState Name="Unselected" /&gt;
                            &lt;VisualState Name="Selected"&gt;
                                &lt;Storyboard&gt;
</pre>

```

```

        <DoubleAnimation
Storyboard.TargetName="SelectedBackground"
    Storyboard.TargetProperty="Opacity"
    To=".75"
    Duration="0" />
    </Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup Name="ActiveStates">
    <VisualStateGroup.Transitions>
        <VisualTransition GeneratedDuration="0" />
    </VisualStateGroup.Transitions>
    <VisualState Name="Active" />
    <VisualState Name="Inactive">
        <Storyboard>
            <ColorAnimation Duration="0"
                Storyboard.TargetName="NormalText"
                Storyboard.TargetProperty="

(TextElement.Foreground).
                (SolidColorBrush.Color)"
                To="#FF777777" />
        </Storyboard>
    </VisualState>
</VisualStateGroup>
<VisualStateGroup Name="CalendarButtonFocusStates">
    <VisualStateGroup.Transitions>
        <VisualTransition GeneratedDuration="0" />
    </VisualStateGroup.Transitions>
    <VisualState Name="CalendarButtonFocused">
        <Storyboard>
            <ObjectAnimationUsingKeyFrames Duration="0"

Storyboard.TargetName="CalendarButtonFocusVisual"

Storyboard.TargetProperty="Visibility">
                <DiscreteObjectKeyFrame KeyTime="0">
                    <DiscreteObjectKeyFrame.Value>
                        <Visibility>Visible</Visibility>
                    </DiscreteObjectKeyFrame.Value>
                </DiscreteObjectKeyFrame>
            </ObjectAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
    <VisualState Name="CalendarButtonUnfocused" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Rectangle x:Name="SelectedBackground"
    RadiusX="1"
    RadiusY="1"
    Opacity="0">
    <Rectangle.Fill>
        <SolidColorBrush Color="{DynamicResource
SelectedBackgroundColor}" />
    </Rectangle.Fill>
</Rectangle>

```

```

        <Rectangle x:Name="Background"
            RadiusX="1"
            RadiusY="1"
            Opacity="0">
            <Rectangle.Fill>
                <SolidColorBrush Color="{DynamicResource
SelectedBackgroundColor}" />
            </Rectangle.Fill>
        </Rectangle>
        <ContentPresenter x:Name="NormalText"
            HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}"
            VerticalAlignment="{TemplateBinding
VerticalContentAlignment}"
            Margin="1,0,1,1">
            <TextElement.Foreground>
                <SolidColorBrush Color="#FF333333" />
            </TextElement.Foreground>
        </ContentPresenter>
        <Rectangle x:Name="CalendarButtonFocusVisual"
            Visibility="Collapsed"
            IsHitTestVisible="false"
            RadiusX="1"
            RadiusY="1">
            <Rectangle.Stroke>
                <SolidColorBrush Color="{DynamicResource
SelectedBackgroundColor}" />
            </Rectangle.Stroke>
        </Rectangle>
    </Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
<Setter Property="Background">
    <Setter.Value>
        <SolidColorBrush Color="{DynamicResource ControlMediumColor}" />
    </Setter.Value>
</Setter>
</Style>

<!--Button to go to the previous month or year.--&gt;
&lt;ControlTemplate x:Key="PreviousButtonTemplate"
    TargetType="{x:Type Button}"&gt;
    &lt;Grid Cursor="Hand"&gt;
        &lt;VisualStateManager.VisualStateGroups&gt;
            &lt;VisualStateGroup x:Name="CommonStates"&gt;
                &lt;VisualState x:Name="Normal" /&gt;
                &lt;VisualState x:Name="MouseOver"&gt;
                    &lt;Storyboard&gt;
                        &lt;ColorAnimation Duration="0"
                            Storyboard.TargetName="path"
                            Storyboard.TargetProperty="(Shape.Fill).(
SolidColorBrush.Color)" To="{DynamicResource GlyphMouseOver}" /&gt;
                    &lt;/Storyboard&gt;
                &lt;/VisualState&gt;
            &lt;/VisualStateGroup&gt;
        &lt;/VisualStateManager.VisualStateGroups&gt;
    &lt;/Grid&gt;
&lt;/ControlTemplate&gt;
</pre>

```

```

        </VisualState>
        <VisualState x:Name="Disabled">
            <Storyboard>
                <DoubleAnimation Duration="0"
                    To=".5"
                    Storyboard.TargetProperty="(Shape.Fill).(
Brush.Opacity)">
                    Storyboard.TargetName="path" />
            </Storyboard>
        </VisualState>
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<!--<Rectangle Fill="Transparent" Opacity="1" Stretch="Fill"/><!--&gt;
&lt;Grid Background="Transparent"&gt;
    &lt;Path x:Name="path"
        Margin="14,-6,0,0"
        Stretch="Fill"
        HorizontalAlignment="Left"
        Height="10"
        VerticalAlignment="Center"
        Width="6"
        Data="M288.75,232.25 L288.75,240.625 L283,236.625 z"&gt;
        &lt;Path.Fill&gt;
            &lt;SolidColorBrush Color="{DynamicResource GlyphColor}" /&gt;
        &lt;/Path.Fill&gt;
    &lt;/Path&gt;
&lt;/Grid&gt;
&lt;/Grid&gt;
&lt;/ControlTemplate&gt;

&lt;!--Button to go to the next month or year.--&gt;
&lt;ControlTemplate x:Key="NextButtonTemplate"
    TargetType="{x:Type Button}"&gt;
    &lt;Grid Cursor="Hand"&gt;
        &lt;VisualStateManager.VisualStateGroups&gt;
            &lt;VisualStateGroup x:Name="CommonStates"&gt;
                &lt;VisualState x:Name="Normal" /&gt;
                &lt;VisualState x:Name="MouseOver"&gt;
                    &lt;Storyboard&gt;
                        &lt;ColorAnimation Duration="0"
                            To="{StaticResource GlyphMouseOver}"
                            Storyboard.TargetProperty="(Shape.Fill).(
SolidColorBrush.Color)"&gt;
                            Storyboard.TargetName="path" /&gt;
                    &lt;/Storyboard&gt;
                &lt;/VisualState&gt;
                &lt;VisualState x:Name="Disabled"&gt;
                    &lt;Storyboard&gt;
                        &lt;DoubleAnimation Duration="0"
                            To=".5"
                            Storyboard.TargetProperty="(Shape.Fill).(
Brush.Opacity)"&gt;
                            Storyboard.TargetName="path" /&gt;
                    &lt;/Storyboard&gt;
                &lt;/VisualState&gt;
            &lt;/VisualStateGroup&gt;
        &lt;/VisualStateManager.VisualStateGroups&gt;
    &lt;/Grid&gt;
</pre>

```

```

        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
<!--<Rectangle Fill="#11E5EBF1" Opacity="1" Stretch="Fill"/>-->
<Grid Background="Transparent">
    <Path x:Name="path"
        Data="M282.875,231.875 L282.875,240.375 L288.625,236 z"
        HorizontalAlignment="Right"
        Height="10"
        Margin="0,-6,14,0"
        Stretch="Fill"
        VerticalAlignment="Center"
        Width="6">
        <Path.Fill>
            <SolidColorBrush Color="{DynamicResource GlyphColor}" />
        </Path.Fill>
    </Path>
</Grid>
</Grid>
</ControlTemplate>

<!--Button to go up a level to the year or decade.-->
<ControlTemplate x:Key="HeaderButtonTemplate"
    TargetType="{x:Type Button}">
    <Grid Cursor="Hand">
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CommonStates">
                <VisualState x:Name="Normal" />
                <VisualState x:Name="MouseOver">
                    <Storyboard>
                        <ColorAnimation Duration="0"
                            To="{DynamicResource GlyphMouseOver}"
                            Storyboard.TargetProperty="

                            (TextElement.Foreground).
                            (SolidColorBrush.Color)"
                            Storyboard.TargetName="buttonContent" />
                    </Storyboard>
                </VisualState>
                <VisualState x:Name="Disabled">
                    <Storyboard>
                        <DoubleAnimation Duration="0"
                            To=".5"
                            Storyboard.TargetProperty="Opacity"
                            Storyboard.TargetName="buttonContent" />
                    </Storyboard>
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
        <ContentPresenter x:Name="buttonContent"
            Margin="1,4,1,9"
            ContentTemplate="{TemplateBinding ContentTemplate}"
            Content="{TemplateBinding Content}"
            TextElement.Foreground="#FF333333"
            HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}"
            VerticalAlignment="{TemplateBinding

```

```

        VerticalContentAlignment}" />
    </Grid>
</ControlTemplate>

<Style x:Key="CalendarItemStyle" TargetType="{x:Type CalendarItem}">
    <Setter Property="Margin"
        Value="0,3,0,3" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type CalendarItem}">
                <ControlTemplate.Resources>
                    <DataTemplate x:Key="{x:Static
CalendarItem.DayTitleTemplateResourceKey}">
                        <TextBlock Foreground="#FF333333"
                            FontWeight="Bold"
                            FontSize="9.5"
                            FontFamily="Verdana"
                            Margin="0,6,0,6"
                            Text="{Binding}"
                            HorizontalAlignment="Center"
                            VerticalAlignment="Center" />
                    </DataTemplate>
                </ControlTemplate.Resources>
                <Grid x:Name="PART_Root">
                    <Grid.Resources>
                        <SolidColorBrush x:Key="DisabledColor"
                            Color="#A5FFFFFF" />
                    </Grid.Resources>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="Disabled">
                                <Storyboard>
                                    <DoubleAnimation Duration="0"
                                        To="1"
                                        Storyboard.TargetProperty="Opacity"
                                        Storyboard.TargetName="PART_DisabledVisual" />
                                </Storyboard>
                            </VisualState>
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                    <Border BorderBrush="{TemplateBinding BorderBrush}"
                        BorderThickness="{TemplateBinding BorderThickness}"
                        Background="{TemplateBinding Background}"
                        CornerRadius="1">
                        <Border BorderBrush="#FFFFFF"
                            BorderThickness="2"
                            CornerRadius="1">
                            <Grid>
                                <Grid.Resources>
                                </Grid.Resources>
                                <Grid.ColumnDefinitions>
                                    <ColumnDefinition Width="Auto" />
                                    <ColumnDefinition Width="Auto" />

```

```
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition Height="*" />
</Grid.RowDefinitions>
<Button x:Name="PART_PreviousButton"
        Template="{StaticResource PreviousButtonTemplate}"
        Focusable="False"
        HorizontalAlignment="Left"
        Grid.Column="0"
        Grid.Row="0"
        Height="20"
        Width="28" />
<Button x:Name="PART_HeaderButton"
        FontWeight="Bold"
        Focusable="False"
        FontSize="10.5"
        HorizontalAlignment="Center"
        VerticalAlignment="Center"
        Grid.Column="1"
        Grid.Row="0"
        Template="{StaticResource HeaderButtonTemplate}" />
<Button x:Name="PART_NextButton"
        Focusable="False"
        HorizontalAlignment="Right"
        Grid.Column="2"
        Grid.Row="0"
        Template="{StaticResource NextButtonTemplate}"
        Height="20"
        Width="28" />
<Grid x:Name="PART_MonthView"
      Visibility="Visible"
      Grid.ColumnSpan="3"
      Grid.Row="1"
      Margin="6,-1,6,6"
      HorizontalAlignment="Center">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="Auto" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
</Grid>
```

```
</Grid>
<Grid x:Name="PART_YearView"
      Visibility="Hidden"
      Grid.ColumnSpan="3"
      Grid.Row="1"
      HorizontalAlignment="Center"
      Margin="6,-3,7,6">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto" />
      <ColumnDefinition Width="Auto" />
      <ColumnDefinition Width="Auto" />
      <ColumnDefinition Width="Auto" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
  </Grid>
</Grid>
</Border>
</Border>
<Rectangle x:Name="PART_DisabledVisual"
           Fill="{StaticResource DisabledColor}"
           Opacity="0"
           RadiusY="2"
           RadiusX="2"
           Stretch="Fill"
           Stroke="{StaticResource DisabledColor}"
           StrokeThickness="1"
           Visibility="Collapsed" />
</Grid>
<ControlTemplate.Triggers>
  <Trigger Property="IsEnabled"
           Value="False">
    <Setter Property="Visibility"
           TargetName="PART_DisabledVisual"
           Value="Visible" />
  </Trigger>
  <DataTrigger Binding="{Binding DisplayMode,
                           RelativeSource={RelativeSource FindAncestor,
                           AncestorType={x:Type Calendar}}}"
               Value="Year">
    <Setter Property="Visibility"
           TargetName="PART_MonthView"
           Value="Hidden" />
    <Setter Property="Visibility"
           TargetName="PART_YearView"
           Value="Visible" />
  </DataTrigger>
  <DataTrigger Binding="{Binding DisplayMode,
                           RelativeSource={RelativeSource FindAncestor,
                           AncestorType={x:Type Calendar}}}"
               Value="Decade">
    <Setter Property="Visibility"
```

```

        TargetName="PART_MonthView"
        Value="Hidden" />
    <Setter Property="Visibility"
        TargetName="PART_YearView"
        Value="Visible" />
    </DataTrigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style TargetType="{x:Type Calendar}">
    <Setter Property="CalendarButtonStyle"
        Value="{StaticResource CalendarButtonStyle}" />
    <Setter Property="CalendarDayButtonStyle"
        Value="{StaticResource CalendarDayButtonStyle}" />
    <Setter Property="CalendarItemStyle"
        Value="{StaticResource CalendarItemStyle}" />
    <Setter Property="Foreground"
        Value="#FF333333" />
    <Setter Property="Background">
        <Setter.Value>
            <LinearGradientBrush EndPoint="0.5,1"
                StartPoint="0.5,0">

                <!--The first two gradient stops specifies the background for
                    the calendar's heading and navigation buttons.-->
                <GradientStop Color="{DynamicResource HeaderTopColor}"
                    Offset="0" />
                <GradientStop Color="{DynamicResource ControlMediumColor}"
                    Offset="0.16" />

                <!--The next gradient stop specifies the background for
                    the calendar area.-->
                <GradientStop Color="{DynamicResource ControlLightColor}"
                    Offset="0.16" />
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
    <Setter Property="BorderBrush">
        <Setter.Value>
            <LinearGradientBrush EndPoint="0,1"
                StartPoint="0,0">
                <GradientStop Color="{DynamicResource BorderLightColor}"
                    Offset="0" />
                <GradientStop Color="{DynamicResource BorderDarkColor}"
                    Offset="1" />
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
    <Setter Property="BorderThickness"
        Value="1" />
    <Setter Property="Template">
        <Setter.Value>

```

```

<ControlTemplate TargetType="{x:Type Calendar}">
    <StackPanel x:Name="PART_Root"
        HorizontalAlignment="Center">
        <CalendarItem x:Name="PART_CalendarItem"
            BorderBrush="{TemplateBinding BorderBrush}"
            BorderThickness="{TemplateBinding BorderThickness}"
            Background="{TemplateBinding Background}"
            Style="{TemplateBinding CalendarItemStyle}" />
    </StackPanel>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

```

```
<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

CheckBox 样式和模板

项目 • 2023/02/06

本主题介绍 [CheckBox](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

CheckBox 部件

[CheckBox](#) 控件没有任何已命名的部件。

CheckBox 状态

下表列出了 [CheckBox](#) 控件的视觉对象状态。

VisualState 名称 名称	VisualStateGroup	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在控件上方。
已按下	CommonStates	已按下控件。
已禁用	CommonStates	已禁用控件。
已设定焦点	FocusStates	控件有焦点。
失去焦点	FocusStates	控件没有焦点。
已选中	CheckStates	<code>IsChecked</code> 上声明的默认值为 <code>true</code> 。
未选中	CheckStates	<code>IsChecked</code> 上声明的默认值为 <code>false</code> 。
不确定	CheckStates	<code>IsThreeState</code> 为 <code>true</code> 且 <code>IsChecked</code> 为 <code>null</code> 。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidUnfocused	ValidationStates	如果控件具有焦点，则 Validation.HasError 附加属性为 <code>true</code> 。
InvalidFocused	ValidationStates	如果控件不具有焦点，则 Validation.HasError 附加属性为 <code>true</code> 。

CheckBox ControlTemplate 示例

以下示例演示如何定义 CheckBox 控件的 ControlTemplate。

XAML

```
<Style x:Key="{x:Type CheckBox}">
    TargetType="{x:Type CheckBox}"
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="FocusVisualStyle"
        Value="{DynamicResource CheckBoxFocusVisual}" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type CheckBox}">
                <BulletDecorator Background="Transparent">
                    <BulletDecorator.Bullet>
                        <Border x:Name="Border"
                            Width="13"
                            Height="13"
                            CornerRadius="0"
                            BorderThickness="1">
                            <Border.BorderBrush>
                                <LinearGradientBrush StartPoint="0,0"
                                    EndPoint="0,1">
                                    <LinearGradientBrush.GradientStops>
                                        <GradientStopCollection>
                                            <GradientStop Color="{DynamicResource BorderLightColor}"
                                                Offset="0.0" />
                                            <GradientStop Color="{DynamicResource BorderDarkColor}"
                                                Offset="1.0" />
                                        </GradientStopCollection>
                                    </LinearGradientBrush.GradientStops>
                                </LinearGradientBrush>
                            </Border.BorderBrush>
                            <Border.Background>
                                <LinearGradientBrush StartPoint="0,0"
                                    EndPoint="0,1">
                                    <LinearGradientBrush.GradientStops>
                                        <GradientStopCollection>
                                            <GradientStop Color="{DynamicResource ControlLightColor}" />
                                            <GradientStop Color="{DynamicResource ControlMediumColor}"
                                                Offset="1.0" />
                                        </GradientStopCollection>
                                    </LinearGradientBrush.GradientStops>
                                </LinearGradientBrush>
                            </Border.Background>
                        </Border>
                    </BulletDecorator.Bullet>
                </BulletDecorator>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

```

        Width="7"
        Height="7"
        x:Name="CheckMark"
        SnapsToDevicePixels="False"
        StrokeThickness="2"
        Data="M 0 0 L 7 7 M 0 7 L 7 0">
    <Path.Stroke>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Stroke>
</Path>
<Path Visibility="Collapsed"
        Width="7"
        Height="7"
        x:Name="IndeterminateMark"
        SnapsToDevicePixels="False"
        StrokeThickness="2"
        Data="M 0 7 L 7 0">
    <Path.Stroke>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Stroke>
</Path>
</Grid>
</Border>
</BulletDecorator.Bullet>
<VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
        <VisualState x:Name="Normal" />
        <VisualState x:Name="MouseOver">
            <Storyboard>
                <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Border"
Storyboard.TargetProperty=""
(Panel.Background).
                    (GradientBrush.GradientStops)[1].
(GradientStop.Color)">
                    <EasingColorKeyFrame KeyTime="0"
Value="{StaticResource
ControlMouseOverColor}" />
                </ColorAnimationUsingKeyFrames>
            </Storyboard>
        </VisualState>
        <VisualState x:Name="Pressed">
            <Storyboard>
                <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Border"
Storyboard.TargetProperty=""
(Panel.Background).
                    (GradientBrush.GradientStops)[1].
(GradientStop.Color)">
                    <EasingColorKeyFrame KeyTime="0"
Value="{StaticResource
ControlPressedColor}" />
                </ColorAnimationUsingKeyFrames>
                <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Border"

```

```

        Storyboard.TargetProperty="

(Border.BorderBrush).
    (GradientBrush.GradientStops)[0].
(GradientStop.Color)">
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource
PressedBorderDarkColor}" />
    </ColorAnimationUsingKeyFrames>
    <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Border"
        Storyboard.TargetProperty="

(Border.BorderBrush).
    (GradientBrush.GradientStops)[1].
(GradientStop.Color)">
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource
PressedBorderLightColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Disabled" />
</VisualStateGroup>
<VisualStateGroup x:Name="CheckStates">
    <VisualState x:Name="Checked">
        <Storyboard
            <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="

(UIElement.Visibility)">

Storyboard.TargetName="CheckMark">
            <DiscreteObjectKeyFrame KeyTime="0"
                Value="{x:Static
Visibility.Visible}" />
            </ObjectAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
    <VisualState x:Name="Unchecked" />
    <VisualState x:Name="Indeterminate">
        <Storyboard
            <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="

(UIElement.Visibility)">

Storyboard.TargetName="IndeterminateMark">
            <DiscreteObjectKeyFrame KeyTime="0"
                Value="{x:Static
Visibility.Visible}" />
            </ObjectAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<ContentPresenter Margin="4,0,0,0"
    VerticalAlignment="Center"
    HorizontalAlignment="Left"
    RecognizesAccessKey="True" />
</BulletDecorator>

```

```
    </ControlTemplate>
  </Setter.Value>
</Setter>
</Style>
```

上一示例使用了一个或多个以下资源。

XAML

```
<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>
```

```
<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

ComboBox 样式和模板

项目 • 2023/02/06

本主题介绍 [ComboBox](#) 控件的样式和模板。可以修改默认的 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

ComboBox 部件

下表列出了 [ComboBox](#) 控件的命名部件。

组成部分	类型	描述
PART_EditableTextBox	TextBox	包含 ComboBox 的文本。
PART_Popup	Popup	包含组合框中项的下拉列表。

为 [ComboBox](#) 创建一个 [ControlTemplate](#) 时，你的模板可能在 [ScrollViewer](#) 中包含一个 [ItemsPresenter](#)。（[ItemsPresenter](#) 显示 [ComboBox](#) 中的每个项；[ScrollViewer](#) 支持在控件内滚动）。如果 [ItemsPresenter](#) 不是 [ScrollViewer](#) 的直接子元素，则必须为 [ItemsPresenter](#) 指定名称 `ItemsPresenter`。

ComboBox 状态

下表列出了 [ComboBox](#) 控件的状态。

VisualState 名称	VisualStateGroup 名称	描述
普通	CommonStates	默认状态。
已禁用	CommonStates	已禁用控件。
MouseOver	CommonStates	鼠标指针位于 ComboBox 控件上。
已设定焦点	FocusStates	控件有焦点。
失去焦点	FocusStates	控件没有焦点。
FocusedDropDown	FocusStates	ComboBox 的下拉菜单具有焦点。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性为 <code>true</code> ，控件具有焦点。

VisualState 名称	VisualStateGroup 名称	描述
InvalidUnfocused	ValidationStates	<code>Validation.HasError</code> 附加属性为 <code>true</code> ，控件不具有焦点。
可编辑	EditStates	<code>IsEditable</code> 属性为 <code>true</code> 。
不可编辑	EditStates	<code>IsEditable</code> 属性为 <code>false</code> 。

ComboBoxItem 部件

[ComboBoxItem](#) 控件没有任何已命名的部件。

ComboBoxItem 状态

下表列出了 [ComboBoxItem](#) 控件的状态。

VisualState 名称	VisualStateGroup 名称	描述
普通	CommonStates	默认状态。
已禁用	CommonStates	已禁用控件。
MouseOver	CommonStates	鼠标指针位于 ComboBoxItem 控件上。
已设定焦点	FocusStates	控件有焦点。
失去焦点	FocusStates	控件没有焦点。
选定	SelectionStates	该项当前已选定。
未选定	SelectionStates	未选定该项。
SelectedUnfocused	SelectionStates	该项已选定，但没有焦点。
有效	ValidationStates	该控件使用 Validation 类， <code>Validation.HasError</code> 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	<code>Validation.HasError</code> 附加属性为 <code>true</code> ，控件具有焦点。
InvalidUnfocused	ValidationStates	<code>Validation.HasError</code> 附加属性为 <code>true</code> ，控件不具有焦点。

ComboBox ControlTemplate 示例

以下示例演示如何定义 ComboBox 控件的 ControlTemplate 及关联类型。

XAML

```
<ControlTemplate x:Key="ComboBoxToggleButton"
    TargetType="{x:Type ToggleButton}">
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition Width="20" />
    </Grid.ColumnDefinitions>
    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="CommonStates">
            <VisualState x:Name="Normal" />
            <VisualState x:Name="MouseOver">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="
(Panel.Background)."
                        (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                        Storyboard.TargetName="Border">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource
ControlMouseOverColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
            <VisualState x:Name="Pressed" />
            <VisualState x:Name="Disabled">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="
(Panel.Background)."
                        (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                        Storyboard.TargetName="Border">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource
DisabledControlDarkColor}" />
                    </ColorAnimationUsingKeyFrames>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="
(Shape.Fill)."
                        (SolidColorBrush.Color)">
                        Storyboard.TargetName="Arrow">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource
DisabledForegroundColor}" />
                    </ColorAnimationUsingKeyFrames>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="
(Border.BorderBrush)."
                        (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                        Storyboard.TargetName="Border">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource
DisabledBorderDarkColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
</Grid>
<Path Data="M 0 0 L 20 0 L 20 20 Z" Fill="Black" Stroke="Black" StrokeThickness="1" StrokeDashOffset="1" />
</ControlTemplate>
```

```

        </VisualState>
    </VisualStateGroup>
    <VisualStateGroup x:Name="CheckStates">
        <VisualState x:Name="Checked">
            <Storyboard>
                <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="
                    (Panel.Background).(
                    GradientBrush.GradientStops)[1].(GradientStop.Color)">
                    <Storyboard.TargetName="Border">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource
                            ControlPressedColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
            <VisualState x:Name="Unchecked" />
            <VisualState x:Name="Indeterminate" />
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
<Border x:Name="Border"
    Grid.ColumnSpan="2"
    CornerRadius="2"
    BorderThickness="1">
    <Border.BorderBrush>
        <LinearGradientBrush EndPoint="0,1"
            StartPoint="0,0">
            <GradientStop Color="{DynamicResource BorderLightColor}"
                Offset="0" />
            <GradientStop Color="{DynamicResource BorderDarkColor}"
                Offset="1" />
        </LinearGradientBrush>
    </Border.BorderBrush>
    <Border.Background>

        <LinearGradientBrush StartPoint="0,0"
            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource ControlLightColor}" />
                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                        Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>

    </Border.Background>
</Border>
<Border Grid.Column="0"
    CornerRadius="2,0,0,2"
    Margin="1" >
    <Border.Background>
        <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
    </Border.Background>
</Border>
<Path x:Name="Arrow"

```

```

        Grid.Column="1"
        HorizontalAlignment="Center"
        VerticalAlignment="Center"
        Data="M 0 0 L 4 4 L 8 0 Z" >
    <Path.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Fill>
</Path>
</Grid>
</ControlTemplate>

<ControlTemplate x:Key="ComboBoxTextBox"
                  TargetType="{x:Type TextBox}">
    <Border x:Name="PART_ContentHost"
            Focusable="False"
            Background="{TemplateBinding Background}" />
</ControlTemplate>

<Style x:Key="{x:Type ComboBox}"
       TargetType="{x:Type ComboBox}">
    <Setter Property="SnapsToDevicePixels"
           Value="true" />
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="ScrollViewer.HorizontalScrollBarVisibility"
           Value="Auto" />
    <Setter Property="ScrollViewer.VerticalScrollBarVisibility"
           Value="Auto" />
    <Setter Property="ScrollViewer.CanContentScroll"
           Value="true" />
    <Setter Property="MinWidth"
           Value="120" />
    <Setter Property="MinHeight"
           Value="20" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type ComboBox}">
                <Grid>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="MouseOver" />
                            <VisualState x:Name="Disabled">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames
                                        Storyboard.TargetName="PART_EditableTextBox"
                                        Storyboard.TargetProperty=""
                                        (TextElement.Foreground).
                                            (SolidColorBrush.Color)">
                                        <EasingColorKeyFrame KeyTime="0"
                                            Value="{StaticResource
DisabledForegroundColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```

        </VisualStateGroup>
        <VisualStateGroup x:Name="EditStates">
            <VisualState x:Name="Editable">
                <Storyboard>
                    <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="
                        (UIElement.Visibility)">
                        Storyboard.TargetName="PART_EditableTextBox">
                            <DiscreteObjectKeyFrame KeyTime="0"
                                Value="{x:Static
                        Visibility.Visible}" />
                            </ObjectAnimationUsingKeyFrames>
                            <ObjectAnimationUsingKeyFrames
                                Storyboard.TargetProperty="(UIElement.Visibility)">
                                Storyboard.TargetName="ContentSite">
                                    <DiscreteObjectKeyFrame KeyTime="0"
                                        Value="{x:Static
                        Visibility.Hidden}" />
                                    </ObjectAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="Uneditable" />
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                    <ToggleButton x:Name="ToggleButton"
                        Template="{StaticResource ComboBoxToggleButton}"
                        Grid.Column="2"
                        Focusable="false"
                        ClickMode="Press"
                        IsChecked="{Binding IsDropDownOpen, Mode=TwoWay,
                        RelativeSource={RelativeSource TemplatedParent}}"/>
                    <ContentPresenter x:Name="ContentSite"
                        IsHitTestVisible="False"
                        Content="{TemplateBinding SelectionBoxItem}"
                        ContentTemplate="{TemplateBinding
                        SelectionBoxItemTemplate}"
                        ContentTemplateSelector="{TemplateBinding
                        ItemTemplateSelector}">
                        Margin="3,3,23,3"
                        VerticalAlignment="Stretch"
                        HorizontalAlignment="Left">
                    </ContentPresenter>
                    <TextBox x:Name="PART_EditableTextBox"
                        Style="{x:Null}"
                        Template="{StaticResource ComboBoxTextBox}"
                        HorizontalAlignment="Left"
                        VerticalAlignment="Bottom"
                        Margin="3,3,23,3"
                        Focusable="True"
                        Background="Transparent"
                        Visibility="Hidden"
                        IsReadOnly="{TemplateBinding IsReadOnly}" />
                    <Popup x:Name="Popup"
                        Placement="Bottom">

```

```

        IsOpen="{TemplateBinding IsDropDownOpen}"
        AllowsTransparency="True"
        Focusable="False"
        PopupAnimation="Slide">
    <Grid x:Name="DropDown"
          SnapsToDevicePixels="True"
          MinWidth="{TemplateBinding ActualWidth}"
          MaxHeight="{TemplateBinding MaxDropDownHeight}">
        <Border x:Name="DropDownBorder"
                BorderThickness="1">
            <Border.BorderBrush>
                <SolidColorBrush Color="{DynamicResource
BorderMediumColor}" />
            </Border.BorderBrush>
            <Border.Background>
                <SolidColorBrush Color="{DynamicResource
ControlLightColor}" />
            </Border.Background>
        </Border>
        <ScrollViewer Margin="4,6,4,6"
                      SnapsToDevicePixels="True">
            <StackPanel IsItemsHost="True"

KeyboardNavigation.Directionality="Contained" />
        </ScrollViewer>
    </Grid>
</Popup>
</Grid>
<ControlTemplate.Triggers>
    <Trigger Property="HasItems"
             Value="false">
        <Setter TargetName="DropDownBorder"
                Property="MinHeight"
                Value="95" />
    </Trigger>
    <Trigger Property="IsGrouping"
             Value="true">
        <Setter Property="ScrollViewer.CanContentScroll"
                Value="false" />
    </Trigger>
    <Trigger SourceName="Popup"
             Property="AllowsTransparency"
             Value="true">
        <Setter TargetName="DropDownBorder"
                Property="CornerRadius"
                Value="4" />
        <Setter TargetName="DropDownBorder"
                Property="Margin"
                Value="0,2,0,0" />
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

```

<Style x:Key="{x:Type ComboBoxItem}"
       TargetType="{x:Type ComboBoxItem}">
  <Setter Property="SnapsToDevicePixels"
         Value="true" />
  <Setter Property="OverridesDefaultStyle"
         Value="true" />
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="{x:Type ComboBoxItem}">
        <Border x:Name="Border"
                Padding="2"
                SnapsToDevicePixels="true"
                Background="Transparent">
          <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="SelectionStates">
              <VisualState x:Name="Unselected" />
              <VisualState x:Name="Selected">
                <Storyboard>
                  <ColorAnimationUsingKeyFrames
                    Storyboard.TargetName="Border"
                    Storyboard.TargetProperty=""
                    (Panel.Background).(SolidColorBrush.Color)">
                    <EasingColorKeyFrame KeyTime="0"
                                         Value="{StaticResource
SelectedBackgroundColor}" />
                  </ColorAnimationUsingKeyFrames>
                </Storyboard>
              </VisualState>
              <VisualState x:Name="SelectedUnfocused">
                <Storyboard>
                  <ColorAnimationUsingKeyFrames
                    Storyboard.TargetName="Border"
                    Storyboard.TargetProperty=""
                    (Panel.Background).(SolidColorBrush.Color)">
                    <EasingColorKeyFrame KeyTime="0"
                                         Value="{StaticResource
SelectedUnfocusedColor}" />
                  </ColorAnimationUsingKeyFrames>
                </Storyboard>
              </VisualState>
            </VisualStateGroup>
          </VisualStateManager.VisualStateGroups>
          <ContentPresenter />
        </Border>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```
<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
```

```
        Offset="1" />
    </LinearGradientBrush>

    <LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
        StartPoint="0,0"
        EndPoint="1,0">
        <LinearGradientBrush.GradientStops>
            <GradientStopCollection>
                <GradientStop Color="#000000FF"
                    Offset="0" />
                <GradientStop Color="#600000FF"
                    Offset="0.4" />
                <GradientStop Color="#600000FF"
                    Offset="0.6" />
                <GradientStop Color="#000000FF"
                    Offset="1" />
            </GradientStopCollection>
        </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

ContextMenu 样式和模板

项目 • 2023/02/06

本主题介绍 [ContextMenu](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

ContextMenu 部件

[ContextMenu](#) 控件没有任何已命名的部件。

为 [ContextMenu](#) 创建一个 [ControlTemplate](#) 时，你的模板可能在 [ScrollViewer](#) 中包含一个 [ItemsPresenter](#)。（[ItemsPresenter](#) 显示 [ContextMenu](#) 中的每个项；[ScrollViewer](#) 支持在控件内滚动）。如果 [ItemsPresenter](#) 不是 [ScrollViewer](#) 的直接子级，则必须将 [ItemsPresenter](#) 命名为 `ItemsPresenter`。

ContextMenu 状态

下表列出了 [ContextMenu](#) 控件的视觉对象状态。

VisualState 名称 名称	VisualStateGroup 名称	描述
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	如果控件不具有焦点，则 Validation.HasError 附加属性为 <code>true</code> 。

ContextMenu ControlTemplate 示例

以下示例演示如何定义 [ContextMenu](#) 控件的 [ControlTemplate](#)。

XAML

```
<Style TargetType="{x:Type ContextMenu}">
    <Setter Property="SnapsToDevicePixels"
        Value="True" />
    <Setter Property="OverridesDefaultStyle"
        Value="True" />
    <Setter Property="Grid.IsSharedSizeScope"
```

```

        Value="true" />
<Setter Property="HasDropShadow"
        Value="True" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type ContextMenu}">
            <Border x:Name="Border"
                Background="{StaticResource MenuPopupBrush}"
                BorderThickness="1">
                <Border.BorderBrush>
                    <SolidColorBrush Color="{StaticResource BorderMediumColor}" />
                </Border.BorderBrush>
                <StackPanel IsItemsHost="True"
                    KeyboardNavigation.DirectionalNavigation="Cycle" />
            </Border>
            <ControlTemplate.Triggers>
                <Trigger Property="HasDropShadow"
                    Value="true">
                    <Setter TargetName="Border"
                        Property="Padding"
                        Value="0,3,0,3" />
                    <Setter TargetName="Border"
                        Property="CornerRadius"
                        Value="4" />
                </Trigger>
            </ControlTemplate.Triggers>
        </ControlTemplate>
    </Setter.Value>
</Setter>
</Style>

```

ControlTemplate 使用以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

```

```

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

DataGrid 样式和模板

项目 · 2023/02/06

本主题介绍 [DataGrid](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#)，为控件提供独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

DataGrid 部件

下表列出了 [DataGrid](#) 控件的命名部件。

组成部分	类型	描述
PART_ColumnHeadersPresenter	DataGridColumnHeadersPresenter	包含列标题的行。

为 [DataGrid](#) 创建 [ControlTemplate](#) 时，模板的 [ScrollViewer](#) 中可能包含 [ItemsPresenter](#)。（[ItemsPresenter](#) 显示 [DataGrid](#) 中的每个项；[ScrollViewer](#) 在控件内实现滚动）。如果 [ItemsPresenter](#) 不是 [ScrollViewer](#) 的直接子级，则必须将 [ItemsPresenter](#) 命名为 `ItemsPresenter`。

[DataGrid](#) 的默认模板包含一个 [ScrollViewer](#) 控件。有关 [ScrollViewer](#) 定义的部分的详细信息，请参阅 [ScrollViewer 样式和模板](#)。

DataGrid 状态

下表列出了 [DataGrid](#) 控件的可视状态。

VisualState 名称	VisualStateGroup 名称	描述
普通	CommonStates	默认状态。
已禁用	CommonStates	已禁用控件。
InvalidFocused	ValidationStates	控件无效，但具有焦点。
InvalidUnfocused	ValidationStates	控件无效，并且没有焦点。
有效	ValidationStates	控件有效。

DataGridCell 部件

[DataGridCell](#) 元素没有任何命名部件。

DataGridCell 状态

下表列出了 DataGridCell 元素的可视状态。

VisualState 名称	VisualStateGroup 名称	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在单元格上方。
已设定焦点	FocusStates	单元格具有焦点。
失去焦点	FocusStates	单元格没有焦点
当前	CurrentStates	单元格是当前单元格。
常规	CurrentStates	单元格不是当前单元格。
显示	InteractionStates	单元格处于显示模式。
编辑	InteractionStates	单元格处于编辑模式。
选定	SelectionStates	单元格处于选定状态。
未选定	SelectionStates	单元格未处于选定状态。
InvalidFocused	ValidationStates	单元格无效，但具有焦点。
InvalidUnfocused	ValidationStates	单元格无效，并且没有焦点。
有效	ValidationStates	单元格有效。

DataGridRow 部件

DataGridRow 元素没有任何命名部件。

DataGridRow 状态

下表列出了 DataGridRow 元素的可视状态。

VisualState 名称	VisualStateGroup 名称	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在行上方。

VisualState 名称	VisualStateGroup 名称	描述
MouseOver_Editing	CommonStates	鼠标指针悬停在行上方，行处于编辑模式。
MouseOver_Selected	CommonStates	鼠标指针悬停在行上方，行处于选定状态。
MouseOver_Unfocused_Editing	CommonStates	鼠标指针悬停在行上方，行处于编辑模式，并且没有焦点。
MouseOver_Unfocused_Selected	CommonStates	鼠标指针悬停在行上方，行处于选定状态，并且没有焦点。
Normal_AlternatingRow	CommonStates	该行是一个交替行。
Normal_Editing	CommonStates	该行处于编辑模式。
Normal_Selected	CommonStates	该行处于选定状态。
Unfocused_Editing	CommonStates	该行处于编辑模式，并且没有焦点。
Unfocused_Selected	CommonStates	该行处于选定状态，并且没有焦点。
InvalidFocused	ValidationStates	控件无效，但具有焦点。
InvalidUnfocused	ValidationStates	控件无效，并且没有焦点。
有效	ValidationStates	控件有效。

DataGridRowHeader 部件

下表列出了 DataGridRowHeader 元素的命名部件。

组成部分	类型	描述
PART_TopHeaderGripper	Thumb	用于从顶部调整行标题大小的元素。
PART_BottomHeaderGripper	Thumb	用于从底部调整行标题大小的元素。

DataGridRowHeader States

下表列出了 DataGridRowHeader 元素的可视状态。

VisualState 名称	VisualStateGroup 名称	描述
-----------------------	--------------------------------	-----------

VisualState 名称	VisualStateGroup 名称	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在行上方。
MouseOver_CurrentRow	CommonStates	鼠标指针悬停在行上方，该行是当前行。
MouseOver_CurrentRow_Selected	CommonStates	鼠标指针悬停在行上方，该行是当前行，且处于选定状态。
MouseOver_EditingRow	CommonStates	鼠标指针悬停在行上方，行处于编辑模式。
MouseOver_Selected	CommonStates	鼠标指针悬停在行上方，行处于选定状态。
MouseOver_Unfocused_CurrentRow_Selected	CommonStates	鼠标指针悬停在行上方，该行是当前行，处于选定状态，并且没有焦点。
MouseOver_Unfocused_EditingRow	CommonStates	鼠标指针悬停在行上方，行处于编辑模式，并且没有焦点。
MouseOver_Unfocused_Selected	CommonStates	鼠标指针悬停在行上方，行处于选定状态，并且没有焦点。
Normal_CurrentRow	CommonStates	该行是当前行。
Normal_CurrentRow_Selected	CommonStates	该行是当前行，处于选定状态。
Normal_EditingRow	CommonStates	该行处于编辑模式。
Normal_Selected	CommonStates	该行处于选定状态。
Unfocused_CurrentRow_Selected	CommonStates	该行是当前行，处于选定状态，并且没有焦点。
Unfocused_EditingRow	CommonStates	该行处于编辑模式，并且没有焦点。
Unfocused_Selected	CommonStates	该行处于选定状态，并且没有焦点。
InvalidFocused	ValidationStates	控件无效，但具有焦点。

VisualState 名称	VisualStateGroup 名称	描述
InvalidUnfocused	ValidationStates	控件无效，并且没有焦点。
有效	ValidationStates	控件有效。

DataGridColumnHeadersPresenter 部件

下表列出了 [DataGridColumnHeadersPresenter](#) 元素的命名部件。

组成部分	类型	描述
PART_FillerColumnHeader	DataGridColumnHeader	列标题的占位符。

DataGridColumnHeadersPresenter 状态

下表列出了 [DataGridColumnHeadersPresenter](#) 元素的可视状态。

VisualState 名称	VisualStateGroup 名称	描述
InvalidFocused	ValidationStates	单元格无效，但具有焦点。
InvalidUnfocused	ValidationStates	单元格无效，并且没有焦点。
有效	ValidationStates	单元格有效。

DataGridColumnHeader Parts

下表列出了 [DataGridColumnHeader](#) 元素的命名部件。

组成部分	类型	描述
PART_LeftHeaderGripper	Thumb	用于从左侧调整列标题大小的元素。
PART_RightHeaderGripper	Thumb	用于从右侧调整列标题大小的元素。

DataGridColumnHeader 状态

下表列出了 [DataGridColumnHeader](#) 元素的可视状态。

VisualState 名称	VisualStateGroup 名称	描述
----------------	---------------------	----

VisualState 名称	VisualStateGroup 名称	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在控件上方。
已按下	CommonStates	已按下控件。
SortAscending	SortStates	列按升序排序。
SortDescending	SortStates	列按降序排序。
未排序	SortStates	该列未排序。
InvalidFocused	ValidationStates	控件无效，但具有焦点。
InvalidUnfocused	ValidationStates	控件无效，并且没有焦点。
有效	ValidationStates	控件有效。

DataGrid ControlTemplate 示例

以下示例演示如何定义 [DataGrid](#) 控件的 [ControlTemplate](#) 及其关联类型。

XAML

```

<BooleanToVisibilityConverter x:Key="bool2VisibilityConverter" />

<!--Style and template for the button in the upper left corner of the
DataGrid.-->
<Style TargetType="{x:Type Button}"
      x:Key="{ComponentResourceKey ResourceId=DataGridSelectAllButtonStyle,
      TypeInTargetAssembly={x:Type DataGrid}}">
<Setter Property="Template">
  <Setter.Value>
    <ControlTemplate TargetType="{x:Type Button}">
      <Grid>
        <VisualStateManager.VisualStateGroups>
          <VisualStateGroup x:Name="CommonStates">
            <VisualState x:Name="Normal" />
            <VisualState x:Name="MouseOver">
              <Storyboard>
                <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Border"
Storyboard.TargetProperty="(
Shape.Fill).(
GradientBrush.GradientStops)[1].(
GradientStop.Color)">
                <EasingColorKeyFrame KeyTime="0"
Value="{StaticResource
ControlMouseOverColor}" />
              
```

```

        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
<VisualState x:Name="Pressed">
    <Storyboard>
        <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Border"
                Storyboard.TargetProperty="

(Shape.Fill).
                (GradientBrush.GradientStops)[1].
(GradientStop.Color)">
            <EasingColorKeyFrame KeyTime="0"
                    Value="{StaticResource
ControlPressedColor}" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
<VisualState x:Name="Disabled">
    <Storyboard>
        <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="
(UIElement.Visibility)"

Storyboard.TargetName="Arrow">
            <DiscreteObjectKeyFrame KeyTime="0"
                    Value="{x:Static
Visibility.Collapsed}" />
        </ObjectAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Rectangle x:Name="Border"
            SnapsToDevicePixels="True">
<Rectangle.Stroke>
    <LinearGradientBrush EndPoint="0.5,1"
                    StartPoint="0.5,0">
        <GradientStop Color="{DynamicResource BorderLightColor}"
Offset="0" />
        <GradientStop Color="{DynamicResource BorderMediumColor}"
Offset="1" />
    </LinearGradientBrush>
</Rectangle.Stroke>
<Rectangle.Fill>
    <LinearGradientBrush EndPoint="0.5,1"
                    StartPoint="0.5,0">
        <GradientStop Color="{DynamicResource ControlLightColor}"
Offset="0" />
        <GradientStop Color="{DynamicResource ControlMediumColor}"
Offset="1" />
    </LinearGradientBrush>
</Rectangle.Fill>
</Rectangle>
<Polygon x:Name="Arrow"
            HorizontalAlignment="Right"
Margin="8,8,3,3"

```

```

        Opacity="0.15"
        Points="0,10 10,10 10,0"
        Stretch="Uniform"
        VerticalAlignment="Bottom">
    <Polygon.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Polygon.Fill>
</Polygon>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!--Style and template for the DataGrid.--&gt;
&lt;Style TargetType="{x:Type DataGrid}"&gt;
    &lt;Setter Property="Foreground"
        Value="{DynamicResource {x:Static SystemColors.ControlTextBrushKey}}" /&gt;
    &lt;Setter Property="BorderBrush"&gt;
        &lt;Setter.Value&gt;
            &lt;LinearGradientBrush EndPoint="0.5,1"
                StartPoint="0.5,0"&gt;
                &lt;GradientStop Color="{DynamicResource BorderLightColor}"
                    Offset="0" /&gt;
                &lt;GradientStop Color="{DynamicResource BorderDarkColor}"
                    Offset="1" /&gt;
            &lt;/LinearGradientBrush&gt;
        &lt;/Setter.Value&gt;
    &lt;/Setter&gt;
    &lt;Setter Property="BorderThickness"
        Value="1" /&gt;
    &lt;Setter Property="RowDetailsVisibilityMode"
        Value="VisibleWhenSelected" /&gt;
    &lt;Setter Property="ScrollViewer.CanContentScroll"
        Value="true" /&gt;
    &lt;Setter Property="ScrollViewer.PanningMode"
        Value="Both" /&gt;
    &lt;Setter Property="Stylus.IsFlicksEnabled"
        Value="False" /&gt;
    &lt;Setter Property="Template"&gt;
        &lt;Setter.Value&gt;
            &lt;ControlTemplate TargetType="{x:Type DataGrid}"&gt;
                &lt;Border x:Name="border"
                    SnapsToDevicePixels="True"
                    BorderBrush="{TemplateBinding BorderBrush}"
                    BorderThickness="{TemplateBinding BorderThickness}"
                    Padding="{TemplateBinding Padding}"&gt;
                    &lt;Border.Background&gt;
                        &lt;SolidColorBrush Color="{DynamicResource ControlLightColor}" /&gt;
                    &lt;/Border.Background&gt;
                    &lt;VisualStateManager.VisualStateGroups&gt;
                        &lt;VisualStateGroup x:Name="CommonStates"&gt;
                            &lt;VisualState x:Name="Disabled"&gt;
                                &lt;Storyboard&gt;
</pre>

```

```

        <ColorAnimationUsingKeyFrames
Storyboard.TargetName="border"
Storyboard.TargetProperty="

(Panel.Background).

(SolidColorBrush.Color)">
<EasingColorKeyFrame KeyTime="0"
Value="{DynamicResource

ControlLightColor}" />
</ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Normal" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<ScrollViewer x:Name="DG_ScrollViewer"
Focusable="false"
Background="Black">
<ScrollViewer.Template>
<ControlTemplate TargetType="{x:Type ScrollViewer}">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>

<Button Focusable="false"
Command="{x:Static DataGrid.SelectAllCommand}"
Style="{DynamicResource {ComponentResourceKey
ResourceId=DataGridSelectAllButtonStyle,
TypeInTargetAssembly={x:Type DataGrid}}}"
Visibility="{Binding HeadersVisibility,
ConverterParameter={x:Static
DataGridHeadersVisibility.All},
Converter={x:Static
DataGrid.HeadersVisibilityConverter},
RelativeSource={RelativeSource AncestorType={x:Type
DataGrid}}}"
Width="{Binding CellsPanelHorizontalOffset,
RelativeSource={RelativeSource AncestorType={x:Type
DataGrid}}}" />

<DataGridColumnHeadersPresenter
x:Name="PART_ColumnHeadersPresenter"
Grid.Column="1"
Visibility="{Binding

HeadersVisibility,
ConverterParameter={x:Static
DataGridHeadersVisibility.Column},
Converter={x:Static

```

```

        DataGrid.HeadersVisibilityConverter},
                    RelativeSource={RelativeSource AncestorType={x:Type
DataGrid}}}" />

        <ScrollContentPresenter
x:Name="PART_ScrollContentPresenter"
                        Grid.ColumnSpan="2"
                        Grid.Row="1"
                        CanContentScroll="{TemplateBinding
CanContentScroll}" />

        <ScrollBar x:Name="PART_VerticalScrollBar"
                        Grid.Column="2"
                        Grid.Row="1"
                        Orientation="Vertical"
                        ViewportSize="{TemplateBinding ViewportHeight}"
                        Maximum="{TemplateBinding ScrollableHeight}"
                        Visibility="{TemplateBinding
ComputedVerticalScrollBarVisibility}"
                        Value="{Binding VerticalOffset, Mode=OneWay,
RelativeSource={RelativeSource TemplatedParent}}"/>

        <Grid Grid.Column="1"
                        Grid.Row="2">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="{Binding
NonFrozenColumnsViewportHorizontalOffset,
RelativeSource={RelativeSource AncestorType=
{x:Type DataGrid}}}" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>

            <ScrollBar x:Name="PART_HorizontalScrollBar"
                        Grid.Column="1"
                        Orientation="Horizontal"
                        ViewportSize="{TemplateBinding
ViewportWidth}"
                        Maximum="{TemplateBinding ScrollableWidth}"
                        Visibility="{TemplateBinding
ComputedHorizontalScrollBarVisibility}"
                        Value="{Binding HorizontalOffset,
Mode=OneWay,
RelativeSource={RelativeSource TemplatedParent}}"/>
        </Grid>
    </Grid>
</ControlTemplate>
</ScrollViewer.Template>
<ItemsPresenter SnapsToDevicePixels="{TemplateBinding
SnapsToDevicePixels}" />
</ScrollViewer>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
<Style.Triggers>

```

```

<Trigger Property="IsGrouping"
    Value="true">
    <Setter Property="ScrollViewer.CanContentScroll"
        Value="false" />
</Trigger>
</Style.Triggers>
</Style>

<!--Style and template for the DataGridCell.-->
<Style TargetType="{x:Type DataGridCell}">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type DataGridCell}">
                <Border x:Name="border"
                    BorderBrush="Transparent"
                    BorderThickness="1"
                    Background="Transparent"
                    SnapsToDevicePixels="True">
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="FocusStates">
                            <VisualState x:Name="Unfocused" />
                            <VisualState x:Name="Focused" />
                        </VisualStateGroup>
                        <VisualStateGroup x:Name="CurrentStates">
                            <VisualState x:Name="Regular" />
                            <VisualState x:Name="Current">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames
                                        Storyboard.TargetName="border"
                                        Storyboard.TargetProperty="

(Border.BorderBrush).(SolidColorBrush.Color)">
                                        <EasingColorKeyFrame KeyTime="0"
                                            Value="{StaticResource
DatagridCurrentCellBorderColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                    <ContentPresenter SnapsToDevicePixels="{TemplateBinding
SnapsToDevicePixels}" />
                </Border>
            </ControlTemplate>
        <Setter.Value>
    </Setter>
</Style>

<!--Style and template for the DataGridRow.-->
<Style TargetType="{x:Type DataGridRow}">
    <Setter Property="Background">
        <Setter.Value>
            <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
        </Setter.Value>
    </Setter>
</Style>

```

```

        </Setter>
    <Setter Property="SnapsToDevicePixels"
            Value="true" />
    <Setter Property="Validation.ErrorTemplate"
            Value="{x:Null}" />
    <Setter Property="ValidationErrorTemplate">
        <Setter.Value>
            <ControlTemplate>
                <TextBlock Foreground="Red"
                           Margin="2,0,0,0"
                           Text="!"
                           VerticalAlignment="Center" />
            </ControlTemplate>
        </Setter.Value>
    </Setter>
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type DataGridRow}">
                <Border x:Name="DGR_Border"
                        BorderBrush="{TemplateBinding BorderBrush}"
                        BorderThickness="{TemplateBinding BorderThickness}"
                        SnapsToDevicePixels="True">
                    <Border.Background>
                        <LinearGradientBrush EndPoint="0.5,1"
                                              StartPoint="0.5,0">
                            <GradientStop Color="Transparent"
                                          Offset="0" />
                            <GradientStop Color="{DynamicResource ControlLightColor}"
                                          Offset="1" />
                        </LinearGradientBrush>
                    </Border.Background>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />

                            <!--Provide a different appearance for every other row.-->
                            <VisualState x:Name="Normal_AlternatingRow">
                                <Storyboard
                                    <ColorAnimationUsingKeyFrames
                                        Storyboard.TargetName="DGR_Border"
                                        Storyboard.TargetProperty="

(Panel.Background).(GradientBrush.GradientStops)[0].(GradientStop.Color)">
                                    <EasingColorKeyFrame KeyTime="0"
                                                          Value="{StaticResource ContentAreaColorLight}" />
                                </ColorAnimationUsingKeyFrames>
                                <ColorAnimationUsingKeyFrames
                                        Storyboard.TargetName="DGR_Border"
                                        Storyboard.TargetProperty="

(Panel.Background).(GradientBrush.GradientStops)[1].(GradientStop.Color)">

```

```

        <EasingColorKeyFrame KeyTime="0"
            Value="{StaticResource
ContentAreaColorDark}" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>

        <!--In this example, a row in Editing or selected mode has an
identical appearances. In other words, the states
Normal_Selected, Unfocused_Selected, Normal_Editing,
MouseOver_Editing, MouseOver_Unfocused_Editing,
and Unfocused_Editing are identical.-->
<VisualState x:Name="Normal_Selected">
    <Storyboard>
        <ColorAnimationUsingKeyFrames
Storyboard.TargetName="DGR_Border"
                Storyboard.TargetProperty="

(Panel.Background).>
        (GradientBrush.GradientStops)[0].
(GradientStop.Color)">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource
ControlMediumColor}" />
            </ColorAnimationUsingKeyFrames>
            <ColorAnimationUsingKeyFrames
Storyboard.TargetName="DGR_Border"
                Storyboard.TargetProperty="

(Panel.Background).>
        (GradientBrush.GradientStops)[1].
(GradientStop.Color)">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource
ControlDarkColor}" />
            </ColorAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>

    <VisualState x:Name="Unfocused_Selected">
        <Storyboard>
            <ColorAnimationUsingKeyFrames
Storyboard.TargetName="DGR_Border"
                Storyboard.TargetProperty="

(Panel.Background).>
        (GradientBrush.GradientStops)[0].
(GradientStop.Color)">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource
ControlMediumColor}" />
            </ColorAnimationUsingKeyFrames>
            <ColorAnimationUsingKeyFrames
Storyboard.TargetName="DGR_Border"
                Storyboard.TargetProperty="

(Panel.Background).>
        (GradientBrush.GradientStops)[1].
(GradientStop.Color)">

```

```

        <EasingColorKeyFrame KeyTime="0"
                             Value="{StaticResource
ControlDarkColor}" />
            </ColorAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>

    <VisualState x:Name="Normal_Editing">
        <Storyboard>
            <ColorAnimationUsingKeyFrames
Storyboard.TargetName="DGR_Border"
                                Storyboard.TargetProperty="
(Panel.Background).
                            (GradientBrush.GradientStops)[0].
(GradientStop.Color)">
                <EasingColorKeyFrame KeyTime="0"
                                     Value="{StaticResource
ControlMediumColor}" />
                    </ColorAnimationUsingKeyFrames>
                    <ColorAnimationUsingKeyFrames
Storyboard.TargetName="DGR_Border"
                                Storyboard.TargetProperty="
(Panel.Background).
                            (GradientBrush.GradientStops)[1].
(GradientStop.Color)">
                <EasingColorKeyFrame KeyTime="0"
                                     Value="{StaticResource
ControlDarkColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>

            <VisualState x:Name="MouseOver_Editing">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames
Storyboard.TargetName="DGR_Border"
                                Storyboard.TargetProperty="
(Panel.Background).
                            (GradientBrush.GradientStops)[0].
(GradientStop.Color)">
                        <EasingColorKeyFrame KeyTime="0"
                                             Value="{StaticResource
ControlMediumColor}" />
                            </ColorAnimationUsingKeyFrames>
                            <ColorAnimationUsingKeyFrames
Storyboard.TargetName="DGR_Border"
                                Storyboard.TargetProperty="
(Panel.Background).
                            (GradientBrush.GradientStops)[1].
(GradientStop.Color)">
                        <EasingColorKeyFrame KeyTime="0"
                                             Value="{StaticResource
ControlDarkColor}" />
                            </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>

```

```

        </VisualState>

        <VisualState x:Name="MouseOver_Unfocused_Editing">
            <Storyboard>
                <ColorAnimationUsingKeyFrames
Storyboard.TargetName="DGR_Border"
                        Storyboard.TargetProperty="

(Panel.Background).
                    (GradientBrush.GradientStops)[0].
(GradientStop.Color)">
                    <EasingColorKeyFrame KeyTime="0"
                        Value="{StaticResource
ControlMediumColor}" />
                    </ColorAnimationUsingKeyFrames>
                    <ColorAnimationUsingKeyFrames
Storyboard.TargetName="DGR_Border"
                        Storyboard.TargetProperty="

(Panel.Background).
                    (GradientBrush.GradientStops)[1].
(GradientStop.Color)">
                    <EasingColorKeyFrame KeyTime="0"
                        Value="{StaticResource
ControlDarkColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>

            <VisualState x:Name="Unfocused_Editing">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames
Storyboard.TargetName="DGR_Border"
                        Storyboard.TargetProperty="

(Panel.Background).
                    (GradientBrush.GradientStops)[0].
(GradientStop.Color)">
                    <EasingColorKeyFrame KeyTime="0"
                        Value="{StaticResource
ControlMediumColor}" />
                    </ColorAnimationUsingKeyFrames>
                    <ColorAnimationUsingKeyFrames
Storyboard.TargetName="DGR_Border"
                        Storyboard.TargetProperty="

(Panel.Background).
                    (GradientBrush.GradientStops)[1].
(GradientStop.Color)">
                    <EasingColorKeyFrame KeyTime="0"
                        Value="{StaticResource
ControlDarkColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>

            <VisualState x:Name="MouseOver">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames

```

```

Storyboard.TargetName="DGR_Border"
    Storyboard.TargetProperty=""
(Panel.Background).
    (GradientBrush.GradientStops)[0].
(GradientStop.Color)">
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource
ControlMediumColor}" />
    </ColorAnimationUsingKeyFrames>
    <ColorAnimationUsingKeyFrames
Storyboard.TargetName="DGR_Border"
    Storyboard.TargetProperty=""
(Panel.Background).
    (GradientBrush.GradientStops)[1].
(GradientStop.Color)">
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource
ControlMouseOverColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>

<!--In this example, the appearance of a selected row
that has the mouse over it is the same regardless of
whether the row is selected. In other words, the states
MouseOver_Editing and MouseOver_Unfocused_Editing are
identical.--&gt;
&lt;VisualState x:Name="MouseOver_Selected"&gt;
    &lt;Storyboard&gt;
        &lt;ColorAnimationUsingKeyFrames
Storyboard.TargetName="DGR_Border"
    Storyboard.TargetProperty=""
(Panel.Background).
        (GradientBrush.GradientStops)[0].
(GradientStop.Color)"&gt;
            &lt;EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource
ControlMouseOverColor}" /&gt;
            &lt;/ColorAnimationUsingKeyFrames&gt;
            &lt;ColorAnimationUsingKeyFrames
Storyboard.TargetName="DGR_Border"
    Storyboard.TargetProperty=""
(Panel.Background).
        (GradientBrush.GradientStops)[1].
(GradientStop.Color)"&gt;
            &lt;EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource
ControlMouseOverColor}" /&gt;
            &lt;/ColorAnimationUsingKeyFrames&gt;
&lt;/Storyboard&gt;
&lt;/VisualState&gt;

&lt;VisualState x:Name="MouseOver_Unfocused_Selected"&gt;
    &lt;Storyboard&gt;
        &lt;ColorAnimationUsingKeyFrames
</pre>

```

```

Storyboard.TargetName="DGR_Border"                               Storyboard.TargetProperty=""
(Panel.Background).
    (GradientBrush.GradientStops)[0].
(GradientStop.Color)">
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource
ControlMouseOverColor}" />
    </ColorAnimationUsingKeyFrames>
    <ColorAnimationUsingKeyFrames
Storyboard.TargetName="DGR_Border"                               Storyboard.TargetProperty=""
(Panel.Background).
    (GradientBrush.GradientStops)[1].
(GradientStop.Color)">
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource
ControlMouseOverColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>

<SelectiveScrollingGrid>
    <SelectiveScrollingGrid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="*" />
    </SelectiveScrollingGrid.ColumnDefinitions>
    <SelectiveScrollingGrid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="Auto" />
    </SelectiveScrollingGrid.RowDefinitions>
    <DataGridCellsPresenter Grid.Column="1"
        ItemsPanel="{TemplateBinding
ItemsPanel}"
        SnapsToDevicePixels="{TemplateBinding
SnapsToDevicePixels}" />
    <DataGridDetailsPresenter Grid.Column="1"
        Grid.Row="1"
        Visibility="{TemplateBinding
DetailsVisibility}">
        SelectiveScrollingGrid.SelectiveScrollingOrientation=
            "{Binding AreRowDetailsFrozen,
            ConverterParameter={x:Static
SelectiveScrollingOrientation.Vertical},
            Converter={x:Static DataGrid.RowDetailsScrollingConverter},
            RelativeSource={RelativeSource AncestorType={x:Type
DataGrid}}}">
            <DataGridRowHeader Grid.RowSpan="2"
                SelectiveScrollingGrid.SelectiveScrollingOrientation="Vertical"
                    Visibility="{Binding HeadersVisibility,
                    ConverterParameter={x:Static DataGridHeadersVisibility.Row},
```

```

        Converter={x:Static DataGrid.HeadersVisibilityConverter},
        RelativeSource={RelativeSource AncestorType={x:Type
DataGrid}}}" />
    </SelectiveScrollingGrid>
</Border>
</ControlTemplate>
<Setter.Value>
</Setter>
</Style>

<!--Style and template for the resize control on the DataGridRowHeader.--&gt;
&lt;Style x:Key="RowHeaderGripperStyle"
    TargetType="{x:Type Thumb}"&gt;
&lt;Setter Property="Height"
    Value="8" /&gt;
&lt;Setter Property="Background"
    Value="Transparent" /&gt;
&lt;Setter Property="Cursor"
    Value="SizeNS" /&gt;
&lt;Setter Property="Template"&gt;
    &lt;Setter.Value&gt;
        &lt;ControlTemplate TargetType="{x:Type Thumb}"&gt;
            &lt;Border Background="{TemplateBinding Background}"
                Padding="{TemplateBinding Padding}" /&gt;
        &lt;/ControlTemplate&gt;
    &lt;/Setter.Value&gt;
&lt;/Setter&gt;
&lt;/Style&gt;

<!--Style and template for the DataGridRowHeader.--&gt;
&lt;Style TargetType="{x:Type DataGridRowHeader}"&gt;
&lt;Setter Property="Template"&gt;
    &lt;Setter.Value&gt;
        &lt;ControlTemplate TargetType="{x:Type DataGridRowHeader}"&gt;
            &lt;Grid&gt;
                &lt;VisualStateManager.VisualStateGroups&gt;
                    &lt;!--This example does not specify an appearance for every
                        state. You can add storyboard to the states that are listed
                        to change the appearance of the DataGridRowHeader when it is
                        in a specific state.--&gt;
                    &lt;VisualStateGroup x:Name="CommonStates"&gt;
                        &lt;VisualState x:Name="Normal" /&gt;
                        &lt;VisualState x:Name="Normal_CurrentRow" /&gt;
                        &lt;VisualState x:Name="Unfocused_EditingRow" /&gt;
                        &lt;VisualState x:Name="Normal_EditingRow" /&gt;
                        &lt;VisualState x:Name="MouseOver"&gt;
                            &lt;Storyboard&gt;
                                &lt;ColorAnimationUsingKeyFrames
Storyboard.TargetName="rowHeaderBorder"
Storyboard.TargetProperty="

(Panel.Background).(GradientBrush.GradientStops)[1].
(GradientStop.Color)"&gt;
                            &lt;EasingColorKeyFrame KeyTime="0"
Value="{StaticResource
</pre>

```

```

ControlMouseOverColor}" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
<VisualState x:Name="MouseOver_CurrentRow" />
<VisualState x:Name="MouseOver_Unfocused_EditingRow" />
<VisualState x:Name="MouseOver_EditingRow" />
<VisualState x:Name="MouseOver_Unfocused_Selected" />
<VisualState x:Name="MouseOver_Selected" />
<VisualState x:Name="MouseOver_Unfocused_CurrentRow_Selected" />
/>
<VisualState x:Name="MouseOver_CurrentRow_Selected" />
<VisualState x:Name="Unfocused_Selected" />
<VisualState x:Name="Unfocused_CurrentRow_Selected" />
<VisualState x:Name="Normal_CurrentRow_Selected" />
<VisualState x:Name="Normal_Selected" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border x:Name="rowHeaderBorder"
        Width="10"
        BorderThickness="1">
    <Border.BorderBrush>
        <LinearGradientBrush EndPoint="0.5,1"
                            StartPoint="0.5,0">
            <GradientStop Color="{DynamicResource BorderLightColor}"
                           Offset="0" />
            <GradientStop Color="{DynamicResource BorderDarkColor}"
                           Offset="1" />
        </LinearGradientBrush>
    </Border.BorderBrush>
    <Border.Background>
        <LinearGradientBrush EndPoint="0.5,1"
                            StartPoint="0.5,0">
            <GradientStop Color="{DynamicResource ControlLightColor}"
                           Offset="0" />
            <GradientStop Color="{DynamicResource ControlMediumColor}"
                           Offset="1" />
        </LinearGradientBrush>
    </Border.Background>
    <StackPanel Orientation="Horizontal">
        <ContentPresenter VerticalAlignment="Center"
                          SnapsToDevicePixels="{TemplateBinding
SnapsToDevicePixels}" />
        <Control SnapsToDevicePixels="false"
                 Template="{Binding Validation.ErrorTemplate,
RelativeSource={RelativeSource AncestorType={x:Type
DataGridRow}}}" Visibility="{Binding (Validation.HasError),
Converter={StaticResource bool2VisibilityConverter},
RelativeSource={RelativeSource AncestorType={x:Type
DataGridRow}}}" />
    </StackPanel>
</Border>
<Thumb x:Name="PART_TopHeaderGripper"

```

```

        Style="{StaticResource RowHeaderGripperStyle}"
        VerticalAlignment="Top" />
    <Thumb x:Name="PART_BottomHeaderGripper"
        Style="{StaticResource RowHeaderGripperStyle}"
        VerticalAlignment="Bottom" />
    </Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!--Style and template for the resize control on the DataGridColumnHeader.-->
>
<Style x:Key="ColumnHeaderGripperStyle"
    TargetType="{x:Type Thumb}">
    <Setter Property="Width"
        Value="8" />
    <Setter Property="Background"
        Value="Transparent" />
    <Setter Property="Cursor"
        Value="SizeWE" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Thumb}">
                <Border Background="{TemplateBinding Background}"
                    Padding="{TemplateBinding Padding}" />
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<!--Style and template for the DataGridColumnHeader.-->
<Style TargetType="{x:Type DataGridColumnHeader}">
    <Setter Property="VerticalContentAlignment"
        Value="Center" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type DataGridColumnHeader}">
                <Grid>
                    <Border x:Name="columnHeaderBorder"
                        BorderThickness="1"
                        Padding="3,0,3,0">
                        <Border.BorderBrush>
                            <LinearGradientBrush EndPoint="0.5,1"
                                StartPoint="0.5,0">
                                <GradientStop Color="{DynamicResource BorderLightColor}"
                                    Offset="0" />
                                <GradientStop Color="{DynamicResource BorderDarkColor}"
                                    Offset="1" />
                            </LinearGradientBrush>
                        </Border.BorderBrush>
                        <Border.Background>
                            <LinearGradientBrush EndPoint="0.5,1"
                                StartPoint="0.5,0">
                                <GradientStop Color="{DynamicResource ControlLightColor}">

```

```

        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="1" />
    </LinearGradientBrush>
</Border.Background>
<ContentPresenter HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}"
    SnapsToDevicePixels="{TemplateBinding
SnapsToDevicePixels}"
    VerticalAlignment="{TemplateBinding
VerticalContentAlignment}" />
</Border>

<Thumb x:Name="PART_LeftHeaderGripper"
    HorizontalAlignment="Left"
    Style="{StaticResource ColumnHeaderGripperStyle}" />
<Thumb x:Name="PART_RightHeaderGripper"
    HorizontalAlignment="Right"
    Style="{StaticResource ColumnHeaderGripperStyle}" />
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
<Setter Property="Background">
<Setter.Value>
    <LinearGradientBrush EndPoint="0.5,1"
        StartPoint="0.5,0">
        <GradientStop Color="{DynamicResource ControlLightColor}"
            Offset="0" />
        <GradientStop Color="{DynamicResource ControlMediumColor}"
            Offset="1" />
    </LinearGradientBrush>
</Setter.Value>
</Setter>
</Style>

<!--Style and template for the DataGridColumnHeadersPresenter.-->
<Style TargetType="{x:Type DataGridColumnHeadersPresenter}">
<Setter Property="Template">
<Setter.Value>
    <ControlTemplate TargetType="{x:Type DataGridColumnHeadersPresenter}">
        <Grid>
            <DataGridColumnHeader x:Name="PART_FillerColumnHeader"
                IsHitTestVisible="False" />
            <ItemsPresenter />
        </Grid>
    </ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```
<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
```

```
        Offset="1" />
    </LinearGradientBrush>

    <LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
        StartPoint="0,0"
        EndPoint="1,0">
        <LinearGradientBrush.GradientStops>
            <GradientStopCollection>
                <GradientStop Color="#000000FF"
                    Offset="0" />
                <GradientStop Color="#600000FF"
                    Offset="0.4" />
                <GradientStop Color="#600000FF"
                    Offset="0.6" />
                <GradientStop Color="#000000FF"
                    Offset="1" />
            </GradientStopCollection>
        </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

DatePicker 样式和模板

项目 · 2023/02/06

本主题介绍 [DatePicker](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

DatePicker 部件

下表列出了 [DatePicker](#) 控件的已命名部件。

组成部分	类型	描述
PART_Root	Grid	控件的根。
PART_Button	Button	打开和关闭 Calendar 的按钮。
PART_TextBox	DatePickerTextBox	允许输入日期的文本框。
PART_Popup	Popup	DatePicker 控件的弹出窗口。

DatePicker 状态

下表列出了 [DatePicker](#) 控件的视觉对象状态。

VisualState 名称	VisualStateGroup 名称	描述
普通	CommonStates	默认状态。
已禁用	CommonStates	禁用 DatePicker 。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

DatePickerTextBox 部件

下表列出了 [DatePickerTextBox](#) 控件的已命名部件。

组成部分	类型	描述
PART_Watermark	ContentControl	DatePicker 中包含初始文本的元素。
PART_ContentElement	FrameworkElement	可包含 FrameworkElement 的视觉对象元素。 TextBox 的文本显示在此元素中。

DatePickerTextBox 状态

下表列出了 DatePickerTextBox 控件的视觉对象状态。

VisualState 名称 名称	VisualStateGroup	描述
普通	CommonStates	默认状态。
已禁用	CommonStates	禁用 DatePickerTextBox。
MouseOver	CommonStates	鼠标指针悬停在 DatePickerTextBox 上方。
ReadOnly	CommonStates	用户不能更改 DatePickerTextBox 中的文本。
已设定焦点	FocusStates	控件有焦点。
失去焦点	FocusStates	控件没有焦点。
已打水印	WatermarkStates	控件显示其初始文本。 DatePickerTextBox 处于用户未输入文本或未选择日期的状态。
无水印	WatermarkStates	用户在 DatePickerTextBox 中输入了文本或在 DatePicker 中选择了一个日期。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 false。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是 true，并且控件具有焦点。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是 true，并且控件不具有焦点。

DatePicker ControlTemplate 示例

下例演示如何定义 DatePicker 控件的 ControlTemplate。

XAML

```

<!--In this example, an implicit style for Calendar is defined elsewhere
in the application. DatePickerCalendarStyle is based on the implicit
style so that the DatePicker will use the application's calendar style.-->
<Style x:Key="DatePickerCalendarStyle"
       TargetType="{x:Type Calendar}"
       BasedOn="{StaticResource {x:Type Calendar}}"/>

<!--The template for the button that displays the calendar.-->
<Style x:Key="DropDownButtonStyle"
       TargetType="Button">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Button}">
                <Grid>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualStateGroup.Transitions>
                                <VisualTransition GeneratedDuration="0" />
                                <VisualTransition GeneratedDuration="0:0:0.1"
                                      To="MouseOver" />
                                <VisualTransition GeneratedDuration="0:0:0.1"
                                      To="Pressed" />
                            </VisualStateGroup.Transitions>
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="MouseOver">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames BeginTime="0"
                                         Duration="00:00:00.001"
                                         Storyboard.TargetName="BackgroundGradient"
                                         Storyboard.TargetProperty="<!--(Border.Background)-->
                                         (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                                        <SplineColorKeyFrame KeyTime="0"
                                             Value="#F2FFFFFF" />
                                    </ColorAnimationUsingKeyFrames>
                                    <ColorAnimationUsingKeyFrames BeginTime="0"
                                         Duration="00:00:00.001"
                                         Storyboard.TargetName="BackgroundGradient"
                                         Storyboard.TargetProperty="<!--(Border.Background)-->
                                         (GradientBrush.GradientStops)[2].(GradientStop.Color)">
                                        <SplineColorKeyFrame KeyTime="0"
                                             Value="#CCFFFFFF" />
                                    </ColorAnimationUsingKeyFrames>
                                    <ColorAnimation Duration="0"
                                         To="#FF448DCA"
                                         Storyboard.TargetName="Background"
                                         Storyboard.TargetProperty="<!--(SolidColorBrush.Color)-->
                                         (SolidColorBrush.Color)">
                                        <Storyboard.TargetName="Background" />
                                    <ColorAnimationUsingKeyFrames BeginTime="0"
                                         Duration="00:00:00.001"
                                         Storyboard.TargetName="Background" />
                                </Storyboard>
                            </VisualState>
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```
Storyboard.TargetName="BackgroundGradient"           Storyboard.TargetProperty="

(Border.Background).

(GradientBrush.GradientStops)[3].(GradientStop.Color)">
<SplineColorKeyFrame KeyTime="0"
Value="#FFFFFF" />
</ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Pressed">
<Storyboard>
<ColorAnimationUsingKeyFrames BeginTime="0"
Duration="00:00:00.001"

Storyboard.TargetName="Background"           Storyboard.TargetProperty="

(Border.Background).

(SolidColorBrush.Color)">
<SplineColorKeyFrame KeyTime="0"
Value="#FF448DCA" />
</ColorAnimationUsingKeyFrames>
<DoubleAnimationUsingKeyFrames BeginTime="0"
Duration="00:00:00.001"
Storyboard.TargetProperty="

(UIElement.Opacity)">

Storyboard.TargetName="Highlight">
<SplineDoubleKeyFrame KeyTime="0"
Value="1" />
</DoubleAnimationUsingKeyFrames>
<ColorAnimationUsingKeyFrames BeginTime="0"
Duration="00:00:00.001"

Storyboard.TargetName="BackgroundGradient"           Storyboard.TargetProperty="

(Border.Background).

(GradientBrush.GradientStops)[0].(GradientStop.Color)">
<SplineColorKeyFrame KeyTime="0"
Value="#F4FFFFFF" />
</ColorAnimationUsingKeyFrames>
<ColorAnimationUsingKeyFrames BeginTime="0"
Duration="00:00:00.001"

Storyboard.TargetName="BackgroundGradient"           Storyboard.TargetProperty="

(Border.Background).

(GradientBrush.GradientStops)[1].(GradientStop.Color)">
<SplineColorKeyFrame KeyTime="0"
Value="#EAFFFFFF" />
</ColorAnimationUsingKeyFrames>
<ColorAnimationUsingKeyFrames BeginTime="0"
Duration="00:00:00.001"

Storyboard.TargetName="BackgroundGradient"
```

```

        Storyboard.TargetProperty=""
    (Border.Background).
        (GradientBrush.GradientStops)[2].(GradientStop.Color)">
        <SplineColorKeyFrame KeyTime="0"
            Value="#C6FFFFFF" />
    </ColorAnimationUsingKeyFrames>
    <ColorAnimationUsingKeyFrames BeginTime="0"
        Duration="00:00:00.001"

Storyboard.TargetName="BackgroundGradient"
    Storyboard.TargetProperty=""
    (Border.Background).
        (GradientBrush.GradientStops)[3].(GradientStop.Color)">
        <SplineColorKeyFrame KeyTime="0"
            Value="#6BFFFFFF" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Disabled" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Grid Background="#11FFFFFF"
    FlowDirection="LeftToRight"
    HorizontalAlignment="Center"
    Height="18"
    Margin="0"
    VerticalAlignment="Center"
    Width="19">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="20*" />
        <ColumnDefinition Width="20*" />
        <ColumnDefinition Width="20*" />
        <ColumnDefinition Width="20*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="23*" />
        <RowDefinition Height="19*" />
        <RowDefinition Height="19*" />
        <RowDefinition Height="19*" />
    </Grid.RowDefinitions>
    <Border x:Name="Highlight"
        BorderThickness="1"
        Grid.ColumnSpan="4"
        CornerRadius="0,0,1,1"
        Margin="-1"
        Opacity="1"
        Grid.Row="0"
        Grid.RowSpan="4">
        <Border.BorderBrush>
            <SolidColorBrush Color="{DynamicResource
ControlPressedColor}" />
        </Border.BorderBrush>
    </Border>
    <Border x:Name="Background"
        BorderBrush="#FFFFFF"

```

```

        BorderThickness="1"
        Grid.ColumnSpan="4"
        CornerRadius=".5"
        Margin="0,-1,0,0"
        Opacity="1"
        Grid.Row="1"
        Grid.RowSpan="3">
    <Border.Background>
        <SolidColorBrush Color="{DynamicResource ControlDarkColor}" />

```

```

    </Border.Background>
</Border>
<Border x:Name="BackgroundGradient"
        BorderBrush="#BF000000"
        BorderThickness="1"
        Grid.ColumnSpan="4"
        CornerRadius=".5"
        Margin="0,-1,0,0"
        Opacity="1"
        Grid.Row="1"
        Grid.RowSpan="3">
    <Border.Background>
        <LinearGradientBrush EndPoint=".7,1"
                            StartPoint=".7,0">
            <GradientStop Color="#FFFFFF"
                           Offset="0" />
            <GradientStop Color="#F9FFFFFF"
                           Offset="0.375" />
            <GradientStop Color="#E5FFFFFF"
                           Offset="0.625" />
            <GradientStop Color="#C6FFFFFF"
                           Offset="1" />
        </LinearGradientBrush>
    </Border.Background>
</Border>
<Rectangle Grid.ColumnSpan="4"
           Grid.RowSpan="1"
           StrokeThickness="1">
    <Rectangle.Fill>
        <LinearGradientBrush EndPoint="0,1"
                            StartPoint="0,0">
            <GradientStop Color="{DynamicResource HeaderTopColor}" />
            <GradientStop Color="{DynamicResource ControlMediumColor}"
                           Offset="1" />
        </LinearGradientBrush>
    </Rectangle.Fill>
    <Rectangle.Stroke>
        <LinearGradientBrush EndPoint="0.48,-1"
                            StartPoint="0.48,1.25">
            <GradientStop Color="#FF494949" />
            <GradientStop Color="#FF9F9F9F"
                           Offset="1" />
        </LinearGradientBrush>
    </Rectangle.Stroke>
</Rectangle>

```

```
<Path Fill="#FF2F2F2F"
      Grid.Row="1"
      Grid.Column="0"
      Grid.RowSpan="3"
      Grid.ColumnSpan="4"
      HorizontalAlignment="Center"
      VerticalAlignment="Center"
      RenderTransformOrigin="0.5,0.5"
      Margin="4,3,4,3"
      Stretch="Fill"
      Data="M11.426758,8.4305077 L11.749023,8.4305077
            L11.749023,16.331387 L10.674805,16.331387
            L10.674805,10.299648 L9.0742188,11.298672
            L9.0742188,10.294277 C9.4788408,10.090176
            9.9094238,9.8090878 10.365967,9.4510155
            C10.82251,9.0929432 11.176106,8.7527733
            11.426758,8.4305077 z M14.65086,8.4305077
            L18.566387,8.4305077 L18.566387,9.3435936
            L15.671368,9.3435936 L15.671368,11.255703
            C15.936341,11.058764 16.27293,10.960293
            16.681133,10.960293 C17.411602,10.960293
            17.969301,11.178717 18.354229,11.615566
            C18.739157,12.052416 18.931622,12.673672
            18.931622,13.479336 C18.931622,15.452317
            18.052553,16.438808 16.294415,16.438808
            C15.560365,16.438808 14.951641,16.234707
            14.468243,15.826504 L14.881817,14.929531
            C15.368796,15.326992 15.837872,15.525723
            16.289043,15.525723 C17.298809,15.525723
            17.803692,14.895514 17.803692,13.635098
            C17.803692,12.460618 17.305971,11.873379
            16.310528,11.873379 C15.83071,11.873379
            15.399232,12.079271 15.016094,12.491055
            L14.65086,12.238613 z" />
<Ellipse Grid.ColumnSpan="4"
          Fill="#FFFFFF"
          HorizontalAlignment="Center"
          Height="3"
          StrokeThickness="0"
          VerticalAlignment="Center"
          Width="3" />
<Border x:Name="DisabledVisual"
        BorderBrush="#B2FFFFFF"
        BorderThickness="1"
        Grid.ColumnSpan="4"
        CornerRadius="0,0,.5,.5"
        Opacity="0"
        Grid.Row="0"
        Grid.RowSpan="4" />
</Grid>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
```

```

<Style TargetType="{x:Type DatePicker}">
    <Setter Property="Foreground"
        Value="#FF333333" />
    <Setter Property="IsTodayHighlighted"
        Value="True" />
    <Setter Property="SelectedDateFormat"
        Value="Short" />
    <Setter Property="Padding"
        Value="2" />
    <Setter Property="BorderThickness"
        Value="1" />
    <Setter Property="HorizontalContentAlignment"
        Value="Stretch" />
    <!--Set CalendarStyle to DatePickerCalendarStyle.-->
    <Setter Property="CalendarStyle"
        Value="{DynamicResource DatePickerCalendarStyle}" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type DatePicker}">
                <Border BorderThickness="{TemplateBinding BorderThickness}"
                    Padding="{TemplateBinding Padding}">
                    <Border.BorderBrush>
                        <LinearGradientBrush EndPoint="0.5,1"
                            StartPoint="0.5,0">
                            <GradientStop Color="{DynamicResource BorderLightColor}"
                                Offset="0" />
                            <GradientStop Color="{DynamicResource BorderDarkColor}"
                                Offset="1" />
                        </LinearGradientBrush>
                    </Border.BorderBrush>
                    <Border.Background>
                        <LinearGradientBrush EndPoint="0.5,1"
                            StartPoint="0.5,0">
                            <GradientStop Color="{DynamicResource HeaderTopColor}"
                                Offset="0" />
                            <GradientStop Color="{DynamicResource ControlMediumColor}"
                                Offset="1" />
                        </LinearGradientBrush>
                    </Border.Background>
                </Border>
                <VisualStateManager.VisualStateGroups>
                    <VisualStateGroup x:Name="CommonStates">
                        <VisualState x:Name="Normal" />
                        <VisualState x:Name="Disabled">
                            <Storyboard>
                                <DoubleAnimation Duration="0"
                                    To="1"
                                    Storyboard.TargetProperty="Opacity" />
                                <Storyboard.TargetName="PART_DisabledVisual" />
                            </Storyboard>
                        </VisualState>
                    </VisualStateGroup>
                </VisualStateManager.VisualStateGroups>
            <Grid x:Name="PART_Root">

```

```
        HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}"
        VerticalAlignment="{TemplateBinding
VerticalContentAlignment}">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="Auto" />
    </Grid.ColumnDefinitions>
    <Button x:Name="PART_Button"
        Grid.Column="1"
        Foreground="{TemplateBinding Foreground}"
        Focusable="False"
        HorizontalAlignment="Left"
        Margin="3,0,3,0"
        Grid.Row="0"
        Style="{StaticResource DropDownButtonStyle}"
        VerticalAlignment="Top" />
    <DatePickerTextBox x:Name="PART_TextBox"
        Grid.Column="0"
        Foreground="{TemplateBinding Foreground}"
        Focusable="{TemplateBinding Focusable}"
        HorizontalContentAlignment="Stretch"
        Grid.Row="0"
        VerticalContentAlignment="Stretch" />
<Grid x:Name="PART_DisabledVisual"
    Grid.ColumnSpan="2"
    Grid.Column="0"
    IsHitTestVisible="False"
    Opacity="0"
    Grid.Row="0">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="Auto" />
    </Grid.ColumnDefinitions>
    <Rectangle Grid.Column="0"
        Fill="#A5FFFFFF"
        RadiusY="1"
        Grid.Row="0"
        RadiusX="1" />
    <Rectangle Grid.Column="1"
        Fill="#A5FFFFFF"
        Height="18"
        Margin="3,0,3,0"
        RadiusY="1"
        Grid.Row="0"
        RadiusX="1"
        Width="19" />
    <Popup x:Name="PART_Popup"
        AllowsTransparency="True"
        Placement="Bottom"
        PlacementTarget="{Binding ElementName=PART_TextBox}"
        StaysOpen="False" />
</Grid>
</Grid>
</Border>
```

```
    </ControlTemplate>
  </Setter.Value>
</Setter>
</Style>
```

上一示例使用了一个或多个以下资源。

XAML

```
<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>
```

```
<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

DocumentViewer 样式和模板

项目 • 2023/02/06

本主题介绍 DocumentViewer 控件的样式和模板。可以修改默认 ControlTemplate 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

DocumentViewer 部件

下表列出了 DocumentViewer 控件的命名部件。

组成部分	类型	描述
PART_ContentHost	ScrollViewer	内容和滚动区域。
PART_FindToolBarHost	ContentControl	搜索框，默认位于底部。

DocumentViewer 状态

下表列出了 DocumentViewer 控件的视觉对象状态。

VisualState 名称	VisualStateGroup 名称	描述
有效	ValidationStates	该控件使用 Validation 类，Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

DocumentViewer ControlTemplate 示例

以下示例演示如何定义 DocumentViewer 控件的 ControlTemplate。

XAML

```
<Style x:Key="{x:Type DocumentViewer}">
    TargetType="{x:Type DocumentViewer}"
    <Setter Property="Foreground"
        Value="{DynamicResource {x:Static SystemColors.WindowTextBrushKey}}"/>
```

```
<Setter Property="Background"
         Value="{DynamicResource {x:Static SystemColors.ControlBrushKey}}"
/>
<Setter Property="FocusVisualStyle"
         Value="{x:Null}" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type DocumentViewer}">
            <Border BorderThickness="{TemplateBinding BorderThickness}"
                    BorderBrush="{TemplateBinding BorderBrush}"
                    Focusable="False">
                <Grid KeyboardNavigation.TabNavigation="Local">
                    <Grid.Background>
                        <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
                    </Grid.Background>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto" />
                        <RowDefinition Height="*" />
                        <RowDefinition Height="Auto" />
                    </Grid.RowDefinitions>
                    <ToolBar ToolBarTray.IsLocked="True"
                            KeyboardNavigation.TabNavigation="Continue">
                        <Button Command="ApplicationCommands.Print"
                                CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
                                Content="Print" />
                        <Button Command="ApplicationCommands.Copy"
                                CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
                                Content="Copy" />
                        <Separator />
                        <Button Command="NavigationCommands.IncreaseZoom"
                                CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
                                Content="Zoom In" />
                        <Button Command="NavigationCommands.DecreaseZoom"
                                CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
                                Content="Zoom Out" />
                        <Separator />
                        <Button Command="NavigationCommands.Zoom"
                                CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
                                CommandParameter="100.0"
                                Content="Actual Size" />
                        <Button Command="DocumentViewer.FitToWidthCommand"
                                CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
                                Content="Fit to Width" />
                        <Button Command="DocumentViewer.FitToMaxPagesAcrossCommand"
                                CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
                                CommandParameter="1"
                                Content="Whole Page" />
                    </ToolBar>
                </Grid>
            </Border>
        </ControlTemplate>
    </Setter.Value>
</Setter>
```

```

        <Button Command="DocumentViewer.FitToMaxPagesAcrossCommand"
            CommandTarget="{Binding RelativeSource={RelativeSource
TemplatedParent}}"
            CommandParameter="2"
            Content="Two Pages" />
    </ToolBar>

    <ScrollViewer Grid.Row="1"
        CanContentScroll="true"
        HorizontalScrollBarVisibility="Auto"
        x:Name="PART_ContentHost"
        IsTabStop="true">
        <ScrollViewer.Background>
            <LinearGradientBrush EndPoint="0.5,1"
                StartPoint="0.5,0">
                <GradientStop Color="{DynamicResource ControlLightColor}"
                    Offset="0" />
                <GradientStop Color="{DynamicResource ControlMediumColor}"
                    Offset="1" />
            </LinearGradientBrush>
        </ScrollViewer.Background>
    </ScrollViewer>

    <ContentControl Grid.Row="2"
        x:Name="PART_FindToolBarHost"/>
    </Grid>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>

```

```

<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

Expander 样式和模板

项目 · 2023/02/06

本主题介绍 [Expander](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

Expander 部件

[Expander](#) 控件没有任何命名部件。

Expander 状态

下表列出了 [Expander](#) 控件可能的视觉对象状态。

VisualState 名称	VisualStateGroup 名称	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在控件上方。
已禁用	CommonStates	已禁用控件。
已设定焦点	FocusStates	控件有焦点。
失去焦点	FocusStates	控件没有焦点。
展开	ExpansionStates	该控件已展开。
Collapsed	ExpansionStates	该控件未展开。
ExpandDown	ExpandDirectionStates	该控件向下展开。
ExpandUp	ExpandDirectionStates	该控件向上展开。
ExpandLeft	ExpandDirectionStates	该控件向左展开。
ExpandRight	ExpandDirectionStates	该控件向右展开。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

Expander ControlTemplate 示例

下例演示如何定义 Expander 控件的 ControlTemplate。

XAML

```
<ControlTemplate x:Key="ExpanderToggleButton"
    TargetType="{x:Type ToggleButton}">
    <Border x:Name="Border"
        CornerRadius="2,0,0,0"
        BorderThickness="0,0,1,0">
        <Border.Background>
            <LinearGradientBrush EndPoint="0.5,1"
                StartPoint="0.5,0">
                <GradientStop Color="{DynamicResource ControlLightColor}" />
                <GradientStop Color="{DynamicResource ControlMediumColor}"
                    Offset="1" />
            </LinearGradientBrush>
        </Border.Background>
        <Border.BorderBrush>
            <LinearGradientBrush StartPoint="0,0"
                EndPoint="0,1">
                <LinearGradientBrush.GradientStops>
                    <GradientStopCollection>
                        <GradientStop Color="{DynamicResource BorderLightColor}"
                            Offset="0.0" />
                        <GradientStop Color="{DynamicResource BorderDarkColor}"
                            Offset="1.0" />
                    </GradientStopCollection>
                </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
        </Border.BorderBrush>
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CommonStates">
                <VisualState x:Name="Normal" />
                <VisualState x:Name="MouseOver">
                    <Storyboard>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                            Storyboard.TargetProperty="

(Panel.Background).(GradientBrush.GradientStops)[1].(GradientStop.Color)">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource

ControlMouseOverColor}" />
                        </ColorAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
                <VisualState x:Name="Pressed">
                    <Storyboard>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                            Storyboard.TargetProperty="

(Panel.Background).(GradientBrush.GradientStops)[1].(GradientStop.Color)">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource

ControlPressedColor}" />
                        </ColorAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
    </Border>
</ControlTemplate>
```

```

        (Panel.Background).
            (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                <EasingColorKeyFrame KeyTime="0"
                    Value="{StaticResource
ControlPressedColor}" />
            </ColorAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
    <VisualState x:Name="Disabled">
        <Storyboard>
            <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                Storyboard.TargetProperty="

        (Panel.Background).
            (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                <EasingColorKeyFrame KeyTime="0"
                    Value="{StaticResource
DisabledControlDarkColor}" />
            </ColorAnimationUsingKeyFrames>
            <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                Storyboard.TargetProperty="

        (Border.BorderBrush).
            (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                <EasingColorKeyFrame KeyTime="0"
                    Value="{StaticResource
DisabledBorderLightColor}" />
            </ColorAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
</VisualStateManager>
<VisualStateManager x:Name="CheckStates">
    <VisualState x:Name="Checked">
        <Storyboard>
            <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="

(UIElement.Visibility)"

        Storyboard.TargetName="CollapsedArrow">
            <DiscreteObjectKeyFrame KeyTime="0"
                Value="{x:Static Visibility.Hidden}">
        />
            </ObjectAnimationUsingKeyFrames>
            <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="

(UIElement.Visibility)">
                <DiscreteObjectKeyFrame KeyTime="0"
                    Value="{x:Static Visibility.Visible}">
            />
            </ObjectAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
    <VisualState x:Name="Unchecked" />
    <VisualState x:Name="Indeterminate" />
</VisualStateManager>
</VisualStateManagerGroups>
<Grid>

```

```

        <Path x:Name="CollapsedArrow"
              HorizontalAlignment="Center"
              VerticalAlignment="Center"
              Data="M 0 0 L 4 4 L 8 0 Z">
            <Path.Fill>
              <SolidColorBrush Color="{DynamicResource GlyphColor}" />
            </Path.Fill>
        </Path>
        <Path x:Name="ExpandedArrow"
              HorizontalAlignment="Center"
              VerticalAlignment="Center"
              Visibility="Collapsed"
              Data="M 0 4 L 4 0 L 8 4 Z">
            <Path.Fill>
              <SolidColorBrush Color="{DynamicResource GlyphColor}" />
            </Path.Fill>
        </Path>
      </Grid>
    </Border>
  </ControlTemplate>

<Style TargetType="{x:Type Expander}">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="{x:Type Expander}">
        <Grid>
          <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition x:Name="ContentRow"
                          Height="0" />
          </Grid.RowDefinitions>
          <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CommonStates">
              <VisualState x:Name="Normal" />
              <VisualState x:Name="MouseOver" />
              <VisualState x:Name="Disabled">
                <Storyboard
                  <ColorAnimationUsingKeyFrames
                    Storyboard.TargetName="Border"
                    Storyboard.TargetProperty=""
                    (Panel.Background).
                    (GradientBrush.GradientStops)[1].
                    (GradientStop.Color)">
                  <EasingColorKeyFrame KeyTime="0"
                    Value="{StaticResource
                    DisabledControlDarkColor}" />
                </ColorAnimationUsingKeyFrames>
                <ColorAnimationUsingKeyFrames
                  Storyboard.TargetName="Border"
                  Storyboard.TargetProperty=""
                  (Border.BorderBrush).
                  (GradientBrush.GradientStops)[1].
                  (GradientStop.Color)">
                  <EasingColorKeyFrame KeyTime="0"
                    Value="{StaticResource

```

```

        DisabledBorderLightColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border x:Name="Border"
    Grid.Row="0"
    BorderThickness="1"
    CornerRadius="2,2,0,0">
    <Border.BorderBrush>
        <LinearGradientBrush EndPoint="0,1"
            StartPoint="0,0">
            <GradientStop Color="{DynamicResource BorderLightColor}"
                Offset="0" />
            <GradientStop Color="{DynamicResource BorderDarkColor}"
                Offset="1" />
        </LinearGradientBrush>
    </Border.BorderBrush>
    <Border.Background>

        <LinearGradientBrush StartPoint="0,0"
            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource
ControlLightColor}"
                        Offset="0.0" />
                    <GradientStop Color="{DynamicResource
ControlMediumColor}"
                        Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Border.Background>
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="20" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <ToggleButton OverridesDefaultStyle="True"
            Template="{StaticResource ExpanderToggleButton}"
            IsChecked="{Binding IsExpanded, Mode=TwoWay,
RelativeSource={RelativeSource TemplatedParent}}">
            <ToggleButton.Background>
                <LinearGradientBrush EndPoint="0.5,1"
                    StartPoint="0.5,0">
                    <GradientStop Color="{DynamicResource
ControlLightColor}"
                        Offset="0" />
                    <GradientStop Color="{DynamicResource
ControlMediumColor}"
                        Offset="1" />
                </LinearGradientBrush>
            </ToggleButton.Background>
        </ToggleButton>
    </Grid>

```

```

        </LinearGradientBrush>
    </ToggleButton.Background>
</ToggleButton>
<ContentPresenter Grid.Column="1"
    Margin="4"
    ContentSource="Header"
    RecognizesAccessKey="True" />
</Grid>
</Border>
<Border x:Name="Content"
    Grid.Row="1"
    BorderThickness="1,0,1,1"
    CornerRadius="0,0,2,2">
    <Border.BorderBrush>
        <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
    </Border.BorderBrush>
    <Border.Background>
        <SolidColorBrush Color="{DynamicResource ContentAreaColorDark}" />
    </Border.Background>
    <ContentPresenter Margin="4" />
</Border>
</Grid>
<ControlTemplate.Triggers>
    <Trigger Property="IsExpanded"
        Value="True">
        <Setter TargetName="ContentRow"
            Property="Height"
            Value="{Binding Height, ElementName=Content}" />
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

```

```
<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
                     EndPoint="0.5,1"
                     StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
                  Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
                  Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
                  Offset="1" />

```

```
        Offset="1" />
    </GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

Frame 样式和模板

项目 • 2023/02/06

本主题介绍 [Frame 控件](#) 的样式和模板。 可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。 有关详细信息，请参阅[为控件创建模板](#)。

Frame 部件

下表列出了 [Frame 控件](#) 的已命名部件。

组成部分	类型	描述
PART_FrameCP	ContentPresenter	内容区域。

Frame 状态

下表列出了 [Frame 控件](#) 的视觉对象状态。

VisualState 名称	VisualStateGroup 名称	描述
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

Frame ControlTemplate 示例

下例演示如何定义 [Frame 控件](#) 的 [ControlTemplate](#)。

XAML

```
<!-- Back/Forward Button Style -->

<Style x:Key="FrameButtonStyle"
    TargetType="{x:Type Button}">
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
```

```

<Setter Property="Command"
        Value="NavigationCommands.BrowseBack" />
<Setter Property="Focusable"
        Value="false" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type Button}">
            <Grid>
                <VisualStateManager.VisualStateGroups>
                    <VisualStateGroup x:Name="CommonStates">
                        <VisualState x:Name="Normal" />
                        <VisualState x:Name="MouseOver">
                            <Storyboard>
                                <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Ellipse"
Storyboard.TargetProperty=""
(Shape.Fill).
                                (GradientBrush.GradientStops)[1].
(GradientStop.Color)">
                                    <EasingColorKeyFrame KeyTime="0"
                                            Value="{StaticResource
ControlMouseOverColor}" />
                                    </ColorAnimationUsingKeyFrames>
                            </Storyboard>
                        </VisualState>
                        <VisualState x:Name="Pressed">
                            <Storyboard>
                                <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Ellipse"
Storyboard.TargetProperty=""
(Shape.Fill).
                                (GradientBrush.GradientStops)[1].
(GradientStop.Color)">
                                    <EasingColorKeyFrame KeyTime="0"
                                            Value="{StaticResource
ControlPressedColor}" />
                                    </ColorAnimationUsingKeyFrames>
                            </Storyboard>
                        </VisualState>
                        <VisualState x:Name="Disabled">
                            <Storyboard>
                                <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Ellipse"
Storyboard.TargetProperty=""
(Shape.Fill).
                                (GradientBrush.GradientStops)[1].
(GradientStop.Color)">
                                    <EasingColorKeyFrame KeyTime="0"
                                            Value="{StaticResource
DisabledControlDarkColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Arrow"
Storyboard.TargetProperty=""
(Shape.Fill).>

```

```

        (SolidColorBrush.Color)">
        <EasingColorKeyFrame KeyTime="0"
                             Value="{StaticResource
DisabledForegroundColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Ellipse x:Name="Ellipse"
         StrokeThickness="1"
         Width="16"
         Height="16">
    <Ellipse.Stroke>
        <SolidColorBrush Color="{DynamicResource NavButtonFrameColor}" />
    </Ellipse.Stroke>
    <Ellipse.Fill>
        <LinearGradientBrush StartPoint="0,0"
                             EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource
ControlLightColor}" />
                    <GradientStop Color="{DynamicResource
ControlMediumColor}" />
                    <GradientStop Color="Black" Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Ellipse.Fill>
</Ellipse>

<Path x:Name="Arrow"
      Margin="0,0,2,0"
      HorizontalAlignment="Center"
      VerticalAlignment="Center"
      Data="M 4 0 L 0 4 L 4 8 Z" >
    <Path.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Fill>
</Path>
</Grid>
<ControlTemplate.Triggers>
    <Trigger Property="Command"
             Value="{x:Static NavigationCommands.BrowseForward}">
        <Setter TargetName="Arrow"
               Property="Data"
               Value="M 0 0 L 4 4 L 0 8 z" />
        <Setter TargetName="Arrow"
               Property="Margin"
               Value="2,0,0,0" />
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>

```

```

        </Setter.Value>
    </Setter>
</Style>

<!-- Frame Menu Style -->

<Style x:Key="FrameMenu"
    TargetType="{x:Type Menu}">
<Setter Property="OverridesDefaultStyle"
    Value="true" />
<Setter Property="KeyboardNavigation.TabNavigation"
    Value="None" />
<Setter Property="IsMainMenu"
    Value="false" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type Menu}">
            <DockPanel IsItemsHost="true" />
        </ControlTemplate>
    </Setter.Value>
</Setter>
</Style>

<!-- Frame Menu Header Style -->

<Style x:Key="FrameHeaderMenuItem"
    TargetType="{x:Type MenuItem}">
<Setter Property="OverridesDefaultStyle"
    Value="true" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type MenuItem}">
            <Grid>
                <Popup x:Name="PART_Popup"
                    Placement="Bottom"
                    VerticalOffset="2"
                    IsOpen="{TemplateBinding IsSubmenuOpen}"
                    AllowsTransparency="True"
                    Focusable="False"
                    PopupAnimation="Fade">
                    <Border x:Name="SubMenuBorder"
                        BorderThickness="1"
                        Background="{DynamicResource MenuPopupBrush}">
                        <Border.BorderBrush>
                            <SolidColorBrush Color="{StaticResource BorderMediumColor}" />
                    </Border.BorderBrush>
                    <StackPanel IsItemsHost="true"
                        Margin="2"
                        KeyboardNavigation.TabNavigation="Cycle"
                        KeyboardNavigation.DirectionalNavigation="Cycle" />
                </Popup>
            </Grid>
        </ControlTemplate>
    </Setter.Value>
</Setter>
</Style>

```

```

<Grid x:Name="Panel"
      Width="24"
      Background="Transparent"
      HorizontalAlignment="Right">

    <Border Visibility="Hidden"
            x:Name="HighlightBorder"
            BorderThickness="1"
            CornerRadius="2">
      <Border.BorderBrush>
        <LinearGradientBrush StartPoint="0,0"
                             EndPoint="0,1">
          <LinearGradientBrush.GradientStops>
            <GradientStopCollection>
              <GradientStop Color="{DynamicResource
BorderLightColor}" Offset="0.0" />
              <GradientStop Color="{DynamicResource
BorderDarkColor}" Offset="1.0" />
            </GradientStopCollection>
          </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>

      </Border.BorderBrush>
      <Border.Background>
        <LinearGradientBrush StartPoint="0,0"
                             EndPoint="0,1">
          <LinearGradientBrush.GradientStops>
            <GradientStopCollection>
              <GradientStop Color="{DynamicResource
ControlLightColor}" />
              <GradientStop Color="{DynamicResource
ControlMouseOverColor}" Offset="1.0" />
            </GradientStopCollection>
          </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>

      </Border.Background>
    </Border>
    <Path x:Name="Arrow"
          SnapsToDevicePixels="false"
          HorizontalAlignment="Right"
          VerticalAlignment="Center"
          Margin="0,2,4,0"
          StrokeLineJoin="Round"
          Data="M 0 0 L 4 4 L 8 0 Z">
      <Path.Stroke>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
      </Path.Stroke>
    </Path>
  </Grid>
</Grid>
<ControlTemplate.Triggers>

```

```
<Trigger Property="IsHighlighted"
    Value="true">
    <Setter TargetName="HighlightBorder"
        Property="Visibility"
        Value="Visible" />
</Trigger>
<Trigger Property="Is_submenuOpen"
    Value="true">
    <Setter TargetName="HighlightBorder"
        Property="BorderBrush">
        <Setter.Value>
            <LinearGradientBrush StartPoint="0,0"
                EndPoint="0,1">
                <GradientBrush.GradientStops>
                    <GradientStopCollection>
                        <GradientStop Color="{DynamicResource
BorderDarkColor}"
                            Offset="0.0" />
                        <GradientStop Color="{DynamicResource
BorderMediumColor}"
                            Offset="1.0" />
                    </GradientStopCollection>
                </GradientBrush.GradientStops>
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
    <Setter Property="Background"
        TargetName="HighlightBorder">
        <Setter.Value>
            <LinearGradientBrush EndPoint="0,1"
                StartPoint="0,0">
                <GradientStop Color="{DynamicResource ControlLightColor}"
                    Offset="0" />
                <GradientStop Color="{DynamicResource
ControlPressedColor}"
                    Offset="0.984" />
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
</Trigger>
<Trigger Property="IsEnabled"
    Value="false">
    <Setter TargetName="Arrow"
        Property="Fill">
        <Setter.Value>
            <SolidColorBrush Color="{DynamicResource
DisabledForegroundColor}" />
        </Setter.Value>
    </Setter>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
```

```
</Style>

<!-- Frame Menu Item Style -->

<Style x:Key="FrameSubmenuItem"
       TargetType="{x:Type MenuItem}"
       >
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="Header"
           Value="{Binding (JournalEntry.Name)}" />
    <Setter Property="Command"
           Value="NavigationCommands.NavigateJournal" />
    <Setter Property="CommandTarget"
           Value="{Binding TemplatedParent,
             RelativeSource={RelativeSource AncestorType={x:Type Menu}}}" />
    <Setter Property="CommandParameter"
           Value="{Binding RelativeSource={RelativeSource Self}}" />
    <Setter Property="JournalEntryUnifiedViewConverter.JournalEntryPosition"
           Value="{Binding
             (JournalEntryUnifiedViewConverter.JournalEntryPosition)}" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type MenuItem}">
                <Border BorderThickness="1"
                       Name="Border">
                    <Grid x:Name="Panel"
                          Background="Transparent"
                          SnapsToDevicePixels="true"
                          Height="35"
                          Width="250">
                        <Path x:Name="Glyph"
                              SagsToDevicePixels="false"
                              Margin="7,5"
                              Width="10"
                              Height="10"
                              HorizontalAlignment="Left"
                              StrokeStartLineCap="Triangle"
                              StrokeEndLineCap="Triangle"
                              StrokeThickness="2">
                            <Path.Stroke>
                                <SolidColorBrush Color="{DynamicResource GlyphColor}" />
                            </Path.Stroke>
                        </Path>
                        <ContentPresenter ContentSource="Header"
                                          Margin="24,5,50,5" />
                    </Grid>
                </Border>
                <ControlTemplate.Triggers>
                    <Trigger
Property="JournalEntryUnifiedViewConverter.JournalEntryPosition"
Value="Current">
                        <Setter TargetName="Glyph"
                               Property="Data"
                               Value="M 0,5 L 2.5,8 L 7,3 " />
                    </Trigger>
                </ControlTemplate.Triggers>
            </ControlTemplate>
        </Setter.Value>
    </Setter>

```

```

        </Trigger>
        <Trigger Property="IsHighlighted"
            Value="true">
            <Setter Property="Background"
                TargetName="Border">
                <Setter.Value>
                    <LinearGradientBrush EndPoint="0.5,1"
                        StartPoint="0.5,0">
                        <GradientStop Color="Transparent"
                            Offset="0" />
                        <GradientStop Color="{DynamicResource
ControlMouseOverColor}"
                            Offset="1" />
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
            <Setter Property="BorderBrush"
                TargetName="Border">
                <Setter.Value>
                    <LinearGradientBrush EndPoint="0.5,1"
                        StartPoint="0.5,0">
                        <GradientStop Color="{DynamicResource BorderMediumColor}"
                            Offset="0" />
                        <GradientStop Color="Transparent"
                            Offset="1" />
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
        </Trigger>
        <MultiTrigger>
            <MultiTrigger.Conditions>
                <Condition Property="IsHighlighted"
                    Value="true" />
            <Condition
Property="JournalEntryUnifiedViewConverter.JournalEntryPosition"
                    Value="Forward" />
            </MultiTrigger.Conditions>
            <Setter TargetName="Glyph"
                Property="Data"
                Value="M 3 1 L 7 5 L 3 9 z" />
            <Setter TargetName="Glyph"
                Property="Stroke"
                Value="{x:Null}" />
            <Setter TargetName="Glyph"
                Property="Fill">
                <Setter.Value>
                    <SolidColorBrush Color="{StaticResource GlyphColor}" />
                </Setter.Value>
            </Setter>
        </MultiTrigger>
        <MultiTrigger>
            <MultiTrigger.Conditions>
                <Condition Property="IsHighlighted"
                    Value="true" />
            <Condition

```

```

Property="JournalEntryUnifiedViewConverter.JournalEntryPosition"
    Value="Back" />
</MultiTrigger.Conditions>
<Setter TargetName="Glyph"
    Property="Data"
    Value="M 7 1 L 3 5 L 7 9 z" />
<Setter TargetName="Glyph"
    Property="Stroke"
    Value="{x:Null}" />
<Setter TargetName="Glyph"
    Property="Fill">
    <Setter.Value>
        <SolidColorBrush Color="{StaticResource GlyphColor}" />
    </Setter.Value>
    </Setter>
</MultiTrigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!-- Merges Back and Forward Navigation Stacks --&gt;

&lt;JournalEntryUnifiedViewConverter x:Key="JournalEntryUnifiedViewConverter"
/&gt;

<!-- SimpleStyles: Frame --&gt;

&lt;Style x:Key="{x:Type Frame}"
    TargetType="{x:Type Frame}"&gt;
&lt;Setter Property="SnapsToDevicePixels"
    Value="true" /&gt;
&lt;Setter Property="Template"&gt;
    &lt;Setter.Value&gt;
        &lt;ControlTemplate TargetType="{x:Type Frame}"&gt;
            &lt;DockPanel&gt;
                &lt;Border DockPanel.Dock="Top"
                    Height="22"
                    BorderThickness="1"&gt;
                    &lt;Border.BorderBrush&gt;
                        &lt;LinearGradientBrush EndPoint="0.5,1"
                            StartPoint="0.5,0"&gt;
                            &lt;GradientStop Color="{DynamicResource BorderLightColor}"
                                Offset="0" /&gt;
                            &lt;GradientStop Color="{DynamicResource BorderDarkColor}"
                                Offset="1" /&gt;
                        &lt;/LinearGradientBrush&gt;
                    &lt;/Border.BorderBrush&gt;
                &lt;Grid&gt;
                    &lt;Grid.Background&gt;
                        &lt;LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1"&gt;
                            &lt;LinearGradientBrush.GradientStops&gt;
</pre>

```

```

        <GradientStopCollection>
            <GradientStop Color="{DynamicResource
ControlLightColor}"
                           Offset="0.0" />
            <GradientStop Color="{DynamicResource
ControlMediumColor}"
                           Offset="1.0" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

</Grid.Background>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="16" />
    <ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>

<Menu x:Name="NavMenu"
      Grid.ColumnSpan="3"
      Height="16"
      Margin="1,0,0,0"
      VerticalAlignment="Center"
      Style="{StaticResource FrameMenu}">
    <MenuItem Style="{StaticResource FrameHeaderMenuItem}"
              ItemContainerStyle="{StaticResource
FrameSubmenuItem}"
              IsSubmenuOpen="{Binding (MenuItem.IsSubmenuOpen),
Mode=TwoWay, RelativeSource={RelativeSource
TemplatedParent}}">
        <MenuItem.ItemsSource>
            <MultiBinding Converter="{StaticResource
JournalEntryUnifiedViewConverter}">
                <Binding RelativeSource="{RelativeSource
TemplatedParent}">
                    Path="BackStack" />
                <Binding RelativeSource="{RelativeSource
TemplatedParent}">
                    Path="ForwardStack" />
            </MultiBinding>
        </MenuItem.ItemsSource>
    </MenuItem>
</Menu>

<Path Grid.Column="0"
      SnapsToDevicePixels="false"
      IsHitTestVisible="false"
      Margin="2,1.5,0,1.5"
      Grid.ColumnSpan="3"
      StrokeThickness="1"
      HorizontalAlignment="Left"
      VerticalAlignment="Center"
      Data="M15,14 Q18,12.9 20.9,14 A8.3,8.3,0,0,0,35.7,8.7
A8.3,8.3,0,0,0,>

```

```

      25.2,0.6 Q18, 3.3 10.8,0.6 A8.3,8.3,0,0,0,0.3,8.7
A8.3,8.3,0,0,0,15,14 z"
    Stroke="{x:Null}">
  <Path.Fill>
    <LinearGradientBrush EndPoint="0.5,1"
      StartPoint="0.5,0">
      <GradientStop Color="{DynamicResource
ControlMediumColor}">
        Offset="0" />
      <GradientStop Color="{DynamicResource ControlDarkColor}">
        Offset="1" />
    </LinearGradientBrush>
  </Path.Fill>
</Path>
<Button Style="{StaticResource FrameButtonStyle}"
  Command="NavigationCommands.BrowseBack"
  Content="M 4 0 L 0 4 L 4 8 Z"
  Margin="2.7,1.5,1.3,1.5"
  Grid.Column="0" />
<Button Style="{StaticResource FrameButtonStyle}"
  Command="NavigationCommands.BrowseForward"
  Content="M 4 0 L 0 4 L 4 8 Z"
  Margin="1.3,1.5,0,1.5"
  Grid.Column="1" />
</Grid>
</Border>
<Border BorderThickness="1">
  <Border.BorderBrush>
    <SolidColorBrush Color="{DynamicResource BorderMediumColor}">
  />
  </Border.BorderBrush>
<ContentPresenter x:Name="PART_FrameCP"
  Height="458"
  Width="640" />
</Border>
</DockPanel>
<ControlTemplate.Triggers>
  <MultiTrigger>
    <MultiTrigger.Conditions>
      <Condition Property="CanGoForward"
        Value="false" />
      <Condition Property="CanGoBack"
        Value="false" />
    </MultiTrigger.Conditions>
    <Setter TargetName="NavMenu"
      Property="IsEnabled"
      Value="false" />
  </MultiTrigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```
<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}">
```

```
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

GroupBox 样式和模板

项目 • 2023/02/06

本主题介绍 [GroupBox](#) 控件的样式和模板。 可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。 有关详细信息，请参阅[为控件创建模板](#)。

GroupBox 部件

[GroupBox](#) 控件没有任何已命名的部件。

GroupBox 状态

下表列出了 [GroupBox](#) 控件的可视状态。

VisualState 名称 名称	VisualStateGroup	描述
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。

GroupBox ControlTemplate 示例

以下示例演示如何定义 [GroupBox](#) 控件的 [ControlTemplate](#)。

XAML

```
<Style TargetType="GroupBox">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="GroupBox">
                <Grid>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto" />
                        <RowDefinition Height="*" />
                    </Grid.RowDefinitions>
                    <Border Grid.Row="0"
                            BorderThickness="1"
                            CornerRadius="2,2,0,0">
```

```

<Border.BorderBrush>
    <LinearGradientBrush StartPoint="0,0"
                           EndPoint="0,1">
        <LinearGradientBrush.GradientStops>
            <GradientStopCollection>
                <GradientStop Color="{DynamicResource BorderLightColor}"
                               Offset="0.0" />
                <GradientStop Color="{DynamicResource BorderDarkColor}"
                               Offset="1.0" />
            </GradientStopCollection>
        </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
</Border.BorderBrush>

<Border.Background>
    <LinearGradientBrush StartPoint="0,0"
                           EndPoint="0,1">
        <LinearGradientBrush.GradientStops>
            <GradientStopCollection>
                <GradientStop Color="{DynamicResource
ControlLightColor}"
                               Offset="0.0" />
                <GradientStop Color="{DynamicResource
ControlMediumColor}"
                               Offset="1.0" />
            </GradientStopCollection>
        </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
</Border.Background>

<ContentPresenter Margin="4"
                  ContentSource="Header"
                  RecognizesAccessKey="True" />
</Border>

<Border Grid.Row="1"
        BorderThickness="1,0,1,1"
        CornerRadius="0,0,2,2">
    <Border.BorderBrush>
        <SolidColorBrush Color="{StaticResource BorderMediumColor}" />
    </Border.BorderBrush>
    <Border.Background>
        <LinearGradientBrush EndPoint="0.5,1"
                           MappingMode="RelativeToBoundingBox"
                           StartPoint="0.5,0">
            <GradientStop Color="{DynamicResource
ContentAreaColorLight}"
                           Offset="0" />
            <GradientStop Color="{DynamicResource ContentAreaColorDark}"
                           Offset="1" />
        </LinearGradientBrush>
    </Border.Background>
    <ContentPresenter Margin="4" />
</Border>
</Grid>

```

```
    </ControlTemplate>
  </Setter.Value>
</Setter>
</Style>
```

ControlTemplate 使用了一个或多个以下资源。

XAML

```
<!--Control colors-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>
```

```
<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

Label 样式和模板

项目 • 2023/02/06

本主题介绍 [Label](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

Label 部件

[Label](#) 控件没有任何命名的部件。

Label 状态

下表列出了 [Label](#) 控件的视觉对象状态。

VisualState 名称 名称	VisualStateGroup	描述
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	如果控件没有焦点， Validation.HasError 附加属性为 <code>true</code> 。

Label ControlTemplate 示例

以下示例演示如何定义 [Label](#) 控件的 [ControlTemplate](#)。

XAML

```
<Style x:Key="{x:Type Label}"  
       TargetType="Label">  
    <Setter Property="HorizontalContentAlignment"  
           Value="Left" />  
    <Setter Property="VerticalContentAlignment"  
           Value="Top" />  
    <Setter Property="Template">  
        <Setter.Value>  
            <ControlTemplate TargetType="Label">  
                <Border>  
                    <ContentPresenter HorizontalAlignment="{TemplateBinding  
HorizontalContentAlignment}">
```

```

        VerticalAlignment="{TemplateBinding
VerticalContentAlignment}"
                    RecognizesAccessKey="True" />
    </Border>
    <ControlTemplate.Triggers>
        <Trigger Property="IsEnabled"
Value="false">
            <Setter Property="Foreground">
                <Setter.Value>
                    <SolidColorBrush Color="{DynamicResource
DisabledForegroundColor}" />
                </Setter.Value>
            </Setter>
        </Trigger>
    </ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

ControlTemplate 使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

```

```

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)

- 样式设置和模板化
- 创建控件模板

ListBox 样式和模板

项目 • 2023/02/06

本主题介绍 [ListBox](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

ListBox 部件

[ListBox](#) 控件没有任何命名的部件。

为 [ListBox](#) 创建一个 [ControlTemplate](#) 时，你的模板可能在 [ScrollViewer](#) 中包含一个 [ItemsPresenter](#)。（[ItemsPresenter](#) 显示 [ListBox](#) 中的每个项；[ScrollViewer](#) 支持在控件内滚动）。如果 [ItemsPresenter](#) 不是 [ScrollViewer](#) 的直接子元素，则必须为 [ItemsPresenter](#) 指定名称 `ItemsPresenter`。

ListBox 状态

下表列出了 [ListBox](#) 控件的视觉对象状态。

VisualStyle 名称	VisualStyleGroup 名称	描述
有效	ValidationStates	控件有效。
InvalidFocused	ValidationStates	控件无效，但具有焦点。
InvalidUnfocused	ValidationStates	控件无效，并且没有焦点。

ListBoxItem 部件

[ListBoxItem](#) 控件没有任何命名的部件。

ListBoxItem 状态

下表列出了 [ListBox](#) 控件的视觉对象状态。

VisualStyle 名称	VisualStyleGroup 名称	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在控件上方。

VisualState 名称	VisualStateGroup 名称	描述
已禁用	CommonStates	该项已禁用。
已设定焦点	FocusStates	该项有焦点。
失去焦点	FocusStates	该项没有焦点。
未选定	SelectionStates	未选定该项。
选定	SelectionStates	该项当前已选定。
SelectedUnfocused	SelectionStates	该项已选定，但没有焦点。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

ListBox ControlTemplate 示例

下面的示例演示如何为 [ListBox](#) 和 [ListBoxItem](#) 控件定义 [ControlTemplate](#)。

XAML

```
<Style x:Key="{x:Type ListBox}"
       TargetType="ListBox">
    <Setter Property="SnapsToDevicePixels"
           Value="true" />
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="ScrollViewer.HorizontalScrollBarVisibility"
           Value="Auto" />
    <Setter Property="ScrollViewer.VerticalScrollBarVisibility"
           Value="Auto" />
    <Setter Property="ScrollViewer.CanContentScroll"
           Value="true" />
    <Setter Property="MinWidth"
           Value="120" />
    <Setter Property="MinHeight"
           Value="95" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="ListBox">
                <Border Name="Border"
                       BorderThickness="1"
```



```

<Border.Background>
    <SolidColorBrush Color="Transparent" />
</Border.Background>
<VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="SelectionStates">
        <VisualState x:Name="Unselected" />
        <VisualState x:Name="Selected">
            <Storyboard
                <ColorAnimationUsingKeyFrames
                    Storyboard.TargetName="Border"
                    Storyboard.TargetProperty="Background">
                <EasingColorKeyFrame KeyTime="0"
                    Value="{StaticResource
                        SelectedBackgroundColor}" />
                </ColorAnimationUsingKeyFrames>
            </Storyboard>
        </VisualState>
        <VisualState x:Name="SelectedUnfocused">
            <Storyboard
                <ColorAnimationUsingKeyFrames
                    Storyboard.TargetName="Border"
                    Storyboard.TargetProperty="Background">
                <EasingColorKeyFrame KeyTime="0"
                    Value="{StaticResource
                        SelectedUnfocusedColor}" />
                </ColorAnimationUsingKeyFrames>
            </Storyboard>
        </VisualState>
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<ContentPresenter />
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

```

```

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"

```

```
        Offset="0.4" />
<GradientStop Color="#600000FF"
               Offset="0.6" />
<GradientStop Color="#000000FF"
               Offset="1" />
</GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

ListView 样式和模板

项目 · 2023/02/06

本主题介绍 [ListView](#) 控件的样式和模板。 可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。 有关详细信息，请参阅[为控件创建模板](#)。

ListView 部件

[ListView](#) 控件没有任何命名的部件。

为 [ListView](#) 创建一个 [ControlTemplate](#) 时，你的模板可能在 [ScrollViewer](#) 中包含一个 [ItemsPresenter](#)。（[ItemsPresenter](#) 显示 [ListView](#) 中的每个项；[ScrollViewer](#) 支持在控件内滚动）。如果 [ItemsPresenter](#) 不是 [ScrollViewer](#) 的直接子元素，则必须为 [ItemsPresenter](#) 指定名称 `ItemsPresenter`。

ListView 状态

下表列出了 [ListView](#) 控件的可视状态。

VisualStyle 名称 名称	VisualStyleGroup 名称	描述
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	如果控件不具有焦点，则 Validation.HasError 附加属性为 <code>true</code> 。

ListViewItem 部件

[ListViewItem](#) 控件没有任何命名的部件。

ListViewItem 状态

下表列出了 [ListViewItem](#) 控件的状态。

VisualStyle 名称 名称	VisualStyleGroup 名称	描述

VisualState 名称	VisualStateGroup 名称	描述
普通	CommonStates	默认状态。
已禁用	CommonStates	已禁用控件。
MouseOver	CommonStates	鼠标指针位于 ComboBox 控件上。
已设定焦点	FocusStates	控件有焦点。
失去焦点	FocusStates	控件没有焦点。
选定	SelectionStates	该项当前已选定。
未选定	SelectionStates	未选定该项。
SelectedUnfocused	SelectionStates	该项已选定，但没有焦点。
有效	ValidationStates	该控件使用 Validation 类，Validation.HasError 附加属性为 false。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 true。
InvalidUnfocused	ValidationStates	如果控件不具有焦点，则 Validation.HasError 附加属性为 true。

ListView ControlTemplate 示例

下例演示如何定义 ListView 控件的 ControlTemplate 及其关联类型。

XAML

```
<Style x:Key="{x:Static GridView.GridViewScrollViewerStyleKey}"
       TargetType="ScrollViewer">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="ScrollViewer">
                <Grid Background="{TemplateBinding Background}">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="Auto" />
                    </Grid.ColumnDefinitions>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="*" />
                        <RowDefinition Height="Auto" />
                    </Grid.RowDefinitions>

                    <DockPanel Margin="{TemplateBinding Padding}">
                        <ScrollViewer DockPanel.Dock="Top" />
                    </DockPanel>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

```

        HorizontalScrollBarVisibility="Hidden"
        VerticalScrollBarVisibility="Hidden"
        Focusable="false">
    <GridViewHeaderRowPresenter Margin="2,0,2,0"
        Columns="{Binding
Path=TemplatedParent.View.Columns,
RelativeSource={RelativeSource TemplatedParent}}}"
        ColumnHeaderContainerStyle="

{Binding
Path=TemplatedParent.View.ColumnHeaderContainerStyle,
RelativeSource={RelativeSource TemplatedParent}}"
        ColumnHeaderTemplate="{Binding
Path=TemplatedParent.View.ColumnHeaderTemplate,
RelativeSource={RelativeSource TemplatedParent}}"
        ColumnHeaderTemplateSelector="

{Binding
Path=TemplatedParent.View.ColumnHeaderTemplateSelector,
RelativeSource={RelativeSource TemplatedParent}}"
        AllowsColumnReorder="

{Binding
Path=TemplatedParent.ViewAllowsColumnReorder,
RelativeSource={RelativeSource TemplatedParent}}"
        ColumnHeaderContextMenu="

{Binding
Path=TemplatedParent.View.ColumnHeaderContextMenu,
RelativeSource={RelativeSource TemplatedParent}}"
        ColumnHeaderToolTip="

{Binding
Path=TemplatedParent.View.ColumnHeaderToolTip,
RelativeSource={RelativeSource TemplatedParent}}"
        SnapsToDevicePixels="

{TemplateBinding
SnapsToDevicePixels}" />
</ScrollViewer>

<ScrollContentPresenter Name="PART_ScrollContentPresenter"

KeyboardNavigation.Directionality="Local"
        CanContentScroll="True"
        CanHorizontallyScroll="False"
        CanVerticallyScroll="False" />
</DockPanel>

<ScrollBar Name="PART_HorizontalScrollBar"
        Orientation="Horizontal"
        Grid.Row="1"
        Maximum="{TemplateBinding ScrollableWidth}"
        ViewportSize="{TemplateBinding ViewportWidth}"
        Value="{TemplateBinding HorizontalOffset}"
        Visibility="{TemplateBinding

ComputedHorizontalScrollBarVisibility}" />

<ScrollBar Name="PART_VerticalScrollBar"
        Grid.Column="1"
        Maximum="{TemplateBinding ScrollableHeight}"
        ViewportSize="{TemplateBinding ViewportHeight}"
        Value="{TemplateBinding VerticalOffset}"
        Visibility="{TemplateBinding

```

```

        ComputedVerticalScrollBarVisibility}" />

    </Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="GridViewColumnHeaderGripper"
    TargetType="Thumb">
<Setter Property="Width"
    Value="18" />
<Setter Property="Background">
<Setter.Value>
    <LinearGradientBrush StartPoint="0,0"
        EndPoint="0,1">
        <LinearGradientBrush.GradientStops>
            <GradientStopCollection>
                <GradientStop Color="{DynamicResource BorderLightColor}"
                    Offset="0.0" />
                <GradientStop Color="{DynamicResource BorderDarkColor}"
                    Offset="1.0" />
            </GradientStopCollection>
        </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
</Setter.Value>
</Setter>
<Setter Property="Template">
<Setter.Value>
    <ControlTemplate TargetType="{x:Type Thumb}">
        <Border Padding="{TemplateBinding Padding}"
            Background="Transparent">
            <Rectangle HorizontalAlignment="Center"
                Width="1"
                Fill="{TemplateBinding Background}" />
        </Border>
    </ControlTemplate>
</Setter.Value>
</Setter>
<Setter Property="BorderBrush">
<Setter.Value>
    <LinearGradientBrush EndPoint="0.5,1"
        StartPoint="0.5,0">
        <GradientStop Color="Black"
            Offset="0" />
        <GradientStop Color="White"
            Offset="1" />
    </LinearGradientBrush>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="{x:Type GridViewColumnHeader}"
    TargetType="GridViewColumnHeader">
<Setter Property="HorizontalContentAlignment"

```

```

        Value="Center" />
<Setter Property="VerticalContentAlignment"
        Value="Center" />
<Setter Property="Foreground"
        Value="{DynamicResource {x:Static
SystemColors.ControlTextBrushKey}}" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="GridViewColumnHeader">
            <Grid>
                <VisualStateManager.VisualStateGroups>
                    <VisualStateGroup x:Name="CommonStates">
                        <VisualState x:Name="Normal" />
                        <VisualState x:Name="MouseOver">
                            <Storyboard>
                                <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="
(Panel.Background)."
(GradientBrush.GradientStops)[1].(GradientStop.Color)">
                                    Storyboard.TargetName="HeaderBorder">
                                        <EasingColorKeyFrame KeyTime="0"
Value="{StaticResource
ControlMouseOverColor}" />
                                            </ColorAnimationUsingKeyFrames>
                                        </Storyboard>
                                    </VisualState>
                                    <VisualState x:Name="Pressed" />
                                    <VisualState x:Name="Disabled" />
                                </VisualStateGroup>
                            </VisualStateManager.VisualStateGroups>
                            <Border x:Name="HeaderBorder"
BorderThickness="0,1,0,1"
Padding="2,0,2,0">
                                <Border.BorderBrush>
                                    <LinearGradientBrush StartPoint="0,0"
EndPoint="0,1">
                                        <LinearGradientBrush.GradientStops>
                                            <GradientStopCollection>
                                                <GradientStop Color="{DynamicResource BorderLightColor}"
Offset="0.0" />
                                                <GradientStop Color="{DynamicResource BorderDarkColor}"
Offset="1.0" />
                                            </GradientStopCollection>
                                        </LinearGradientBrush.GradientStops>
                                    </LinearGradientBrush>
                                </Border.BorderBrush>
                                <Border.Background>
                                    <LinearGradientBrush StartPoint="0,0"
EndPoint="0,1">
                                        <LinearGradientBrush.GradientStops>
                                            <GradientStopCollection>
                                                <GradientStop Color="{DynamicResource
ControlLightColor}" />
                                            </GradientStopCollection>
                                        </LinearGradientBrush.GradientStops>
                                    </LinearGradientBrush>
                                </Border.Background>
                            </Border>
                        </VisualState>
                    </VisualStateGroup>
                </VisualStateManager.VisualStateGroups>
            </Grid>
        </ControlTemplate>
    </Setter.Value>
</Setter>

```

```

                Offset="0.0" />
            <GradientStop Color="{DynamicResource ControlMediumColor}">
                Offset="1.0" />
            </GradientStopCollection>
        </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>

</Border.Background>
<ContentPresenter x:Name="HeaderContent"
    Margin="0,0,0,1"
    RecognizesAccessKey="True"
    VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
    HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
    SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}"/>
</Border>
<Thumb x:Name="PART_HeaderGripper"
    HorizontalAlignment="Right"
    Margin="0,0,-9,0"
    Style="{StaticResource GridViewColumnHeaderGripper}" />
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
<Style.Triggers>
    <Trigger Property="Role"
        Value="Floating">
        <Setter Property="Opacity"
            Value="0.7" />
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="GridViewColumnHeader">
                    <Canvas Name="PART_FloatingHeaderCanvas">
                        <Rectangle Fill="#60000000"
                            Width="{TemplateBinding ActualWidth}"
                            Height="{TemplateBinding ActualHeight}" />
                    </Canvas>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Trigger>
    <Trigger Property="Role"
        Value="Padding">
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="GridViewColumnHeader">
                    <Border Name="HeaderBorder"
                        BorderThickness="0,1,0,1">
                        <Border.Background>
                            <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
                        </Border.Background>
                    </Border>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Trigger>
</Style.Triggers>

```

```

        <Border.BorderBrush>
            <LinearGradientBrush StartPoint="0,0"
                EndPoint="0,1">
                <LinearGradientBrush.GradientStops>
                    <GradientStopCollection>
                        <GradientStop Color="{DynamicResource
BorderLightColor}" Offset="0.0" />
                        <GradientStop Color="{DynamicResource
BorderDarkColor}" Offset="1.0" />
                    </GradientStopCollection>
                </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
        </Border.BorderBrush>
    </Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Trigger>
</Style.Triggers>
</Style>

<Style x:Key="{x:Type ListView}"
    TargetType="ListView">
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="ScrollViewer.HorizontalScrollBarVisibility"
        Value="Auto" />
    <Setter Property="ScrollViewer.VerticalScrollBarVisibility"
        Value="Auto" />
    <Setter Property="ScrollViewer.CanContentScroll"
        Value="true" />
    <Setter Property="VerticalContentAlignment"
        Value="Center" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="ListView">
                <Border Name="Border"
                    BorderThickness="1">
                    <Border.Background>
                        <SolidColorBrush Color="{StaticResource ControlLightColor}" />
                    </Border.Background>
                    <Border.BorderBrush>
                        <SolidColorBrush Color="{StaticResource BorderMediumColor}" />
                    </Border.BorderBrush>
                    <ScrollViewer Style="{DynamicResource
{x:Static GridView.GridViewScrollViewerStyleKey}}">
                        <ItemsPresenter />
                    </ScrollViewer>
                </Border>
            <ControlTemplate.Triggers>
                <Trigger Property="IsGrouping"

```

```

        Value="true">
    <Setter Property="ScrollViewer.CanContentScroll"
        Value="false" />
</Trigger>
<Trigger Property="IsEnabled"
        Value="false">
    <Setter TargetName="Border"
        Property="Background">
        <Setter.Value>
            <SolidColorBrush Color="{DynamicResource
DisabledBorderLightColor}" />
        </Setter.Value>
    </Setter>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="{x:Type ListViewItem}"
    TargetType="ListViewItem">
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="ListBoxItem">
                <Border x:Name="Border"
                    Padding="2"
                    SnapsToDevicePixels="true"
                    Background="Transparent">
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="MouseOver" />
                            <VisualState x:Name="Disabled" />
                        </VisualStateGroup>
                        <VisualStateGroup x:Name="SelectionStates">
                            <VisualState x:Name="Unselected" />
                            <VisualState x:Name="Selected">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Border"
Storyboard.TargetProperty=""
(Panel.Background).
                                        (SolidColorBrush.Color)">
                                        <EasingColorKeyFrame KeyTime="0"
                                            Value="{StaticResource
SelectedBackgroundColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="SelectedUnfocused">

```

```

        <Storyboard>
            <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Border"
                Storyboard.TargetProperty="

(Panel.Background).
                    (SolidColorBrush.Color)">
                    <EasingColorKeyFrame KeyTime="0"
                        Value="{StaticResource
SelectedUnfocusedColor}" />
                </ColorAnimationUsingKeyFrames>
            </Storyboard>
        </VisualState>
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<GridViewRowPresenter VerticalAlignment="{TemplateBinding
VerticalContentAlignment}" />
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

ControlTemplate 示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>

```

```

<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)

- 控件样式和模板
- 控件自定义
- 样式设置和模板化
- 创建控件模板

Menu 样式和模板

项目 • 2023/02/06

本主题介绍 [Menu](#) 控件的样式和模板。可以修改默认的 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

Menu 部件

[Menu](#) 控件没有任何已命名的部件。

为 [Menu](#) 创建一个 [ControlTemplate](#) 时，你的模板可能在 [ScrollViewer](#) 中包含一个 [ItemsPresenter](#)。（[ItemsPresenter](#) 显示 [Menu](#) 中的每一项；[ScrollViewer](#) 允许在控件内滚动）。如果 [ItemsPresenter](#) 不是 [ScrollViewer](#) 的直接子元素，则必须为 [ItemsPresenter](#) 指定名称 `ItemsPresenter`。

Menu 状态

下表列出了 [Menu](#) 控件的可视状态。

VisualState 名称 名称	VisualStateGroup 名称	描述
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	如果控件具有焦点，则 Validation.HasError 附加属性为 <code>true</code> 。
InvalidUnfocused	ValidationStates	如果控件不具有焦点，则 Validation.HasError 附加属性为 <code>true</code> 。

MenuItem 部件

下表列出了 [Menu](#) 控件的命名部件。

组成部分	类型	描述
PART_Popup	Popup	子菜单区域。

为 [MenuItem](#) 创建一个 [ControlTemplate](#) 时，你的模板可能在 [ScrollViewer](#) 中包含一个 [ItemsPresenter](#)。（[ItemsPresenter](#) 显示 [MenuItem](#) 中的每一项；[ScrollViewer](#) 允许在控

件内滚动）。如果 ItemsPresenter 不是 ScrollViewer 的直接子元素，则必须为 ItemsPresenter 指定名称 ItemsPresenter。

MenuItem 状态

下表列出了 MenuItem 控件的可视状态。

VisualState 名称	VisualStateGroup 名称	描述
有效	ValidationStates	该控件使用 Validation 类，Validation.HasError 附加属性为 false。
InvalidFocused	ValidationStates	如果控件具有焦点，则 Validation.HasError 附加属性为 true。
InvalidUnfocused	ValidationStates	如果控件不具有焦点，则 Validation.HasError 附加属性为 true。

Menu 和 MenuItem ControlTemplate 示例

以下示例演示如何定义 Menu 控件的 ControlTemplate。

XAML

```
<Style x:Key="{x:Type Menu}">
    TargetType="{x:Type Menu}"
    <Setter Property="OverridesDefaultStyle"
        Value="True" />
    <Setter Property="SnapsToDevicePixels"
        Value="True" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Menu}">
                <Border BorderThickness="1">
                    <Border.BorderBrush>
                        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStop Collection>
                                    <GradientStop Color="{DynamicResource BorderLightColor}"
                                        Offset="0.0" />
                                    <GradientStop Color="{DynamicResource BorderDarkColor}"
                                        Offset="1.0" />
                                </GradientStop Collection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Border.BorderBrush>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

```
</Border.BorderBrush>
<Border.Background>
    <LinearGradientBrush EndPoint="0.5,1"
        StartPoint="0.5,0">
        <GradientStop Color="{DynamicResource ControlLightColor}"
            Offset="0" />
        <GradientStop Color="{DynamicResource ControlMediumColor}"
            Offset="1" />
    </LinearGradientBrush>
</Border.Background>
<StackPanel ClipToBounds="True"
    Orientation="Horizontal"
    IsItemsHost="True" />
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
```

以下示例演示如何定义 [MenuItem](#) 控件的 [ControlTemplate](#)。

XAML

```
<Style x:Key="{x:Static MenuItem.SeparatorStyleKey}"
    TargetType="{x:Type Separator}">
    <Setter Property="Height"
        Value="1" />
    <Setter Property="Margin"
        Value="0,4,0,4" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Separator}">
                <Border BorderThickness="1">
                    <Border.BorderBrush>
                        <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                    </Border.BorderBrush>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<!-- TopLevelHeader -->
<ControlTemplate x:Key="{x:Static MenuItem.TopLevelHeaderTemplateKey}"
    TargetType="{x:Type MenuItem}">
    <Border x:Name="Border">
        <Grid>
            <ContentPresenter Margin="6,3,6,3"
                ContentSource="Header"
                RecognizesAccessKey="True" />
            <Popup x:Name="Popup"
                Placement="Bottom"
                IsOpen="{TemplateBinding IsSubmenuOpen}">
```

```

        AllowsTransparency="True"
        Focusable="False"
        PopupAnimation="Fade">
    <Border x:Name="SubmenuBorder"
            SnapsToDevicePixels="True"
            BorderThickness="1"
            Background="{DynamicResource MenuPopupBrush}">
        <Border.BorderBrush>
            <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
        </Border.BorderBrush>
        <ScrollViewer CanContentScroll="True"
                    Style="{StaticResource MenuScrollViewer}">
            <StackPanel IsItemsHost="True"
                        KeyboardNavigation.Directionality="Cycle" />
        </ScrollViewer>
    </Border>
</Popups>
</Grid>
</Border>
<ControlTemplate.Triggers>
    <Trigger Property="IsSuspendedPopupAnimation"
              Value="true">
        <Setter TargetName="Popup"
                  Property="PopupAnimation"
                  Value="None" />
    </Trigger>
    <Trigger Property="IsHighlighted"
              Value="true">
        <Setter TargetName="Border"
                  Property="BorderBrush"
                  Value="Transparent" />
        <Setter Property="Background"
                  TargetName="Border">
            <Setter.Value>
                <LinearGradientBrush StartPoint="0,0"
                                      EndPoint="0,1">
                    <LinearGradientBrush.GradientStops>
                        <GradientStopCollection>
                            <GradientStop Color="{StaticResource ControlLightColor}" />
                            <GradientStop Color="{StaticResource ControlMouseOverColor}"
                                          Offset="1.0" />
                        </GradientStopCollection>
                    </LinearGradientBrush.GradientStops>
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
    </Trigger>
    <Trigger SourceName="Popup"
              Property="AllowsTransparency"
              Value="True">
        <Setter TargetName="SubmenuBorder"
                  Property="CornerRadius"
                  Value="0,0,4,4" />
        <Setter TargetName="SubmenuBorder"

```

```
        Property="Padding"
        Value="0,0,0,3" />
    </Trigger>
    <Trigger Property="IsEnabled"
        Value="False">
        <Setter Property="Foreground">
            <Setter.Value>
                <SolidColorBrush Color="{StaticResource DisabledForegroundColor}" />
            </Setter.Value>
        </Setter>
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>

<!-- TopLevelItem -->
<ControlTemplate x:Key="{x:Static MenuItem.TopLevelItemTemplateKey}"
    TargetType="{x:Type MenuItem}">
    <Border x:Name="Border">
        <Grid>
            <ContentPresenter Margin="6,3,6,3"
                ContentSource="Header"
                RecognizesAccessKey="True" />
        </Grid>
    </Border>
    <ControlTemplate.Triggers>
        <Trigger Property="IsHighlighted"
            Value="true">
            <Setter Property="Background"
                TargetName="Border">
                <Setter.Value>
                    <LinearGradientBrush StartPoint="0,0"
                        EndPoint="0,1">
                        <LinearGradientBrush.GradientStops>
                            <GradientStopCollection>
                                <GradientStop Color="{StaticResource ControlLightColor}" />
                                <GradientStop Color="{StaticResource ControlMouseOverColor}"
                                    Offset="1.0" />
                            </GradientStopCollection>
                        </LinearGradientBrush.GradientStops>
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
        </Trigger>
        <Trigger Property="IsEnabled"
            Value="False">
            <Setter Property="Foreground">
                <Setter.Value>
                    <SolidColorBrush Color="{StaticResource DisabledForegroundColor}" />
                </Setter.Value>
            </Setter>
        </Trigger>
    </ControlTemplate.Triggers>
```

```

        </ControlTemplate>

    <!-- SubmenuItem -->
    <ControlTemplate x:Key="{x:Static MenuItem.SubmenuItemTemplateKey}"
                      TargetType="{x:Type MenuItem}">
        <Border x:Name="Border"
                BorderThickness="1">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto"
                                      SharedSizeGroup="Icon" />
                    <ColumnDefinition Width="*" />
                    <ColumnDefinition Width="Auto"
                                      SharedSizeGroup="Shortcut" />
                    <ColumnDefinition Width="13" />
                </Grid.ColumnDefinitions>
                <ContentPresenter x:Name="Icon"
                                  Margin="6,0,6,0"
                                  VerticalAlignment="Center"
                                  ContentSource="Icon" />
                <Border x:Name="Check"
                        Width="13"
                        Height="13"
                        Visibility="Collapsed"
                        Margin="6,0,6,0"
                        BorderThickness="1">
                    <Border.BorderBrush>
                        <LinearGradientBrush StartPoint="0,0"
                                            EndPoint="0,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStopCollection>
                                    <GradientStop Color="{DynamicResource BorderLightColor}"
                                                  Offset="0.0" />
                                    <GradientStop Color="{DynamicResource BorderDarkColor}"
                                                  Offset="1.0" />
                                </GradientStopCollection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Border.BorderBrush>
                    <Border.Background>
                        <LinearGradientBrush StartPoint="0,0"
                                            EndPoint="0,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStopCollection>
                                    <GradientStop Color="{DynamicResource ControlLightColor}" />
                                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                                                  Offset="1.0" />
                                </GradientStopCollection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Border.Background>
                    <Path x:Name="CheckMark"
                          Width="7"
                          Height="7"

```

```
        Visibility="Hidden"
        SnapsToDevicePixels="False"
        StrokeThickness="2"
        Data="M 0 0 L 7 7 M 0 7 L 7 0">
    <Path.Stroke>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Stroke>
</Path>
</Border>
<ContentPresenter x:Name="HeaderHost"
    Grid.Column="1"
    ContentSource="Header"
    RecognizesAccessKey="True" />
<TextBlock x:Name="InputGestureText"
    Grid.Column="2"
    Text="{TemplateBinding InputGestureText}"
    Margin="5,2,0,2"
    DockPanel.Dock="Right" />
</Grid>
</Border>
<ControlTemplate.Triggers>
    <Trigger Property="ButtonBase.Command"
        Value="{x:Null}" />
    <Trigger Property="Icon"
        Value="{x:Null}">
        <Setter TargetName="Icon"
            Property="Visibility"
            Value="Hidden" />
    </Trigger>
    <Trigger Property="IsChecked"
        Value="true">
        <Setter TargetName="CheckMark"
            Property="Visibility"
            Value="Visible" />
    </Trigger>
    <Trigger Property="IsCheckable"
        Value="true">
        <Setter TargetName="Check"
            Property="Visibility"
            Value="Visible" />
        <Setter TargetName="Icon"
            Property="Visibility"
            Value="Hidden" />
    </Trigger>
    <Trigger Property="IsHighlighted"
        Value="true">
        <Setter Property="Background"
            TargetName="Border">
            <Setter.Value>
                <LinearGradientBrush EndPoint="0.5,1"
                    StartPoint="0.5,0">
                    <GradientStop Color="Transparent"
                        Offset="0" />
                    <GradientStop Color="{DynamicResource ControlMouseOverColor}"
                        Offset="1" />
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
    </Trigger>

```

```

        </LinearGradientBrush>
    </Setter.Value>
</Setter>
<Setter Property="BorderBrush"
        TargetName="Border">
    <Setter.Value>
        <LinearGradientBrush EndPoint="0.5,1"
                            StartPoint="0.5,0">
            <GradientStop Color="{DynamicResource BorderMediumColor}"
                           Offset="0" />
            <GradientStop Color="Transparent"
                           Offset="1" />
        </LinearGradientBrush>
    </Setter.Value>
</Setter>
</Trigger>
<Trigger Property="IsEnabled"
         Value="false">
    <Setter Property="Foreground">
        <Setter.Value>
            <SolidColorBrush Color="{StaticResource DisabledForegroundColor}" />
        </Setter.Value>
    </Setter>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>

<ControlTemplate x:Key="{x:Static MenuItem.SubmenuHeaderTemplateKey}"
                 TargetType="{x:Type MenuItem}">
    <Border x:Name="Border"
           BorderThickness="1">
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto"
                                  SharedSizeGroup="Icon" />
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="Auto"
                                  SharedSizeGroup="Shortcut" />
                <ColumnDefinition Width="13" />
            </Grid.ColumnDefinitions>
            <ContentPresenter x:Name="Icon"
                              Margin="6,0,6,0"
                              VerticalAlignment="Center"
                              ContentSource="Icon" />
            <ContentPresenter x:Name="HeaderHost"
                              Grid.Column="1"
                              ContentSource="Header"
                              RecognizesAccessKey="True" />
            <TextBlock x:Name="InputGestureText"
                      Grid.Column="2"
                      Text="{TemplateBinding InputGestureText}"
                      Margin="5,2,2,2"
                      DockPanel.Dock="Right" />
            <Path Grid.Column="3"

```

```

        HorizontalAlignment="Center"
        VerticalAlignment="Center"
        Data="M 0 0 L 0 7 L 4 3.5 Z">
    <Path.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Fill>
</Path>
<Popup x:Name="Popup"
    Placement="Right"
    HorizontalOffset="-4"
    IsOpen="{TemplateBinding IsSubmenuOpen}"
    AllowsTransparency="True"
    Focusable="False"
    PopupAnimation="Fade">
    <Border x:Name="SubmenuBorder"
        SnapsToDevicePixels="True"
        Background="{DynamicResource MenuPopupBrush}"
        BorderThickness="1">
        <Border.BorderBrush>
            <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
        </Border.BorderBrush>
        <ScrollViewer CanContentScroll="True"
            Style="{StaticResource MenuScrollViewer}">
            <StackPanel IsItemsHost="True"
                KeyboardNavigation.DirectionalNavigation="Cycle" />
        </ScrollViewer>
    </Border>
</Popup>
</Grid>
</Border>
<ControlTemplate.Triggers>
    <Trigger Property="Icon"
        Value="{x:Null}">
        <Setter TargetName="Icon"
            Property="Visibility"
            Value="Collapsed" />
    </Trigger>
    <Trigger Property="IsHighlighted"
        Value="true">
        <Setter Property="Background"
            TargetName="Border">
            <Setter.Value>
                <LinearGradientBrush EndPoint="0.5,1"
                    StartPoint="0.5,0">
                    <GradientStop Color="Transparent"
                        Offset="0" />
                    <GradientStop Color="{DynamicResource ControlMouseOverColor}"
                        Offset="1" />
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
        <Setter Property="BorderBrush"
            TargetName="Border">
            <Setter.Value>
                <LinearGradientBrush EndPoint="0.5,1"

```

```

                StartPoint="0.5,0">
            <GradientStop Color="{DynamicResource BorderMediumColor}"
                           Offset="0" />
            <GradientStop Color="Transparent"
                           Offset="1" />
        </LinearGradientBrush>
    </Setter.Value>
</Setter>
</Trigger>
<Trigger SourceName="Popup"
         Property="AllowsTransparency"
         Value="True">
    <Setter TargetName="SubmenuBorder"
            Property="CornerRadius"
            Value="4" />
    <Setter TargetName="SubmenuBorder"
            Property="Padding"
            Value="0,3,0,3" />
</Trigger>
<Trigger Property="IsEnabled"
         Value="false">
    <Setter Property="Foreground">
        <Setter.Value>
            <SolidColorBrush Color="{StaticResource DisabledForegroundColor}" />
        </Setter.Value>
    </Setter>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>


<Style x:Key="{x:Type MenuItem}"
       TargetType="{x:Type MenuItem}">
    <Setter Property="OverridesDefaultStyle"
           Value="True" />
    <Style.Triggers>
        <Trigger Property="Role"
                 Value="TopLevelHeader">
            <Setter Property="Template"
                   Value="{StaticResource {x:Static MenuItem.TopLevelHeaderTemplateKey}}" />
            <Setter Property="Grid.IsSharedSizeScope"
                   Value="true" />
        </Trigger>
        <Trigger Property="Role"
                 Value="TopLevelItem">
            <Setter Property="Template"
                   Value="{StaticResource {x:Static MenuItem.TopLevelItemTemplateKey}}" />
        </Trigger>
        <Trigger Property="Role"
                 Value="SubmenuHeader">
            <Setter Property="Template"
                   Value="{StaticResource {x:Static

```

```
MenuItem.SubmenuHeaderTemplateKey}}}" />
    </Trigger>
    <Trigger Property="Role"
        Value="SubmenuItem">
        <Setter Property="Template"
            Value="{StaticResource {x:Static
MenuItem.SubmenuItemTemplateKey}}}" />
    </Trigger>
</Style.Triggers>
</Style>
```

以下示例定义了前面示例中使用的 `MenuScrollViewer`。

XAML

```
<!--ScrollView for a MenuItem-->
<MenuScrollingVisibilityConverter x:Key="MenuScrollingVisibilityConverter"
/>

<Style x:Key="MenuScrollViewer"
    TargetType="{x:Type ScrollViewer}"
    BasedOn="{x:Null}">
    <Setter Property="HorizontalScrollBarVisibility"
        Value="Hidden" />
    <Setter Property="VerticalScrollBarVisibility"
        Value="Auto" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type ScrollViewer}">
                <Grid SnapsToDevicePixels="True">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*" />
                    </Grid.ColumnDefinitions>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto" />
                        <RowDefinition Height="*" />
                        <RowDefinition Height="Auto" />
                    </Grid.RowDefinitions>
                    <Border Grid.Row="1"
                        Grid.Column="0">
                        <ScrollContentPresenter Margin="{TemplateBinding Padding}" />
                    </Border>
                    <RepeatButton Style="{StaticResource MenuScrollButton}"
                        Grid.Row="0"
                        Grid.Column="0"
                        Command="{x:Static ScrollBar.LineUpCommand}"
                        CommandTarget="{Binding RelativeSource=
{RelativeSource TemplatedParent}}"
                        Focusable="False">
                        <RepeatButton.Visibility>
                            <MultiBinding FallbackValue="Visibility.Collapsed"
                                Converter="{StaticResource
MenuScrollingVisibilityConverter}">
```

```

                ConverterParameter="0">
            <Binding RelativeSource="{RelativeSource TemplatedParent}"
                      Path="ComputedVerticalScrollBarVisibility" />
            <Binding RelativeSource="{RelativeSource TemplatedParent}"
                      Path="VerticalOffset" />
            <Binding RelativeSource="{RelativeSource TemplatedParent}"
                      Path="ExtentHeight" />
            <Binding RelativeSource="{RelativeSource TemplatedParent}"
                      Path="ViewportHeight" />
        </MultiBinding>
    </RepeatButton.Visibility>
    <Path Fill="{DynamicResource {x:Static
SystemColors.MenuTextBrushKey}}"
          Data="{StaticResource UpArrow}" />
</RepeatButton>
<RepeatButton Style="{StaticResource MenuScrollBarButton}"
              Grid.Row="2"
              Grid.Column="0"
              Command="{x:Static ScrollBar.LineDownCommand}"
              CommandTarget="{Binding RelativeSource=
{RelativeSource TemplatedParent}}"
              Focusable="False">
    <RepeatButton.Visibility>
        <MultiBinding FallbackValue="Visibility.Collapsed"
                      Converter="{StaticResource
MenuScrollingVisibilityConverter}"
                      ConverterParameter="100">
            <Binding RelativeSource="{RelativeSource TemplatedParent}"
                      Path="ComputedVerticalScrollBarVisibility" />
            <Binding RelativeSource="{RelativeSource TemplatedParent}"
                      Path="VerticalOffset" />
            <Binding RelativeSource="{RelativeSource TemplatedParent}"
                      Path="ExtentHeight" />
            <Binding RelativeSource="{RelativeSource TemplatedParent}"
                      Path="ViewportHeight" />
        </MultiBinding>
    </RepeatButton.Visibility>
    <Path Fill="{DynamicResource {x:Static
SystemColors.MenuTextBrushKey}}"
          Data="{StaticResource DownArrow}" />
</RepeatButton>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

ControlTemplate 示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>

```

```

<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
                     EndPoint="0.5,1"
                     StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
                  Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
                  Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
                  Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
                     StartPoint="0,0">

```

```
        EndPoint="1,0">
<LinearGradientBrush.GradientStops>
    <GradientStopCollection>
        <GradientStop Color="#000000FF"
                      Offset="0" />
        <GradientStop Color="#600000FF"
                      Offset="0.4" />
        <GradientStop Color="#600000FF"
                      Offset="0.6" />
        <GradientStop Color="#000000FF"
                      Offset="1" />
    </GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

NavigationWindow 样式和模板

项目 • 2023/02/06

本主题介绍 [NavigationWindow](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

NavigationWindow 部件

下表列出了 [NavigationWindow](#) 控件的命名部件。

组成部分	类型	描述
PART_NavWinCP	ContentPresenter	内容的区域。

NavigationWindow 状态

下表列出了 [NavigationWindow](#) 控件的视觉对象状态。

VisualState 名称	VisualStateGroup 名称	描述
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

NavigationWindow ControlTemplate 示例

尽管此示例包含默认情况下在 [NavigationWindow](#) 的 [ControlTemplate](#) 中定义的所有元素，但应将特定值视为示例。

XAML

```
<Style x:Key="NavWinButtonStyle"
       TargetType="{x:Type Button}">
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="Command"
           Value="NavigationCommands.BrowseBack" />
```

```

<Setter Property="Focusable"
       Value="false" />
<Setter Property="Template">
  <Setter.Value>
    <ControlTemplate TargetType="{x:Type Button}">
      <Grid>
        <VisualStateManager.VisualStateGroups>
          <VisualStateGroup x:Name="CommonStates">
            <VisualState x:Name="Normal" />
            <VisualState x:Name="MouseOver">
              <Storyboard>
                <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Ellipse"
Storyboard.TargetProperty=""
(Shape.Fill).
                (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                  <EasingColorKeyFrame KeyTime="0"
Value="{StaticResource
ControlMouseOverColor}" />
                </ColorAnimationUsingKeyFrames>
              </Storyboard>
            </VisualState>
            <VisualState x:Name="Pressed">
              <Storyboard>
                <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Ellipse"
Storyboard.TargetProperty=""
(Shape.Fill).
                (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                  <EasingColorKeyFrame KeyTime="0"
Value="{StaticResource
ControlPressedColor}" />
                </ColorAnimationUsingKeyFrames>
              </Storyboard>
            </VisualState>
            <VisualState x:Name="Disabled">
              <Storyboard>
                <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Ellipse"
Storyboard.TargetProperty=""
(Shape.Fill).
                (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                  <EasingColorKeyFrame KeyTime="0"
Value="{StaticResource
DisabledControlDarkColor}" />
                </ColorAnimationUsingKeyFrames>
              </Storyboard>
            </VisualState>
          </VisualStateGroup>
        </VisualStateManager>
        <Path Data="M 0 0 L 100 100" Fill="White" Stroke="Black" StrokeThickness="1"/>
        <Path Data="M 0 0 L 100 100" Fill="Black" Stroke="White" StrokeThickness="1"/>
      </Grid>
    </ControlTemplate>
  </Setter.Value>
</Setter>

```

```

        </Storyboard>

    </VisualState>
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Ellipse x:Name="Ellipse"
    StrokeThickness="1"
    Width="24"
    Height="24">
    <Ellipse.Stroke>
        <SolidColorBrush Color="{DynamicResource NavButtonFrameColor}" />
    </Ellipse.Stroke>
    <Ellipse.Fill>
        <LinearGradientBrush StartPoint="0,0"
            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource ControlLightColor}" />
                    <GradientStop Color="{DynamicResource ControlMediumColor}" />
                    <GradientStop Color="Black" Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Ellipse.Fill>
</Ellipse>
<Path x:Name="Arrow"
    Margin="0,0,3,0"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"
    Data="M 6 0 L 0 6 L 6 12 Z">
    <Path.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Fill>
</Path>
</Grid>
<ControlTemplate.Triggers>
    <Trigger Property="Command"
        Value="{x:Static NavigationCommands.BrowseForward}">
        <Setter TargetName="Arrow"
            Property="Data"
            Value="M 0 0 L 6 6 L 0 12 z" />
        <Setter TargetName="Arrow"
            Property="Margin"
            Value="3,0,0,0" />
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

```

<!-- NavWin Menu Style -->
<Style x:Key="NavWinMenu"
    TargetType="{x:Type Menu}">
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="KeyboardNavigation.TabNavigation"
        Value="None" />
    <Setter Property="IsMainMenu"
        Value="false" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Menu}">
                <DockPanel IsItemsHost="true" />
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<!-- NavWin Menu Header Style -->
<Style x:Key="NavWinHeaderMenuItem"
    TargetType="{x:Type MenuItem}">
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type MenuItem}">
                <Grid>
                    <Popup x:Name="PART_Popup"
                        Placement="Bottom"
                        VerticalOffset="2"
                        IsOpen="{TemplateBinding IsSubmenuOpen}"
                        AllowsTransparency="True"
                        Focusable="False"
                        PopupAnimation="Fade">
                        <Border x:Name="SubMenuBorder"
                            Background="{DynamicResource MenuPopupBrush}"
                            BorderThickness="1">
                            <Border.BorderBrush>
                                <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                            </Border.BorderBrush>
                            <StackPanel IsItemsHost="true"
                                Margin="2"
                                KeyboardNavigation.TabNavigation="Cycle"
                                KeyboardNavigation.DirectionalNavigation="Cycle" />
                        </Border>
                    </Popup>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```
<Border Visibility="Hidden"
        x:Name="HighlightBorder"
        BorderThickness="1"
        CornerRadius="2">
    <Border.BorderBrush>
        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource
BorderLightColor}" Offset="0.0" />
                    <GradientStop Color="{DynamicResource
BorderDarkColor}" Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Border.BorderBrush>
    <Border.Background>
        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource
ControlLightColor}" />
                    <GradientStop Color="{DynamicResource
ControlMouseOverColor}" Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Border.Background>
</Border>
<Path x:Name="Arrow"
      SnapsToDevicePixels="false"
      HorizontalAlignment="Right"
      VerticalAlignment="Center"
      Margin="0,2,4,0"
      StrokeLineJoin="Round"
      Data="M 0 0 L 4 4 L 8 0 Z">
    <Path.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Fill>
</Path>
</Grid>
</Grid>
<ControlTemplate.Triggers>
    <Trigger Property="IsHighlighted"
            Value="true">
        <Setter TargetName="HighlightBorder"
               Property="Visibility"
```

```

                Value="Visible" />
            </Trigger>
            <Trigger Property="Is_submenuOpen"
                Value="true">
                <Setter TargetName="HighlightBorder"
                    Property="BorderBrush">
                    <Setter.Value>
                        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
                            <GradientBrush.GradientStops>
                                <GradientStopCollection>
                                    <GradientStop Color="{DynamicResource
BorderDarkColor}"
                                        Offset="0.0" />
                                    <GradientStop Color="{DynamicResource
BorderMediumColor}"
                                        Offset="1.0" />
                                </GradientStopCollection>
                            </GradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Setter.Value>
                </Setter>
                <Setter Property="Background"
                    TargetName="HighlightBorder">
                    <Setter.Value>
                        <LinearGradientBrush EndPoint="0,1"
                            StartPoint="0,0">
                            <GradientStop Color="{DynamicResource ControlLightColor}"
                                Offset="0" />
                            <GradientStop Color="{DynamicResource
ControlPressedColor}"
                                Offset="0.984" />
                        </LinearGradientBrush>
                    </Setter.Value>
                </Setter>
            </Trigger>
        </ControlTemplate.Triggers>
    </ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!-- NavWin Menu Item Style --&gt;

&lt;Style x:Key="NavWinSubmenuItem"
    TargetType="{x:Type MenuItem}"&gt;
    &lt;Setter Property="OverridesDefaultCellStyle"
        Value="true" /&gt;
    &lt;Setter Property="Header"
        Value="{Binding (JournalEntry.Name)}" /&gt;
    &lt;Setter Property="Command"
        Value="NavigationCommands.NavigateJournal" /&gt;
    &lt;Setter Property="CommandTarget"
</pre>

```

```

        Value="{Binding TemplatedParent, RelativeSource={RelativeSource
AncestorType={x:Type Menu}}}" />
    <Setter Property="CommandParameter"
        Value="{Binding RelativeSource={RelativeSource Self}}"/>
    <Setter Property="JournalEntryUnifiedViewConverter.JournalEntryPosition"
        Value="{Binding
(JournalEntryUnifiedViewConverter.JournalEntryPosition)}"/>
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type MenuItem}">
                <Border Name="Border"
                    BorderThickness="1">
                    <Grid x:Name="Panel"
                        Background="Transparent"
                        SnapsToDevicePixels="true"
                        Height="35"
                        Width="250">
                        <Path x:Name="Glyph"
                            SnapsToDevicePixels="false"
                            Margin="7,5"
                            Width="10"
                            Height="10"
                            HorizontalAlignment="Left"
                            StrokeStartLineCap="Triangle"
                            StrokeEndLineCap="Triangle"
                            StrokeThickness="2">
                            <Path.Stroke>
                                <SolidColorBrush Color="{DynamicResource GlyphColor}" />
                            </Path.Stroke>
                        </Path>
                        <ContentPresenter ContentSource="Header"
                            Margin="24,5,50,5" />
                    </Grid>
                </Border>
                <ControlTemplate.Triggers>
                    <Trigger Value="Current"
Property="JournalEntryUnifiedViewConverter.JournalEntryPosition">
                        <Setter TargetName="Glyph"
                            Property="Data"
                            Value="M 0,5 L 2.5,8 L 7,3 " />
                    </Trigger>
                    <Trigger Property="IsHighlighted"
Value="true">
                        <Setter Property="Background"
                            TargetName="Border">
                            <Setter.Value>
                                <LinearGradientBrush EndPoint="0.5,1"
                                    StartPoint="0.5,0">
                                    <GradientStop Color="Transparent"
                                        Offset="0" />
                                    <GradientStop Color="{DynamicResource
ControlMouseOverColor}"
                                        Offset="1" />
                                </LinearGradientBrush>
                            </Setter.Value>
                        </Setter>
                    </Trigger>
                </ControlTemplate.Triggers>
            </ControlTemplate>
        </Setter.Value>
    </Setter>

```

```
        </Setter.Value>
    </Setter>
    <Setter Property="BorderBrush"
        TargetName="Border">
        <Setter.Value>
            <LinearGradientBrush EndPoint="0.5,1"
                StartPoint="0.5,0">
                <GradientStop Color="{DynamicResource BorderMediumColor}"
                    Offset="0" />
                <GradientStop Color="Transparent"
                    Offset="1" />
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
</Trigger>
<MultiTrigger>
    <MultiTrigger.Conditions>
        <Condition Property="IsHighlighted"
            Value="true" />
        <Condition Value="Forward"
            Property="JournalEntryUnifiedViewConverter.JournalEntryPosition" />
    </MultiTrigger.Conditions>
    <Setter TargetName="Glyph"
        Property="Data"
        Value="M 3 1 L 7 5 L 3 9 z" />
    <Setter TargetName="Glyph"
        Property="Fill">
        <Setter.Value>
            <SolidColorBrush Color="{StaticResource GlyphColor}" />
        </Setter.Value>
    </Setter>
    <Setter TargetName="Glyph"
        Property="Stroke"
        Value="{x:Null}" />
</MultiTrigger>
<MultiTrigger>
    <MultiTrigger.Conditions>
        <Condition Property="IsHighlighted"
            Value="true" />
        <Condition Value="Back"
            Property="JournalEntryUnifiedViewConverter.JournalEntryPosition" />
    </MultiTrigger.Conditions>
    <Setter TargetName="Glyph"
        Property="Data"
        Value="M 7 1 L 3 5 L 7 9 z" />
    <Setter TargetName="Glyph"
        Property="Fill">
        <Setter.Value>
            <SolidColorBrush Color="{StaticResource GlyphColor}" />
        </Setter.Value>
    </Setter>
    <Setter TargetName="Glyph"
        Property="Stroke"
```

```

                Value="{x:Null}" />
            </MultiTrigger>
        </ControlTemplate.Triggers>
    </ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!-- Merges Back and Forward Navigation Stacks -->

<JournalEntryUnifiedViewConverter x:Key="JournalEntryUnifiedViewConverter"
/>

<!-- SimpleStyles: NavigationWindow -->

<Style x:Key="{x:Type NavigationWindow}"
       TargetType="{x:Type NavigationWindow}">
    <Setter Property="SnapsToDevicePixels"
           Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type NavigationWindow}">
                <DockPanel>
                    <DockPanel.Background>
                        <SolidColorBrush Color="{DynamicResource WindowColor}" />
                    </DockPanel.Background>
                    <Border DockPanel.Dock="Top"
                           Height="30"
                           BorderThickness="1">
                        <Border.BorderBrush>
                            <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                    </Border.BorderBrush>
                    <Grid>
                        <Grid.Background>

                            <LinearGradientBrush StartPoint="0,0"
                                               EndPoint="0,1">
                                <LinearGradientBrush.GradientStops>
                                    <GradientStopCollection>
                                        <GradientStop Color="{DynamicResource
ControlLightColor}"
                                              Offset="0.0" />
                                        <GradientStop Color="{DynamicResource
ControlMediumColor}"
                                              Offset="1.0" />
                                    </GradientStopCollection>
                                </LinearGradientBrush.GradientStops>
                            </LinearGradientBrush>

                        </Grid.Background>
                        <Grid.ColumnDefinitions>
                            <ColumnDefinition Width="Auto" />
                            <ColumnDefinition Width="Auto" />
                            <ColumnDefinition Width="16" />
                        </Grid.ColumnDefinitions>
                    </Grid>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>

```

```

        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>

    <Menu x:Name="NavMenu"
        Grid.ColumnSpan="3"
        Height="20"
        Margin="1,0,0,0"
        VerticalAlignment="Center"
        Style="{StaticResource NavWinMenu}">
        <MenuItem Style="{StaticResource NavWinHeaderMenuItem}"
            ItemContainerStyle="{StaticResource
NavWinSubmenuItem}"
            IsSubmenuOpen="{Binding (MenuItem.IsSubmenuOpen),
Mode=TwoWay, RelativeSource={RelativeSource
TemplatedParent}}">
            <MenuItem.ItemsSource>
                <MultiBinding Converter="{StaticResource
JournalEntryUnifiedViewConverter}">
                    <Binding RelativeSource="{RelativeSource
TemplatedParent}">
                        Path="BackStack" />
                    <Binding RelativeSource="{RelativeSource
TemplatedParent}">
                        Path="ForwardStack" />
                </MultiBinding>
            </MenuItem.ItemsSource>
        </MenuItem>
    </Menu>

    <Path Grid.Column="0"
        SnapsToDevicePixels="false"
        IsHitTestVisible="false"
        Margin="2,1.5,0,1.5"
        Grid.ColumnSpan="3"
        StrokeThickness="1"
        HorizontalAlignment="Left"
        VerticalAlignment="Center"
        Data="M22.5767,21.035 Q27,19.37
            31.424,21.035 A12.5,12.5,0,0,0,53.5,13
            A12.5,12.5,0,0,0,37.765,0.926
            Q27,4.93 16.235,0.926
            A12.5,12.5,0,0,0,0.5,13
            A12.5,12.5,0,0,0,22.5767,21.035 z">
        <Path.Stroke>
            <SolidColorBrush Color="{DynamicResource
BorderMediumColor}" />
        </Path.Stroke>
        <Path.Fill>
            <LinearGradientBrush EndPoint="0,1"
                StartPoint="0,0">
                <GradientStop Color="{DynamicResource
ControlMediumColor}">
                    Offset="0" />

```

```

        <GradientStop Color="{DynamicResource ControlDarkColor}"
                      Offset="0.984" />
    </LinearGradientBrush>
</Path.Fill>
</Path>
<Button Style="{StaticResource NavWinButtonStyle}"
        Command="NavigationCommands.BrowseBack"
        Content="M 4 0 L 0 4 L 4 8 Z"
        Margin="3,1.5,2,1.5"
        Grid.Column="0" />
<Button Style="{StaticResource NavWinButtonStyle}"
        Command="NavigationCommands.BrowseForward"
        Content="M 4 0 L 0 4 L 4 8 Z"
        Margin="2,1.5,0,1.5"
        Grid.Column="1" />
</Grid>
</Border>
<Grid>
    <AdornerDecorator>
        <Border BorderThickness="1">
            <Border.BorderBrush>
                <SolidColorBrush Color="{DynamicResource
BorderMediumColor}" />
            </Border.BorderBrush>
            <ContentPresenter x:Name="PART_NavWinCP"
                             ClipToBounds="true" />
        </Border>
    </AdornerDecorator>

    <ResizeGrip x:Name="WindowResizeGrip"
                HorizontalAlignment="Right"
                VerticalAlignment="Bottom"
                Visibility="Collapsed"
                IsTabStop="false" />
</Grid>
</DockPanel>
<ControlTemplate.Triggers>
    <Trigger Property="ResizeMode"
             Value="CanResizeWithGrip">
        <Setter TargetName="WindowResizeGrip"
               Property="Visibility"
               Value="Visible" />
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```

<Style x:Key="{x:Type ResizeGrip}"
    TargetType="{x:Type ResizeGrip}">
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type ResizeGrip}">
                <Border Background="Transparent"
                    SnapsToDevicePixels="True"
                    Width="16"
                    Height="16">
                    <Rectangle Margin="2">
                        <Rectangle.Fill>
                            <DrawingBrush Viewport="0,0,4,4"
                                ViewportUnits="Absolute"
                                Viewbox="0,0,8,8"
                                ViewboxUnits="Absolute"
                                TileMode="Tile">
                                <DrawingBrush.Drawing>
                                    <DrawingGroup>
                                        <DrawingGroup.Children>
                                            <GeometryDrawing Brush="#FFE8EDF9"
                                                Geometry="M 4 4 L 4 8 L
                                                8 8 L 8 4 z" />
                                        </DrawingGroup.Children>
                                    </DrawingGroup>
                                </DrawingBrush.Drawing>
                            </DrawingBrush>
                        </Rectangle.Fill>
                    </Rectangle>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

```

```

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
                     EndPoint="0.5,1"
                     StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
                  Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
                  Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
                  Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
                     StartPoint="0,0"
                     EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                          Offset="0" />
            <GradientStop Color="#600000FF"
                          Offset="0.4" />
            <GradientStop Color="#600000FF"
                          Offset="0.6" />
            <GradientStop Color="#000000FF"
                          Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

```
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

PasswordBox 样式和模板

项目 • 2022/09/27

本主题介绍 [PasswordBox](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

PasswordBox 部件

下表列出了 [PasswordBox](#) 控件的命名部件。

组成部分	类型	描述
PART_ContentHost	FrameworkElement	可包含 FrameworkElement 的可视元素。 PasswordBox 的文本显示在此元素中。

PasswordBox 状态

下表列出了 [PasswordBox](#) 控件的视觉对象状态。

VisualState 名称 名称	VisualStateGroup	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在控件上方。
已禁用	CommonStates	已禁用控件。
已设定焦点	FocusStates	控件有焦点。
失去焦点	FocusStates	控件没有焦点。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

PasswordBox ControlTemplate 示例

以下示例演示如何定义 [PasswordBox](#) 控件的 [ControlTemplate](#)。

XAML

```
<Style x:Key="{x:Type PasswordBox}">
    TargetType="{x:Type PasswordBox}"
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="KeyboardNavigation.TabNavigation"
        Value="None" />
    <Setter Property="FocusVisualStyle"
        Value="{x:Null}" />
    <Setter Property="FontFamily"
        Value="Verdana" />
    <Setter Property="PasswordChar"
        Value="*" />
    <Setter Property="MinWidth"
        Value="120" />
    <Setter Property="MinHeight"
        Value="20" />
    <Setter Property="AllowDrop"
        Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type PasswordBox}">
                <Border x:Name="Border"
                    CornerRadius="2"
                    Padding="2"
                    BorderThickness="1">
                    <Border.Background>
                        <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
                    </Border.Background>
                    <Border.BorderBrush>
                        <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                    </Border.BorderBrush>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="Disabled" />
                            <VisualState x:Name="MouseOver" />
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                    <ScrollViewer x:Name="PART_ContentHost" />
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

```

```
<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

ProgressBar 样式和模板

项目 • 2023/02/06

本主题介绍 [ProgressBar](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

ProgressBar 部件

下表列出了 [ProgressBar](#) 控件的已命名部件。

组成部分	类型	描述
PART_Indicator	FrameworkElement	指示进度的对象。
PART_Track	FrameworkElement	定义进度指示器路径的对象。
PART_GlowRect	FrameworkElement	修饰进度栏的对象。

ProgressBar 状态

下表列出了 [ProgressBar](#) 控件的视觉对象状态。

VisualState 名称	VisualStateGroup 名称	描述
确定	CommonStates	ProgressBar 基于 Value 属性报告进度。
不确定	CommonStates	ProgressBar 使用重复模式报告一般进度。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

ProgressBar ControlTemplate 示例

下例演示如何定义 [ProgressBar](#) 控件的 [ControlTemplate](#)。

XAML

```

<Style x:Key="{x:Type ProgressBar}"
       TargetType="{x:Type ProgressBar}">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="{x:Type ProgressBar}">
        <Grid MinHeight="14"
              MinWidth="200"
              Background="{TemplateBinding Background}">
          <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CommonStates">
              <VisualState x:Name="Determinate" />
              <VisualState x:Name="Indeterminate">
                <Storyboard>
                  <ObjectAnimationUsingKeyFrames Duration="00:00:00"
                      Storyboard.TargetName="PART_Indicator">
                    <Storyboard.TargetProperty="Background">
                      <DiscreteObjectKeyFrame KeyTime="00:00:00">
                        <DiscreteObjectKeyFrame.Value>
                          <SolidColorBrush>Transparent</SolidColorBrush>
                        </DiscreteObjectKeyFrame.Value>
                      </DiscreteObjectKeyFrame>
                    </ObjectAnimationUsingKeyFrames>
                  </Storyboard>
                </VisualState>
              </VisualStateGroup>
            </VisualStateManager.VisualStateGroups>
            <Border x:Name="PART_Track"
                    CornerRadius="2"
                    BorderThickness="1">
              <Border.BorderBrush>
                <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
              </Border.BorderBrush>
            </Border>
            <Border x:Name="PART_Indicator"
                    CornerRadius="2"
                    BorderThickness="1"
                    HorizontalAlignment="Left"
                    Background="{TemplateBinding Foreground}"
                    Margin="0,-1,0,1">
              <Border.BorderBrush>
                <LinearGradientBrush StartPoint="0,0"
                                    EndPoint="0,1">
                  <GradientBrush.GradientStops>
                    <GradientStopCollection>
                      <GradientStop Color="{DynamicResource BorderLightColor}"
                                    Offset="0.0" />
                      <GradientStop Color="{DynamicResource BorderMediumColor}"
                                    Offset="1.0" />
                    </GradientStopCollection>
                  </GradientBrush.GradientStops>
                </LinearGradientBrush>
              </Border.BorderBrush>
            </Border>
          </VisualState>
        </Grid>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>

```

```

        </LinearGradientBrush>

    </Border.BorderBrush>
    <Grid ClipToBounds="True">
        <x:Name="Animation">
            <Rectangle x:Name="PART_GlowRect"
                Width="100"
                HorizontalAlignment="Left"
                Fill="{DynamicResource
ProgressBarIndicatorAnimatedFill}"
                Margin="-100,0,0,0" />
        </Grid>
    </Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
<Setter Property="Background">
<Setter.Value>
    <LinearGradientBrush EndPoint="0,1"
        StartPoint="0,0">
        <GradientStop Color="{DynamicResource ControlLightColor}"
            Offset="0" />
        <GradientStop Color="{DynamicResource ControlMediumColor}"
            Offset="1" />
    </LinearGradientBrush>
</Setter.Value>
</Setter>
<Setter Property="Foreground">
<Setter.Value>
    <LinearGradientBrush EndPoint="0.5,1"
        StartPoint="0.5,0">
        <GradientStop Color="{DynamicResource ControlMediumColor}"
            Offset="0" />
        <GradientStop Color="{DynamicResource ControlDarkColor}"
            Offset="1" />
    </LinearGradientBrush>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

```

```
<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

```
        Offset="0.4" />
    <GradientStop Color="#600000FF"
        Offset="0.6" />
    <GradientStop Color="#000000FF"
        Offset="1" />
</GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

RadioButton 样式和模板

项目 • 2023/02/06

本主题介绍 [RadioButton](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

RadioButton 部件

[RadioButton](#) 控件没有任何命名部件。

RadioButton 状态

下表列出了 [RadioButton](#) 控件的可视状态。

VisualState 名称 名称	VisualStateGroup	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在控件上方。
已按下	CommonStates	已按下控件。
已禁用	CommonStates	已禁用控件。
已设定焦点	FocusStates	控件有焦点。
失去焦点	FocusStates	控件没有焦点。
已选中	CheckStates	<code>IsChecked</code> 上声明的默认值为 <code>true</code> 。
未选中	CheckStates	<code>IsChecked</code> 上声明的默认值为 <code>false</code> 。
不确定	CheckStates	<code>IsThreeState</code> 为 <code>true</code> 且 <code>IsChecked</code> 为 <code>null</code> 。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	如果 Validation.HasError 附加属性为 <code>true</code> ，该控件没有焦点。

RadioButton ControlTemplate 示例

以下示例演示如何定义 RadioButton 控件的 ControlTemplate。

XAML

```
<Style x:Key="{x:Type RadioButton}"
    TargetType="{x:Type RadioButton}">
<Setter Property="SnapsToDevicePixels"
    Value="true" />
<Setter Property="OverridesDefaultStyle"
    Value="true" />
<Setter Property="FocusVisualStyle"
    Value="{DynamicResource RadioButtonFocusVisual}" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type RadioButton}">
            <BulletDecorator Background="Transparent">
                <BulletDecorator.Bullet>
                    <Grid Width="13"
                        Height="13">
                        <Ellipse x:Name="Border"
                            StrokeThickness="1">
                            <Ellipse.Stroke>
                                <LinearGradientBrush EndPoint="0.5,1"
                                    StartPoint="0.5,0">
                                    <GradientStop Color="{DynamicResource BorderLightColor}"
                                        Offset="0" />
                                    <GradientStop Color="{DynamicResource BorderDarkColor}"
                                        Offset="1" />
                                </LinearGradientBrush>
                            </Ellipse.Stroke>
                        <Ellipse.Fill>
                            <LinearGradientBrush StartPoint="0,0"
                                EndPoint="0,1">
                                <LinearGradientBrush.GradientStops>
                                    <GradientStopCollection>
                                        <GradientStop Color="{DynamicResource
ControlLightColor}" />
                                        <GradientStop Color="{DynamicResource
ControlMediumColor}" />
                                    <GradientStopCollection>
                                </LinearGradientBrush.GradientStops>
                            </LinearGradientBrush>
                        </Ellipse.Fill>
                    </Grid>
                </BulletDecorator.Bullet>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

```

<VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
        <VisualState x:Name="Normal" />
        <VisualState x:Name="MouseOver">
            <Storyboard>
                <ColorAnimationUsingKeyFrames
                    Storyboard.TargetName="Border"
                    Storyboard.TargetProperty=""
                    (Shape.Fill).
                        (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource
ControlMouseOverColor}" />
                        </ColorAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
                <VisualState x:Name="Pressed">
                    <Storyboard>
                        <ColorAnimationUsingKeyFrames
                            Storyboard.TargetName="Border"
                            Storyboard.TargetProperty=""
                            (Shape.Fill).
                                (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                                    <EasingColorKeyFrame KeyTime="0"
                                        Value="{StaticResource
ControlPressedColor}" />
                                </ColorAnimationUsingKeyFrames>
                            </Storyboard>
                        </VisualState>
                        <VisualState x:Name="Disabled">
                            <Storyboard>
                                <ColorAnimationUsingKeyFrames
                                    Storyboard.TargetName="Border"
                                    Storyboard.TargetProperty=""
                                    (Shape.Fill).
                                        (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                                            <EasingColorKeyFrame KeyTime="0"
                                                Value="{StaticResource
ControlLightColor}" />
                                        </ColorAnimationUsingKeyFrames>
                                    <ColorAnimationUsingKeyFrames
                                        Storyboard.TargetName="Border"
                                        Storyboard.TargetProperty=""
                                        (Shape.Stroke).
                                            (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                                                <EasingColorKeyFrame KeyTime="0"
                                                    Value="#40000000" />
                                            </ColorAnimationUsingKeyFrames>
                                        <ColorAnimationUsingKeyFrames
                                            Storyboard.TargetName="Border"
                                            Storyboard.TargetProperty=""
                                            (Shape.Stroke).
                                                (GradientBrush.GradientStops)[0].(GradientStop.Color)">
                                                    <EasingColorKeyFrame KeyTime="0"
                                                        Value="#40000000" />
                                                </ColorAnimationUsingKeyFrames>
                                            </ColorAnimationUsingKeyFrames>
                                        </ColorAnimationUsingKeyFrames>
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                        </VisualState>
                    </Storyboard>
                </VisualState>
            </Storyboard>
        </VisualState>
    </VisualStateGroup>
</VisualStateManager>

```

```

        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="CheckStates">
    <VisualState x:Name="Checked">
        <Storyboard>
            <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="
(UIElement.Visibility)">

Storyboard.TargetName="CheckMark">
            <DiscreteObjectKeyFrame KeyTime="0"
                Value="{x:Static
Visibility.Visible}" />
            </ObjectAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
    <VisualState x:Name="Unchecked" />
    <VisualState x:Name="Indeterminate" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<ContentPresenter Margin="4,0,0,0"
    VerticalAlignment="Center"
    HorizontalAlignment="Left"
    RecognizesAccessKey="True" />
</BulletDecorator>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

```

```

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

RepeatButton 样式和模板

项目 · 2022/09/27

本主题介绍 [RepeatButton](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

RepeatButton 部件

[RepeatButton](#) 控件没有任何已命名的部件。

RepeatButton 状态

下表列出了 [RepeatButton](#) 控件的视觉对象状态。

VisualState 名称 名称	VisualStateGroup	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在控件上方。
已按下	CommonStates	已按下控件。
已禁用	CommonStates	已禁用控件。
已设定焦点	FocusStates	控件有焦点。
失去焦点	FocusStates	控件没有焦点。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

RepeatButton ControlTemplate 示例

下例演示如何定义 [RepeatButton](#) 控件的 [ControlTemplate](#)。

XAML

```

<Style x:Key="ScrollBarLineButton"
    TargetType="{x:Type RepeatButton}">
    <Setter Property="SnapsToDevicePixels"
        Value="True" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Focusable"
        Value="false" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type RepeatButton}">
                <Border x:Name="Border"
                    Margin="1"
                    CornerRadius="2"
                    BorderThickness="1">
                    <Border.BorderBrush>
                        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStopCollection>
                                    <GradientStop Color="{DynamicResource BorderMediumColor}"
                                        Offset="0.0" />
                                    <GradientStop Color="{DynamicResource BorderDarkColor}"
                                        Offset="1.0" />
                                </GradientStopCollection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Border.BorderBrush>
                    <Border.Background>
                        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStopCollection>
                                    <GradientStop Color="{DynamicResource
ControlLightColor}"/>
                                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                                        Offset="1.0" />
                                </GradientStopCollection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Border.Background>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="MouseOver" />
                            <VisualState x:Name="Pressed">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Border"
Storyboard.TargetProperty="

(Panel.Background).(
(GradientBrush.GradientStops)[1].(GradientStop.Color)">
<EasingColorKeyFrame KeyTime="0"
Value="{StaticResource

```

```

        ControlPressedColor}" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
<VisualState x:Name="Disabled">
    <Storyboard>
        <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Arrow"
Storyboard.TargetProperty="

(Shape.Fill).

(SolidColorBrush.Color)">
            <EasingColorKeyFrame KeyTime="0"
Value="{StaticResource

DisabledForegroundColorBrush}" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Path x:Name="Arrow"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"
    Data="{Binding Content,
    RelativeSource={RelativeSource TemplatedParent}}}" >
    <Path.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Fill>
</Path>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

```

```

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
                     EndPoint="0.5,1"
                     StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
                  Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
                  Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
                  Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
                     StartPoint="0,0"
                     EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                          Offset="0" />
            <GradientStop Color="#600000FF"
                          Offset="0.4" />
            <GradientStop Color="#600000FF"
                          Offset="0.6" />
            <GradientStop Color="#000000FF"
                          Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

```
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

ScrollBar 样式和模板

项目 • 2023/02/06

本主题介绍 [ScrollBar](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

ScrollBar 部件

下表列出了 [ScrollBar](#) 控件的命名部件。

组成部分	类型	描述
PART_Track	Track	指示 ScrollBar 位置的元素的容器。

ScrollBar 状态

下表列出了 [ScrollBar](#) 控件的可视状态。

VisualState 名称 名称	VisualStateGroup 名称	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在控件上方。
已禁用	CommonStates	已禁用控件。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是 <code>true</code> ，并且控件具有焦点。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性为 <code>true</code> ，控件不具有焦点。

ScrollBar ControlTemplate 示例

以下示例演示如何定义 [ScrollBar](#) 控件的 [ControlTemplate](#)。

XAML

```

<Style x:Key="ScrollBarLineButton"
    TargetType="{x:Type RepeatButton}">
    <Setter Property="SnapsToDevicePixels"
        Value="True" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Focusable"
        Value="false" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type RepeatButton}">
                <Border x:Name="Border"
                    Margin="1"
                    CornerRadius="2"
                    BorderThickness="1">
                    <Border.BorderBrush>
                        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStopCollection>
                                    <GradientStop Color="{DynamicResource BorderMediumColor}"
                                        Offset="0.0" />
                                    <GradientStop Color="{DynamicResource BorderDarkColor}"
                                        Offset="1.0" />
                                </GradientStopCollection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Border.BorderBrush>
                    <Border.Background>
                        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStopCollection>
                                    <GradientStop Color="{DynamicResource
ControlLightColor}"/>
                                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                                        Offset="1.0" />
                                </GradientStopCollection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Border.Background>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="MouseOver" />
                            <VisualState x:Name="Pressed">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Border"
Storyboard.TargetProperty="

(Panel.Background).(
(GradientBrush.GradientStops)[1].(GradientStop.Color)">
<EasingColorKeyFrame KeyTime="0"
Value="{StaticResource

```

```

        ControlPressedColor}" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
<VisualState x:Name="Disabled">
    <Storyboard>
        <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Arrow"
Storyboard.TargetProperty="

(Shape.Fill).
        (SolidColorBrush.Color)">
        <EasingColorKeyFrame KeyTime="0"
Value="{StaticResource

DisabledForegroundColorBrush}" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Path x:Name="Arrow"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"
    Data="{Binding Content,
    RelativeSource={RelativeSource TemplatedParent}}}" >
    <Path.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Fill>
</Path>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="ScrollBarPageButton"
    TargetType="{x:Type RepeatButton}">
<Setter Property="SnapsToDevicePixels"
    Value="True" />
<Setter Property="OverridesDefaultStyle"
    Value="true" />
<Setter Property="IsTabStop"
    Value="false" />
<Setter Property="Focusable"
    Value="false" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type RepeatButton}">
            <Border Background="Transparent" />
        </ControlTemplate>
    </Setter.Value>
</Setter>
</Style>

<Style x:Key="ScrollBarThumb"
    TargetType="{x:Type Thumb}">

```

```

<Setter Property="SnapsToDevicePixels"
        Value="True" />
<Setter Property="OverridesDefaultStyle"
        Value="true" />
<Setter Property="IsTabStop"
        Value="false" />
<Setter Property="Focusable"
        Value="false" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type Thumb}">
            <Border CornerRadius="2"
                    Background="{TemplateBinding Background}"
                    BorderBrush="{TemplateBinding BorderBrush}"
                    BorderThickness="1" />
        </ControlTemplate>
    </Setter.Value>
</Setter>
</Style>

<ControlTemplate x:Key="VerticalScrollBar"
                  TargetType="{x:Type ScrollBar}">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition MaxHeight="18" />
            <RowDefinition Height="0.0001*" />
            <RowDefinition MaxHeight="18" />
        </Grid.RowDefinitions>
        <Border Grid.RowSpan="3"
                CornerRadius="2"
                Background="#F0F0F0" />
        <RepeatButton Grid.Row="0"
                      Style="{StaticResource ScrollBarLineButton}"
                      Height="18"
                      Command="ScrollBar.LineUpCommand"
                      Content="M 0 4 L 8 4 L 4 0 Z" />
        <Track x:Name="PART_Track"
                Grid.Row="1"
                IsDirectionReversed="true">
            <Track.DecreaseRepeatButton>
                <RepeatButton Style="{StaticResource ScrollBarPageButton}"
                              Command="ScrollBar.PageUpCommand" />
            </Track.DecreaseRepeatButton>
            <Track.Thumb>
                <Thumb Style="{StaticResource ScrollBarThumb}"
                      Margin="1,0,1,0">
                    <Thumb.BorderBrush>
                        <LinearGradientBrush StartPoint="0,0"
                                            EndPoint="1,0">
                            <LinearGradientBrush.GradientStops>
                                <GradientStopCollection>
                                    <GradientStop Color="{DynamicResource BorderLightColor}"
                                                  Offset="0.0" />
                                    <GradientStop Color="{DynamicResource BorderDarkColor}" />
                                </GradientStopCollection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Thumb.BorderBrush>
                </Thumb>
            </Track.Thumb>
        </Track>
    </Grid>
</ControlTemplate>

```

```

                Offset="1.0" />
            </GradientStopCollection>
        </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>

</Thumb.BorderBrush>
<Thumb.Background>

    <LinearGradientBrush StartPoint="0,0"
                           EndPoint="1,0">
        <LinearGradientBrush.GradientStops>
            <GradientStopCollection>
                <GradientStop Color="{DynamicResource ControlLightColor}"
                               Offset="0.0" />
                <GradientStop Color="{DynamicResource ControlMediumColor}"
                               Offset="1.0" />
            </GradientStopCollection>
        </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>

</Thumb.Background>
</Thumb>
</Track.Thumb>
<Track.IncreaseRepeatButton>
    <RepeatButton Style="{StaticResource ScrollBarButton}"
                  Command="ScrollBar.PageDownCommand" />
</Track.IncreaseRepeatButton>
</Track>
<RepeatButton Grid.Row="2"
              Style="{StaticResource ScrollBarLineButton}"
              Height="18"
              Command="ScrollBar.LineDownCommand"
              Content="M 0 0 L 4 4 L 8 0 Z" />
</Grid>
</ControlTemplate>

<ControlTemplate x:Key="HorizontalScrollBar"
                 TargetType="{x:Type ScrollBar}">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition MaxWidth="18" />
            <ColumnDefinition Width="0.0001*" />
            <ColumnDefinition MaxWidth="18" />
        </Grid.ColumnDefinitions>
        <Border Grid.ColumnSpan="3"
                CornerRadius="2"
                Background="#F0F0F0" />
        <RepeatButton Grid.Column="0"
                     Style="{StaticResource ScrollBarLineButton}"
                     Width="18"
                     Command="ScrollBar.LineLeftCommand"
                     Content="M 4 0 L 4 8 L 0 4 Z" />
        <Track x:Name="PART_Track"
               Grid.Column="1"
               IsDirectionReversed="False">

```

```

<Track.DecreaseRepeatButton>
    <RepeatButton Style="{StaticResource ScrollBarButton}"
        Command="ScrollBar.PageLeftCommand" />
</Track.DecreaseRepeatButton>
<Track.Thumb>
    <Thumb Style="{StaticResource ScrollBarThumb}"
        Margin="0,1,0,1">

        <Thumb.BorderBrush>

            <LinearGradientBrush StartPoint="0,0"
                EndPoint="1,0">
                <LinearGradientBrush.GradientStops>
                    <GradientStopCollection>
                        <GradientStop Color="{DynamicResource BorderLightColor}"
                            Offset="0.0" />
                        <GradientStop Color="{DynamicResource BorderDarkColor}"
                            Offset="1.0" />
                    </GradientStopCollection>
                </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>

        </Thumb.BorderBrush>
        <Thumb.Background>

            <LinearGradientBrush StartPoint="0,0"
                EndPoint="0,1">
                <LinearGradientBrush.GradientStops>
                    <GradientStopCollection>
                        <GradientStop Color="{DynamicResource ControlLightColor}"
                            Offset="0.0" />
                        <GradientStop Color="{DynamicResource ControlMediumColor}"
                            Offset="1.0" />
                    </GradientStopCollection>
                </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>

        </Thumb.Background>
    </Thumb>
</Track.Thumb>
<Track.IncreaseRepeatButton>
    <RepeatButton Style="{StaticResource ScrollBarButton}"
        Command="ScrollBar.PageRightCommand" />
</Track.IncreaseRepeatButton>
</Track>
<RepeatButton Grid.Column="2"
    Style="{StaticResource ScrollBarLineButton}"
    Width="18"
    Command="ScrollBar.LineRightCommand"
    Content="M 0 0 L 4 4 L 0 8 Z" />

</Grid>
</ControlTemplate>

<Style x:Key="{x:Type ScrollBar}"
    TargetType="{x:Type ScrollBar}">

```

```

<Setter Property="SnapsToDevicePixels"
        Value="True" />
<Setter Property="OverridesDefaultStyle"
        Value="true" />
<Style.Triggers>
    <Trigger Property="Orientation"
              Value="Horizontal">
        <Setter Property="Width"
                  Value="Auto" />
        <Setter Property="Height"
                  Value="18" />
        <Setter Property="Template"
                  Value="{StaticResource HorizontalScrollBar}" />
    </Trigger>
    <Trigger Property="Orientation"
              Value="Vertical">
        <Setter Property="Width"
                  Value="18" />
        <Setter Property="Height"
                  Value="Auto" />
        <Setter Property="Template"
                  Value="{StaticResource VerticalScrollBar}" />
    </Trigger>
</Style.Triggers>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

```

```
<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- Style
- ControlTemplate
- 控件样式和模板
- 控件自定义
- 样式设置和模板化
- 创建控件模板

ScrollViewer 样式和模板

项目 • 2023/02/06

本主题介绍 [ScrollViewer](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

ScrollViewer 部件

下表列出了 [ScrollViewer](#) 控件的已命名部件。

组成部分	类型	描述
PART_ScrollContentPresenter	ScrollContentPresenter	ScrollViewer 中内容的占位符。
PART_HorizontalScrollBar	ScrollBar	用于水平滚动内容的 ScrollBar 。
PART_VerticalScrollBar	ScrollBar	用于垂直滚动内容的 ScrollBar 。

ScrollViewer 状态

下表列出了 [ScrollViewer](#) 控件的视觉对象状态。

VisualState 名称	VisualStateGroup 名称	描述
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

ScrollViewer ControlTemplate 示例

下例演示如何定义 [ScrollViewer](#) 控件的 [ControlTemplate](#)。

XAML

```
<Style x:Key="LeftScrollViewer"  
       TargetType="{x:Type ScrollViewer}">  
    <Setter Property="OverridesDefaultStyle"  
           Value="True" />
```

```

        Value="True" />
<Setter Property="Template">
<Setter.Value>
    <ControlTemplate TargetType="{x:Type ScrollViewer}">
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition />
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition />
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>
            <Border Grid.Column="1"
                    BorderThickness="0,1,1,1">
                <Border.BorderBrush>
                    <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                </Border.BorderBrush>
                <ScrollContentPresenter CanContentScroll="{TemplateBinding CanContentScroll}" />
            </Border>
            <ScrollBar x:Name="PART_VerticalScrollBar"
                       Value="{TemplateBinding VerticalOffset}"
                       Maximum="{TemplateBinding ScrollableHeight}"
                       ViewportSize="{TemplateBinding ViewportHeight}"
                       Visibility="{TemplateBinding ComputedVerticalScrollBarVisibility}"/>
            <ScrollBar x:Name="PART_HorizontalScrollBar"
                       Orientation="Horizontal"
                       Grid.Row="1"
                       Grid.Column="1"
                       Value="{TemplateBinding HorizontalOffset}"
                       Maximum="{TemplateBinding ScrollableWidth}"
                       ViewportSize="{TemplateBinding ViewportWidth}"
                       Visibility="{TemplateBinding ComputedHorizontalScrollBarVisibility}"/>
        </Grid>
    </ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

```

```

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
                     EndPoint="0.5,1"
                     StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
                  Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
                  Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
                  Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
                     StartPoint="0,0"
                     EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>

```

```
<GradientStop Color="#000000FF"
               Offset="0" />
<GradientStop Color="#600000FF"
               Offset="0.4" />
<GradientStop Color="#600000FF"
               Offset="0.6" />
<GradientStop Color="#000000FF"
               Offset="1" />
</GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

Slider 样式和模板

项目 • 2023/02/06

本主题介绍 [Slider](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

滑块部件

下表列出了 [Slider](#) 控件的命名部件。

组成部分	类型	描述
PART_Track	Track	指示 Slider 位置的元素的容器。
PART_SelectionRange	FrameworkElement	沿 Slider 显示选择范围的元素。仅当 IsSelectionRangeEnabled 属性为 <code>true</code> 时，选择范围才可见。

滑块状态

下表列出了 [Slider](#) 控件的视觉对象状态。

VisualState 名称 名称	VisualStateGroup 名称	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在控件上方。
已禁用	CommonStates	已禁用控件。
已设定焦点	FocusStates	控件有焦点。
失去焦点	FocusStates	控件没有焦点。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

Slider ControlTemplate 示例

以下示例演示如何定义 `Slider` 控件的 `ControlTemplate`。

XAML

```
<Style x:Key="SliderButtonStyle"
    TargetType="{x:Type RepeatButton}">
<Setter Property="SnapsToDevicePixels"
    Value="true" />
<Setter Property="OverridesDefaultStyle"
    Value="true" />
<Setter Property="IsTabStop"
    Value="false" />
<Setter Property="Focusable"
    Value="false" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type RepeatButton}">
            <Border Background="Transparent" />
        </ControlTemplate>
    </Setter.Value>
</Setter>
</Style>

<Style x:Key="SliderThumbStyle"
    TargetType="{x:Type Thumb}">
<Setter Property="SnapsToDevicePixels"
    Value="true" />
<Setter Property="OverridesDefaultStyle"
    Value="true" />
<Setter Property="Height"
    Value="14" />
<Setter Property="Width"
    Value="14" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type Thumb}">
            <Ellipse x:Name="Ellipse"
                StrokeThickness="1">
                <Ellipse.Stroke>
                    <LinearGradientBrush StartPoint="0,0"
                        EndPoint="0,1">
                        <LinearGradientBrush.GradientStops>
                            <GradientStop Collection>
                                <GradientStop Color="{DynamicResource BorderLightColor}"
                                    Offset="0.0" />
                                <GradientStop Color="{DynamicResource BorderDarkColor}"
                                    Offset="1.0" />
                            </GradientStop Collection>
                        </LinearGradientBrush.GradientStops>
                    </LinearGradientBrush>
                </Ellipse.Stroke>
            </Ellipse>
        </ControlTemplate>
    </Setter.Value>
</Setter>
</Style>
```

```

<Ellipse.Fill>
    <LinearGradientBrush EndPoint="0.5,1"
        StartPoint="0.5,0">
        <GradientStop Color="{DynamicResource ControlMediumColor}"
            Offset="1" />
        <GradientStop Color="{DynamicResource ControlLightColor}" />
    </LinearGradientBrush>
</Ellipse.Fill>
<VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
        <VisualState x:Name="Normal" />
        <VisualState x:Name="MouseOver">
            <Storyboard>
                <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="
(Shape.Fill).(GradientBrush.GradientStops)[0].(GradientStop.Color)">
                    Storyboard.TargetName="Ellipse">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource
ControlMouseOverColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
            <VisualState x:Name="Pressed">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="
(Shape.Fill).(GradientBrush.GradientStops)[0].(GradientStop.Color)">
                        Storyboard.TargetName="Ellipse">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource
ControlPressedColor}" />
                        </ColorAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
                <VisualState x:Name="Disabled">
                    <Storyboard>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="
(Shape.Fill).(GradientBrush.GradientStops)[0].(GradientStop.Color)">
                            Storyboard.TargetName="Ellipse">
                                <EasingColorKeyFrame KeyTime="0"
                                    Value="{StaticResource
DisabledControlDarkColor}" />
                            </ColorAnimationUsingKeyFrames>
                        </Storyboard>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateManager.VisualStateGroups>
        </Ellipse>
    </ControlTemplate>
</Setter.Value>

```

```

        </Setter>
    </Style>

    <!--Template when the orientation of the Slider is Horizontal.-->
    <ControlTemplate x:Key="HorizontalSlider"
                      TargetType="{x:Type Slider}">
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto"
                               MinHeight="{TemplateBinding MinHeight}" />
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>
            <TickBar x:Name="TopTick"
                     SnapsToDevicePixels="True"
                     Placement="Top"
                     Height="4"
                     Visibility="Collapsed">
                <TickBar.Fill>
                    <SolidColorBrush Color="{DynamicResource GlyphColor}" />
                </TickBar.Fill>
            </TickBar>
            <Border x:Name="TrackBackground"
                    Margin="0"
                    CornerRadius="2"
                    Height="4"
                    Grid.Row="1"
                    BorderThickness="1">
                <Border.BorderBrush>
                    <LinearGradientBrush StartPoint="0,0"
                                         EndPoint="0,1">
                        <LinearGradientBrush.GradientStops>
                            <GradientStopCollection>
                                <GradientStop Color="{DynamicResource BorderLightColor}"
                                              Offset="0.0" />
                                <GradientStop Color="{DynamicResource BorderDarkColor}"
                                              Offset="1.0" />
                            </GradientStopCollection>
                        </LinearGradientBrush.GradientStops>
                    </LinearGradientBrush>
                </Border.BorderBrush>
                <Border.Background>
                    <LinearGradientBrush StartPoint="0,0"
                                         EndPoint="0,1">
                        <LinearGradientBrush.GradientStops>
                            <GradientStopCollection>
                                <GradientStop Color="{DynamicResource ControlLightColor}"
                                              Offset="0.0" />
                                <GradientStop Color="{DynamicResource SliderTrackDarkColor}"
                                              Offset="1.0" />
                            </GradientStopCollection>
                        </LinearGradientBrush.GradientStops>
                    </LinearGradientBrush>
                </Border.Background>
            </Border>
        </Grid>
    </ControlTemplate>

```

```

<Track Grid.Row="1"
      x:Name="PART_Track">
    <Track.DecreaseRepeatButton>
      <RepeatButton Style="{StaticResource SliderButtonStyle}"
                    Command="Slider.DecreaseLarge" />
    </Track.DecreaseRepeatButton>
    <Track.Thumb>
      <Thumb Style="{StaticResource SliderThumbStyle}" />
    </Track.Thumb>
    <Track.IncreaseRepeatButton>
      <RepeatButton Style="{StaticResource SliderButtonStyle}"
                    Command="Slider.IncreaseLarge" />
    </Track.IncreaseRepeatButton>
  </Track>
  <TickBar x:Name="BottomTick"
            SnapsToDevicePixels="True"
            Grid.Row="2"
            Fill="{TemplateBinding Foreground}"
            Placement="Bottom"
            Height="4"
            Visibility="Collapsed" />
</Grid>
<ControlTemplate.Triggers>
  <Trigger Property="TickPlacement"
           Value="TopLeft">
    <Setter TargetName="TopTick"
              Property="Visibility"
              Value="Visible" />
  </Trigger>
  <Trigger Property="TickPlacement"
           Value="BottomRight">
    <Setter TargetName="BottomTick"
              Property="Visibility"
              Value="Visible" />
  </Trigger>
  <Trigger Property="TickPlacement"
           Value="Both">
    <Setter TargetName="TopTick"
              Property="Visibility"
              Value="Visible" />
    <Setter TargetName="BottomTick"
              Property="Visibility"
              Value="Visible" />
  </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>

<!--Template when the orientation of the Slider is Vertical.-->
<ControlTemplate x:Key="VerticalSlider"
                  TargetType="{x:Type Slider}">
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto" />
      <ColumnDefinition Width="Auto"
                        MinWidth="{TemplateBinding MinWidth}" />

```

```

        <ColumnDefinition Width="Auto" />
    </Grid.ColumnDefinitions>
    <TickBar x:Name="TopTick"
        SnapsToDevicePixels="True"
        Placement="Left"
        Width="4"
        Visibility="Collapsed">
        <TickBar.Fill>
            <SolidColorBrush Color="{DynamicResource GlyphColor}" />
        </TickBar.Fill>
    </TickBar>

    <Border x:Name="TrackBackground"
        Margin="0"
        CornerRadius="2"
        Width="4"
        Grid.Column="1"
        BorderThickness="1">
        <Border.BorderBrush>
            <LinearGradientBrush StartPoint="0,0"
                EndPoint="1,0">
                <LinearGradientBrush.GradientStops>
                    <GradientStopCollection>
                        <GradientStop Color="{DynamicResource BorderLightColor}"
                            Offset="0.0" />
                        <GradientStop Color="{DynamicResource BorderDarkColor}"
                            Offset="1.0" />
                    </GradientStopCollection>
                </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
        </Border.BorderBrush>
        <Border.Background>
            <LinearGradientBrush EndPoint="1,0"
                StartPoint="0.25,0">
                <GradientStop Color="{DynamicResource ControlLightColor}"
                    Offset="0" />
                <GradientStop Color="{DynamicResource SliderTrackDarkColor}"
                    Offset="1" />
            </LinearGradientBrush>
        </Border.Background>
    </Border>
    <Track Grid.Column="1"
        x:Name="PART_Track">
        <Track.DecreaseRepeatButton>
            <RepeatButton Style="{StaticResource SliderButtonStyle}"
                Command="Slider.DecreaseLarge" />
        </Track.DecreaseRepeatButton>
        <Track.Thumb>
            <Thumb Style="{StaticResource SliderThumbStyle}" />
        </Track.Thumb>
        <Track.IncreaseRepeatButton>
            <RepeatButton Style="{StaticResource SliderButtonStyle}"
                Command="Slider.IncreaseLarge" />
        </Track.IncreaseRepeatButton>
    </Track>

```

```

        </Track>
        <TickBar x:Name="BottomTick"
                  SnapsToDevicePixels="True"
                  Grid.Column="2"
                  Fill="{TemplateBinding Foreground}"
                  Placement="Right"
                  Width="4"
                  Visibility="Collapsed" />
    </Grid>
    <ControlTemplate.Triggers>
        <Trigger Property="TickPlacement"
                  Value="TopLeft">
            <Setter TargetName="TopTick"
                      Property="Visibility"
                      Value="Visible" />
        </Trigger>
        <Trigger Property="TickPlacement"
                  Value="BottomRight">
            <Setter TargetName="BottomTick"
                      Property="Visibility"
                      Value="Visible" />
        </Trigger>
        <Trigger Property="TickPlacement"
                  Value="Both">
            <Setter TargetName="TopTick"
                      Property="Visibility"
                      Value="Visible" />
            <Setter TargetName="BottomTick"
                      Property="Visibility"
                      Value="Visible" />
        </Trigger>
    </ControlTemplate.Triggers>
</ControlTemplate>

<Style TargetType="{x:Type Slider}">
    <Setter Property="SnapsToDevicePixels"
              Value="true" />
    <Setter Property="OverridesDefaultStyle"
              Value="true" />
    <Style.Triggers>
        <Trigger Property="Orientation"
                  Value="Horizontal">
            <Setter Property="MinWidth"
                      Value="104" />
            <Setter Property="MinHeight"
                      Value="21" />
            <Setter Property="Template"
                      Value="{StaticResource HorizontalSlider}" />
        </Trigger>
        <Trigger Property="Orientation"
                  Value="Vertical">
            <Setter Property="MinWidth"
                      Value="21" />
            <Setter Property="MinHeight"
                      Value="104" />
        </Trigger>
    </Style.Triggers>
</Style>

```

```
<Setter Property="Template"
       Value="{StaticResource VerticalSlider}" />
</Trigger>
</Style.Triggers>
</Style>
```

上一示例使用了一个或多个以下资源。

XAML

```
<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>
```

```
<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

StatusBar 样式和模板

项目 • 2023/02/06

本主题介绍 [StatusBar](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

StatusBar 部件

[StatusBar](#) 控件没有任何已命名的部件。

StatusBar 状态

下表列出了 [StatusBar](#) 控件的视觉对象状态。

VisualState 名称 名称	VisualStateGroup	描述
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

StatusBarItem 部件

[StatusBarItem](#) 控件没有任何已命名的部件。

StatusBarItem 状态

下表列出了 [StatusBarItem](#) 控件的视觉对象状态。

VisualState 名称 名称	VisualStateGroup	描述
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。

VisualState 名称	VisualStateGroup 名称	描述
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 true。

StatusBar ControlTemplate 示例

下例演示如何定义 [StatusBar](#) 控件的 [ControlTemplate](#)。

XAML

```

<Style x:Key="{x:Type StatusBar}"
       TargetType="{x:Type StatusBar}">
  <Setter Property="SnapsToDevicePixels"
         Value="True" />
  <Setter Property="OverridesDefaultStyle"
         Value="true" />
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="{x:Type StatusBar}">
        <Border Padding="1">
          <Border.BorderBrush>
            <LinearGradientBrush StartPoint="0,0"
                               EndPoint="0,1">
              <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                  <GradientStop Color="{DynamicResource BorderLightColor}"
                                 Offset="0.0" />
                  <GradientStop Color="{DynamicResource BorderDarkColor}"
                                 Offset="1.0" />
                </GradientStopCollection>
              </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
          </Border.BorderBrush>
          <Border.Background>
            <LinearGradientBrush StartPoint="0,0"
                               EndPoint="0,1">
              <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                  <GradientStop Color="{DynamicResource ControlLightColor}"
                                 Offset="0.0" />
                  <GradientStop Color="{DynamicResource ControlMediumColor}"
                                 Offset="1.0" />
                </GradientStopCollection>
              </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
          </Border.Background>
          <ItemsPresenter />
        </Border>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>

```

```

        </ControlTemplate>
    </Setter.Value>
</Setter>
</Style>

<Style x:Key="{x:Static StatusBar.SeparatorStyleKey}"
    TargetType="{x:Type Separator}">
<Setter Property="OverridesDefaultStyle"
    Value="True" />
<Setter Property="SnapsToDevicePixels"
    Value="True" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type Separator}">
            <Rectangle Width="1"
                Margin="3">
                <Rectangle.Fill>
                    <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                </Rectangle.Fill>
            </Rectangle>
        </ControlTemplate>
    </Setter.Value>
</Setter>
</Style>

<Style x:Key="{x:Type StatusBarItem}"
    TargetType="{x:Type StatusBarItem}">
<Setter Property="OverridesDefaultStyle"
    Value="True" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type StatusBarItem}">
            <ContentPresenter Margin="3"
                Name="ContentSite" />
            <ControlTemplate.Triggers>
                <Trigger Property="IsEnabled"
                    Value="false">
                    <Setter Property="Foreground">
                        <Setter.Value>
                            <SolidColorBrush Color="{StaticResource
DisabledForegroundColor}" />
                        </Setter.Value>
                    </Setter>
                </Trigger>
            </ControlTemplate.Triggers>
        </ControlTemplate>
    </Setter.Value>
</Setter>
</Style>

```

ControlTemplate 使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

```

```
<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

TabControl 样式和模板

项目 • 2023/02/06

本主题介绍 [TabControl](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

TabControl 部件

下表列出了 [TabControl](#) 控件的已命名部件。

组成部分	类型	描述
PART_SelectedContentHost	ContentPresenter	显示当前所选的 TabItem 的内容的对象。

为 [TabControl](#) 创建一个 [ControlTemplate](#) 时，你的模板可能在 [ScrollViewer](#) 中包含一个 [ItemsPresenter](#)。（[ItemsPresenter](#) 显示 [TabControl](#) 中的每个项；[ScrollViewer](#) 支持在控件内滚动）。如果 [ItemsPresenter](#) 不是 [ScrollViewer](#) 的直接子元素，则必须为 [ItemsPresenter](#) 指定名称 `ItemsPresenter`。

TabControl 状态

下表列出了 [TabControl](#) 控件的视觉对象状态。

VisualState 名称 名称	VisualStateGroup	描述
普通	CommonStates	默认状态。
已禁用	CommonStates	已禁用控件。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

TabItem 部件

[TabItem](#) 控件没有任何已命名的部件。

TabItem 状态

下表列出了 TabItem 控件的视觉对象状态。

VisualState 名称 名称	VisualStateGroup	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在控件上方。
已禁用	CommonStates	已禁用控件。
已设定焦点	FocusStates	控件有焦点。
失去焦点	FocusStates	控件没有焦点。
选定	SelectionStates	已选择控件。
未选定	SelectionStates	未选择控件。
有效	ValidationStates	该控件使用 Validation 类 , Validation.HasError 附加属性为 false。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 true。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 true。

TabControl ControlTemplate 示例

下面的示例演示如何为 TabControl 和 TabItem 控件定义 ControlTemplate。

XAML

```
<Style TargetType="{x:Type TabControl}">
    <Setter Property="OverridesDefaultStyle"
        Value="True" />
    <Setter Property="SnapsToDevicePixels"
        Value="True" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type TabControl}">
                <Grid KeyboardNavigation.TabNavigation="Local">
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto" />
                        <RowDefinition Height="*" />
                    </Grid.RowDefinitions>
                    <VisualStateManager.VisualStateGroups>
```

```

<VisualStateGroup x:Name="CommonStates">
    <VisualState x:Name="Disabled">
        <Storyboard>
            <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Border" Storyboard.TargetProperty="

(Border.BorderBrush).SolidColorBrush.Color)">
                <EasingColorKeyFrame KeyTime="0"
Value="#FFAAAAAA" />
            </ColorAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<TabPanel x:Name="HeaderPanel">
    Grid.Row="0"
    Panel.ZIndex="1"
    Margin="0,0,4,-1"
    IsItemsHost="True"
    KeyboardNavigation.TabIndex="1"
    Background="Transparent" />
<Border x:Name="Border">
    Grid.Row="1"
    BorderThickness="1"
    CornerRadius="2"
    KeyboardNavigation.TabNavigation="Local"
    KeyboardNavigation.DirectionalNavigation="Contained"
    KeyboardNavigation.TabIndex="2">
    <Border.Background>
        <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
            <GradientStop Color="{DynamicResource
ContentAreaColorLight}" Offset="0" />
            <GradientStop Color="{DynamicResource ContentAreaColorDark}" Offset="1" />
        </LinearGradientBrush>
    </Border.Background>
    <Border.BorderBrush>
        <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
    </Border.BorderBrush>
    <ContentPresenter x:Name="PART_SelectedContentHost"
Margin="4"
ContentSource="SelectedContent" />
    </Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style TargetType="{x:Type TabItem}">
    <Setter Property="Template">
        <Setter.Value>

```

```

<ControlTemplate TargetType="{x:Type TabItem}">
    <Grid x:Name="Root">
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="SelectionStates">
                <VisualState x:Name="Unselected" />
                <VisualState x:Name="Selected">
                    <Storyboard>
                        <ColorAnimationUsingKeyFrames
                            Storyboard.TargetName="Border"
                            Storyboard.TargetProperty=""
                            (Panel.Background).
                                (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource
ControlPressedColor}" />
                        </ColorAnimationUsingKeyFrames>
                        <ThicknessAnimationUsingKeyFrames
                            Storyboard.TargetProperty="(Border.BorderThickness)">
                            Storyboard.TargetName="Border">
                                <EasingThicknessKeyFrame KeyTime="0"
                                    Value="1,1,1,0" />
                            </ThicknessAnimationUsingKeyFrames>
                        </Storyboard>
                    </VisualState>
                </VisualStateGroup>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="MouseOver" />
                    <VisualState x:Name="Disabled">
                        <Storyboard>
                            <ColorAnimationUsingKeyFrames
                                Storyboard.TargetName="Border"
                                Storyboard.TargetProperty=""
                                (Panel.Background).
                                    (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource
DisabledControlDarkColor}" />
                            </ColorAnimationUsingKeyFrames>
                            <ColorAnimationUsingKeyFrames
                                Storyboard.TargetName="Border"
                                Storyboard.TargetProperty=""
                                (Border.BorderBrush).
                                    (SolidColorBrush.Color)">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource
DisabledBorderLightColor}"/>
                            </ColorAnimationUsingKeyFrames>
                        </Storyboard>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateManager.VisualStateGroups>
            <Border x:Name="Border"
                Margin="0,0,-4,0">

```

```

        BorderThickness="1,1,1,1"
        CornerRadius="2,12,0,0">
    <Border.BorderBrush>
        <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
    </Border.BorderBrush>
    <Border.Background>

        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource ControlLightColor}"
                                  Offset="0.0" />
                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                                  Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>

    </Border.Background>
    <ContentPresenter x:Name="ContentSite"
                      VerticalAlignment="Center"
                      HorizontalAlignment="Center"
                      ContentSource="Header"
                      Margin="12,2,12,2"
                      RecognizesAccessKey="True" />

```

</Border>

</Grid>

<ControlTemplate.Triggers>

<Trigger Property="IsSelected" Value="True">

<Setter Property="Panel.ZIndex" Value="100" />

</Trigger>

</ControlTemplate.Triggers>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>

```

```
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
                     EndPoint="0.5,1"
                     StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
                  Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
                  Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
                  Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
                     StartPoint="0,0"
                     EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
```

```
        Offset="0" />
<GradientStop Color="#600000FF"
               Offset="0.4" />
<GradientStop Color="#600000FF"
               Offset="0.6" />
<GradientStop Color="#000000FF"
               Offset="1" />
</GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

TextBox 样式和模板

项目 • 2023/02/06

本主题介绍 [TextBox 控件](#) 的样式和模板。你可以修改默认的 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

TextBox 部件

下表列出了 [TextBox 控件](#) 的命名部件。

组成部分	类型	描述
PART_ContentHost	FrameworkElement	可包含 FrameworkElement 的可视元素。 TextBox 的文本显示在此元素中。

TextBox 状态

下表列出了 [TextBox 控件](#) 的可视状态。

VisualState 名称 名称	VisualStateGroup	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在控件上方。
已禁用	CommonStates	已禁用控件。
ReadOnly	CommonStates	用户不能更改 TextBox 中的文本。
已设定焦点	FocusStates	控件有焦点。
失去焦点	FocusStates	控件没有焦点。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	如果控件有焦点， Validation.HasError 附加属性为 <code>true</code> 。
InvalidUnfocused	ValidationStates	如果控件没有焦点， Validation.HasError 附加属性为 <code>true</code> 。

TextBox ControlTemplate 示例

下例演示如何定义 TextBox 控件的 ControlTemplate。

XAML

```
<Style TargetType="{x:Type TextBox}">
    <Setter Property="SnapsToDevicePixels"
        Value="True" />
    <Setter Property="OverridesDefaultStyle"
        Value="True" />
    <Setter Property="KeyboardNavigation.TabNavigation"
        Value="None" />
    <Setter Property="FocusVisualStyle"
        Value="{x:Null}" />
    <Setter Property="MinWidth"
        Value="120" />
    <Setter Property="MinHeight"
        Value="20" />
    <Setter Property="AllowDrop"
        Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type TextBoxBase}">
                <Border Name="Border"
                    CornerRadius="2"
                    Padding="2"
                    BorderThickness="1">
                    <Border.Background>
                        <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
                    </Border.Background>
                    <Border.BorderBrush>
                        <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                    </Border.BorderBrush>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="Disabled">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames
                                        Storyboard.TargetName="Border"
                                        Storyboard.TargetProperty=""
                                        (Panel.Background).(
                                            (SolidColorBrush.Color)">
                                            <EasingColorKeyFrame KeyTime="0"
                                                Value="{StaticResource
                                                DisabledControlLightColor}" />
                                            </ColorAnimationUsingKeyFrames>
                                        </Storyboard>
                                    </VisualState>
                                    <VisualState x:Name="ReadOnly">
                                        <Storyboard>
                                            <ColorAnimationUsingKeyFrames
                                                Storyboard.TargetName="Border"
                                                Storyboard.TargetProperty=""
                                                (Panel.Background).(</ColorAnimationUsingKeyFrames>
                                            </Storyboard>
                                        </VisualState>
                                    </VisualStateManager.VisualStateGroups>
                                </Storyboard>
                            </VisualState>
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

```

        (SolidColorBrush.Color)">
        <EasingColorKeyFrame KeyTime="0"
                             Value="{StaticResource
DisabledControlDarkColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="MouseOver" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<ScrollViewer Margin="0"
               x:Name="PART_ContentHost" />
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

```

```
<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)

- 创建控件模板

Thumb 样式和模板

项目 • 2022/09/27

本主题介绍 [Thumb 控件](#) 的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

Thumb 部件

[Thumb 控件](#) 没有任何已命名的部件。

Thumb 状态

下表列出了 [Thumb 控件](#) 的视觉对象状态。

VisualState 名称 名称	VisualStateGroup	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在控件上方。
已按下	CommonStates	已按下控件。
已禁用	CommonStates	已禁用控件。
已设定焦点	FocusStates	控件有焦点。
失去焦点	FocusStates	控件没有焦点。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

Thumb ControlTemplate 示例

下例演示如何定义 [Thumb 控件](#) 的 [ControlTemplate](#)。

XAML

```

<Style x:Key="SliderThumbStyle"
    TargetType="{x:Type Thumb}">
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Height"
        Value="14" />
    <Setter Property="Width"
        Value="14" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Thumb}">
                <Ellipse x:Name="Ellipse"
                    StrokeThickness="1">
                    <Ellipse.Stroke>
                        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStopCollection>
                                    <GradientStop Color="{DynamicResource BorderLightColor}"
                                        Offset="0.0" />
                                    <GradientStop Color="{DynamicResource BorderDarkColor}"
                                        Offset="1.0" />
                                </GradientStopCollection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Ellipse.Stroke>
                    <Ellipse.Fill>
                        <LinearGradientBrush EndPoint="0.5,1"
                            StartPoint="0.5,0">
                            <GradientStop Color="{DynamicResource ControlMediumColor}"
                                Offset="1" />
                            <GradientStop Color="{DynamicResource ControlLightColor}" />
                        </LinearGradientBrush>
                    </Ellipse.Fill>
                <VisualStateManager.VisualStateGroups>
                    <VisualStateGroup x:Name="CommonStates">
                        <VisualState x:Name="Normal" />
                        <VisualState x:Name="MouseOver">
                            <Storyboard>
                                <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="
(Shape.Fill)."
                                    (GradientBrush.GradientStops)[0].(GradientStop.Color)">
                                    <Storyboard.TargetName="Ellipse">
                                        <EasingColorKeyFrame KeyTime="0"
                                            Value="{StaticResource
ControlMouseOverColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="Pressed">
                                <Storyboard>

```

```

        <ColorAnimationUsingKeyFrames Storyboard.TargetProperty=""
(Shape.Fill).
                                (GradientBrush.GradientStops)[0].(GradientStop.Color)">

Storyboard.TargetName="Ellipse">
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource
ControlPressedColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Disabled">
    <Storyboard>
        <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="
(Shape.Fill).
                                (GradientBrush.GradientStops)[0].(GradientStop.Color)">

Storyboard.TargetName="Ellipse">
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource
DisabledControlDarkColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
</Ellipse>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>

```

```
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />

```

</LinearGradientBrush>

```
<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>

```

</LinearGradientBrush>

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

ToggleButton 样式和模板

项目 • 2022/09/27

本主题介绍 [ToggleButton](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

ToggleButton 部件

[ToggleButton](#) 控件没有任何已命名的部件。

ToggleButton 状态

下表列出了 [ToggleButton](#) 控件的视觉对象状态。

VisualState 名称 名称	VisualStateGroup	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在控件上方。
已按下	CommonStates	已按下控件。
已禁用	CommonStates	已禁用控件。
已设定焦点	FocusStates	控件有焦点。
失去焦点	FocusStates	控件没有焦点。
已选中	CheckStates	<code>IsChecked</code> 上声明的默认值为 <code>true</code> 。
未选中	CheckStates	<code>IsChecked</code> 上声明的默认值为 <code>false</code> 。
不确定	CheckStates	<code>IsThreeState</code> 为 <code>true</code> 且 <code>IsChecked</code> 为 <code>null</code> 。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是 <code>true</code> ，并且控件具有焦点。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是 <code>true</code> ，并且控件不具有焦点。

① 备注

如果控件模板中不存在不确定的视觉对象状态，则未选中的可视状态将用作默认视觉对象状态。

ToggleButton ControlTemplate 示例

以下示例演示如何定义 [ToggleButton 控件的 ControlTemplate](#)。

XAML

```
<ControlTemplate x:Key="ComboBoxToggleButton"
                  TargetType="{x:Type ToggleButton}">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
            <ColumnDefinition Width="20" />
        </Grid.ColumnDefinitions>
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CommonStates">
                <VisualState x:Name="Normal" />
                <VisualState x:Name="MouseOver">
                    <Storyboard>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="
                            (Panel.Background)">
                            <GradientBrush.GradientStops><GradientStop.Color>
                                Storyboard.TargetName="Border">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource
ControlMouseOverColor}" />
                            </ColorAnimationUsingKeyFrames>
                        </Storyboard>
                    </VisualState>
                    <VisualState x:Name="Pressed" />
                    <VisualState x:Name="Disabled">
                        <Storyboard>
                            <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="
                            (Panel.Background)">
                                <GradientBrush.GradientStops><GradientStop.Color>
                                    Storyboard.TargetName="Border">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource
DisabledControlDarkColor}" />
                            </ColorAnimationUsingKeyFrames>
                            <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="
                            (Shape.Fill)">
                                <SolidColorBrush.Color>
                                    Storyboard.TargetName="Arrow">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource
DisabledForegroundColor}" />
                            </ColorAnimationUsingKeyFrames>
                            <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="
                            (Border.BorderBrush)">
                                <SolidColorBrush.Color>
                                    Storyboard.TargetName="Arrow">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource
DisabledBorderColor}" />
                            </ColorAnimationUsingKeyFrames>
                        </Storyboard>
                    </VisualState>
                </VisualStateGroup>
            <VisualStateGroup x:Name="FocusStates">
                <VisualState x:Name="Focused" />
                <VisualState x:Name="Unfocused" />
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
        <Image x:Name="Arrow" ...>
    </Grid>
</ControlTemplate>
```

```

        (Border.BorderBrush).

            (GradientBrush.GradientStops)[1].(GradientStop.Color)"
                Storyboard.TargetName="Border">
                <EasingColorKeyFrame KeyTime="0"
                    Value="{StaticResource
DisabledBorderDarkColor}" />
            </ColorAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="CheckStates">
    <VisualState x:Name="Checked">
        <Storyboard>
            <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="
(Panel.Background)."

                (GradientBrush.GradientStops)[1].(GradientStop.Color)"
                    Storyboard.TargetName="Border">
                    <EasingColorKeyFrame KeyTime="0"
                        Value="{StaticResource
ControlPressedColor}" />
                </ColorAnimationUsingKeyFrames>
            </Storyboard>
        </VisualState>
        <VisualState x:Name="Unchecked" />
        <VisualState x:Name="Indeterminate" />
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border x:Name="Border"
    Grid.ColumnSpan="2"
    CornerRadius="2"
    BorderThickness="1">
    <Border.BorderBrush>
        <LinearGradientBrush EndPoint="0,1"
            StartPoint="0,0">
            <GradientStop Color="{DynamicResource BorderLightColor}"
                Offset="0" />
            <GradientStop Color="{DynamicResource BorderDarkColor}"
                Offset="1" />
        </LinearGradientBrush>
    </Border.BorderBrush>
    <Border.Background>

        <LinearGradientBrush StartPoint="0,0"
            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource ControlLightColor}" />
                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                        Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Border.Background>
</Border>

```

```

<Border Grid.Column="0"
        CornerRadius="2,0,0,2"
        Margin="1" >
    <Border.Background>
        <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
    </Border.Background>
</Border>
<Path x:Name="Arrow"
      Grid.Column="1"
      HorizontalAlignment="Center"
      VerticalAlignment="Center"
      Data="M 0 0 L 4 4 L 8 0 Z" >
    <Path.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Fill>
</Path>
</Grid>
</ControlTemplate>

```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>

```

```

<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)

- 样式设置和模板化
- 创建控件模板

ToolBar 样式和模板

项目 • 2023/02/06

本主题介绍 [ToolBar](#) 控件的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

ToolBar 部件

下表列出了 [ToolBar](#) 控件的命名部件。

组成部分	类型	描述
PART_ToolBarPanel	ToolBarPanel	包含 ToolBar 上的控件的对象。
PART_ToolBarOverflowPanel	ToolBarOverflowPanel	包含 ToolBar 的溢出区域中的控件的对象。

为 [ToolBar](#) 创建一个 [ControlTemplate](#) 时，你的模板可能在 [ScrollViewer](#) 中包含一个 [ItemsPresenter](#)。（ [ItemsPresenter](#) 显示 [ToolBar](#) 中的每个项；[ScrollViewer](#) 支持在控件内滚动）。如果 [ItemsPresenter](#) 不是 [ScrollViewer](#) 的直接子元素，则必须为 [ItemsPresenter](#) 指定名称 `ItemsPresenter`。

ToolBar 状态

下表列出了 [ToolBar](#) 控件的视觉对象状态。

VisualState 名称	VisualStateGroup 名称	描述
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

ToolBar ControlTemplate 示例

以下示例演示如何定义 [ToolBar](#) 控件的 [ControlTemplate](#)。

XAML

```

<Style x:Key="ToolBarButtonBaseStyle"
    TargetType="{x:Type ButtonBase}">
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type ButtonBase}">
                <Border x:Name="Border"
                    BorderThickness="1">
                    <Border.BorderBrush>
                        <LinearGradientBrush EndPoint="0.5,1"
                            StartPoint="0.5,0">
                            <GradientStop Color="{DynamicResource BorderLightColor}"
                                Offset="0" />
                            <GradientStop Color="{DynamicResource BorderMediumColor}"
                                Offset="1" />
                        </LinearGradientBrush>
                    </Border.BorderBrush>
                    <Border.Background>
                        <LinearGradientBrush EndPoint="0.5,1"
                            StartPoint="0.5,0">
                            <GradientStop Color="{DynamicResource ControlLightColor}"
                                Offset="0" />
                            <GradientStop Color="{DynamicResource ControlMediumColor}"
                                Offset="1" />
                        </LinearGradientBrush>
                    </Border.Background>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="Pressed">
                                <Storyboard
                                    <ColorAnimationUsingKeyFrames
                                        Storyboard.TargetName="Border"
                                        Storyboard.TargetProperty=""
                                        (Panel.Background).(GradientBrush.GradientStops)[1].(GradientStop.Color)">
                                    <EasingColorKeyFrame KeyTime="0"
                                        Value="{StaticResource
ControlPressedColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="MouseOver">
                                <Storyboard
                                    <ColorAnimationUsingKeyFrames
                                        Storyboard.TargetName="Border"
                                        Storyboard.TargetProperty=""
                                        (Panel.Background).(GradientBrush.GradientStops)[1].(GradientStop.Color)">
                                    <EasingColorKeyFrame KeyTime="0"
                                        Value="{StaticResource
ControlMouseOverColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```

        ControlMouseOverColor}" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
<VisualState x:Name="Disabled">
    <Storyboard>
        <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Border"
Storyboard.TargetProperty="

(Panel.Background).
        (GradientBrush.GradientStops)[1].(GradientStop.Color)">
        <EasingColorKeyFrame KeyTime="0"
            Value="{StaticResource
DisabledControlDarkColor}" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="CheckStates">
    <VisualState x:Name="Checked">
        <Storyboard>
            <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Border"
Storyboard.TargetProperty="

(Panel.Background).
        (GradientBrush.GradientStops)[1].(GradientStop.Color)">
        <EasingColorKeyFrame KeyTime="0"
            Value="{StaticResource
ControlPressedColor}" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
<VisualState x:Name="Unchecked" />
<VisualState x:Name="Indeterminate" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<ContentPresenter Margin="2"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"
    RecognizesAccessKey="True" />
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="{x:StaticToolBar.ButtonStyleKey}"
    BasedOn="{StaticResourceToolBarButtonBaseStyle}"
    TargetType="{x:Type Button}" />
<Style x:Key="{x:StaticToolBar.ToggleButtonStyleKey}"
    BasedOn="{StaticResourceToolBarButtonBaseStyle}"
    TargetType="{x:Type ToggleButton}" />
<Style x:Key="{x:StaticToolBar.CheckBoxStyleKey}"
    BasedOn="{StaticResourceToolBarButtonBaseStyle}"
    TargetType="{x:Type CheckBox}" />

```

```

<Style x:Key="{x:StaticToolBar.RadioButtonStyleKey}"
       BasedOn="{StaticResourceToolBarButtonBaseStyle}"
       TargetType="{x:Type RadioButton}"/>

<Style x:Key="{x:StaticToolBar.TextBoxStyleKey}"
       TargetType="{x:Type TextBox}>
    <Setter Property="SnapsToDevicePixels"
           Value="True" />
    <Setter Property="OverridesDefaultStyle"
           Value="True" />
    <Setter Property="KeyboardNavigation.TabNavigation"
           Value="None" />
    <Setter Property="FocusVisualStyle"
           Value="{x:Null}" />
    <Setter Property="AllowDrop"
           Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type TextBox}>
                <Border x:Name="Border"
                        Padding="2"
                        BorderThickness="1">
                    <Border.Background>
                        <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
                    </Border.Background>
                    <Border.BorderBrush>
                        <SolidColorBrush Color="{StaticResource BorderMediumColor}" />
                    </Border.BorderBrush>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="Disabled">
                                <Storyboard
                                    <ColorAnimationUsingKeyFrames
                                        Storyboard.TargetName="Border"
                                        Storyboard.TargetProperty=""
                                        (Panel.Background).(SolidColorBrush.Color)">
                                    <EasingColorKeyFrame KeyTime="0"
                                            Value="{StaticResource
                                                DisabledControlDarkColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                    <ColorAnimationUsingKeyFrames
                                        Storyboard.TargetName="Border"
                                        Storyboard.TargetProperty=""
                                        (Border.BorderBrush).(SolidColorBrush.Color)">
                                    <EasingColorKeyFrame KeyTime="0"
                                            Value="{StaticResource
                                                DisabledBorderLightColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="ReadOnly" />
                            <VisualState x:Name="MouseOver" />
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```

        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
<ScrollViewer Margin="0"
              x:Name="PART_ContentHost" />
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="ToolBarThumbStyle"
       TargetType="{x:Type Thumb}">
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="Cursor"
           Value="SizeAll" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Thumb}">
                <Border Background="Transparent"
                       SnapsToDevicePixels="True">
                    <Rectangle Margin="0,2">
                        <Rectangle.Fill>
                            <DrawingBrush Viewport="0,0,4,4"
                                          ViewportUnits="Absolute"
                                          Viewbox="0,0,8,8"
                                          ViewboxUnits="Absolute"
                                          TileMode="Tile">
                                <DrawingBrush.Drawing>
                                    <DrawingGroup>
                                        <GeometryDrawing Brush="#AAA"
                                                          Geometry="M 4 4 L 4 8 L 8 8 L 8 4 z" />
                                    </DrawingGroup>
                                </DrawingBrush.Drawing>
                            </DrawingBrush>
                        </Rectangle.Fill>
                    </Rectangle>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<Style x:Key="ToolBarOverflowButtonStyle"
       TargetType="{x:Type ToggleButton}">
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type ToggleButton}">
                <Border x:Name="Border"
                       CornerRadius="0,3,3,0"
                       SnapsToDevicePixels="true">
                    <Border.Background>
                        <LinearGradientBrush EndPoint="0.5,1"

```

```

                StartPoint="0.5,0">
            <GradientStop Color="#00000000"
                           Offset="0" />
            <GradientStop Offset="1" />
        </LinearGradientBrush>
    </Border.Background>
    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="CommonStates">
            <VisualState x:Name="Normal" />
            <VisualState x:Name="Pressed">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Border"
Storyboard.TargetProperty="

(Panel.Background).(GradientBrush.GradientStops)[1].(GradientStop.Color)">
<EasingColorKeyFrame KeyTime="0"
Value="{StaticResource

ControlPressedColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
            <VisualState x:Name="MouseOver">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Border"
Storyboard.TargetProperty="

(Panel.Background).(GradientBrush.GradientStops)[1].(GradientStop.Color)">
<EasingColorKeyFrame KeyTime="0"
Value="{StaticResource

ControlMouseOverColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
            <VisualState x:Name="Disabled">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames
Storyboard.TargetName="Border"
Storyboard.TargetProperty="

(Panel.Background).(GradientBrush.GradientStops)[1].(GradientStop.Color)">
<EasingColorKeyFrame KeyTime="0"
Value="{StaticResource

DisabledBorderLightColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
    <Grid>
        <Path x:Name="Arrow"
              Fill="Black"
              VerticalAlignment="Bottom"
              Margin="2,3"

```

```

        Data="M -0.5 3 L 5.5 3 L 2.5 6 Z" />
    <ContentPresenter />
</Grid>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="{x:TypeToolBar}"
    TargetType="{x:TypeToolBar}">
<Setter Property="SnapsToDevicePixels"
    Value="true" />
<Setter Property="OverridesDefaultStyle"
    Value="true" />
<Setter Property="Template">
<Setter.Value>
    <ControlTemplate TargetType="{x:TypeToolBar}">
        <Border x:Name="Border"
            CornerRadius="2"
            BorderThickness="1">
            <Border.BorderBrush>
                <LinearGradientBrush StartPoint="0,0"
                    EndPoint="0,1">
                    <LinearGradientBrush.GradientStops>
                        <GradientStopCollection>
                            <GradientStop Color="{DynamicResource BorderMediumColor}"
                                Offset="0.0" />
                            <GradientStop Color="{DynamicResource BorderDarkColor}"
                                Offset="1.0" />
                        </GradientStopCollection>
                    </LinearGradientBrush.GradientStops>
                </LinearGradientBrush>
            </Border.BorderBrush>
        </Border.Background>
        <LinearGradientBrush StartPoint="0,0"
            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="#FFF"
                        Offset="0.0" />
                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                        Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Border.Background>
<DockPanel>
    <ToggleButton DockPanel.Dock="Right"
        IsEnabled="{TemplateBinding HasOverflowItems}"
        Style="{StaticResource
ToolBarOverflowButtonStyle}"
        ClickMode="Press"
        IsChecked="{Binding IsOverflowOpen, Mode=TwoWay,

```

```

        RelativeSource={RelativeSource TemplatedParent}}}">
    <Popup x:Name="OverflowPopup"
        AllowsTransparency="true"
        Placement="Bottom"
        StaysOpen="false"
        Focusable="false"
        PopupAnimation="Slide"
        IsOpen="{Binding IsOverflowOpen,
RelativeSource={RelativeSource TemplatedParent}}">
        <Border x:Name="DropDownBorder"
            BorderThickness="1">
            <Border.BorderBrush>
                <SolidColorBrush Color="{DynamicResource
BorderMediumColor}" />
            </Border.BorderBrush>
            <Border.Background>
                <LinearGradientBrush EndPoint="0.5,1"
                    StartPoint="0.5,0">
                    <GradientStop Color="{DynamicResource
ControlLightColor}" />
                    <GradientStop Color="{DynamicResource
ControlMediumColor}" /
                        Offset="1" />
                </LinearGradientBrush>
            </Border.Background>
            <ToolBarOverflowPanel x:Name="PART_ToolBarOverflowPanel"
                Margin="2"
                WrapWidth="200"
                Focusable="true"
                FocusVisualStyle="{x:Null}">
                KeyboardNavigation.TabNavigation="Cycle"

                KeyboardNavigation.DirectionNavigation="Cycle" />
            </Border>
        </Popup>
    </ToggleButton>

    <Thumb x:Name="ToolBarThumb"
        Style="{StaticResource ToolBarThumbStyle}"
        Width="10" />
    <ToolBarPanel x:Name="PART_ToolBarPanel"
        IsItemsHost="true"
        Margin="0,1,2,2" />
    </DockPanel>
</Border>
<ControlTemplate.Triggers>
    <Trigger Property="IsOverflowOpen"
        Value="true">
        <Setter TargetName="ToolBarThumb"
            Property="IsEnabled"
            Value="false" />
    </Trigger>
    <Trigger Property="ToolBarTray.IsLocked"
        Value="true">

```

```

        <Setter TargetName="ToolBarThumb"
            Property="Visibility"
            Value="Collapsed" />
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="{x:TypeToolBarTray}"
    TargetType="{x:TypeToolBarTray}">
<Setter Property="Background">
<Setter.Value>
    <LinearGradientBrush StartPoint="0,0"
        EndPoint="1,0">
        <LinearGradientBrush.GradientStops>
            <GradientStopCollection>
                <GradientStop Color="#FFF"
                    Offset="0.0" />
                <GradientStop Color="{DynamicResource WindowColor}"
                    Offset="1.0" />
            </GradientStopCollection>
        </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

```

```
<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
                     EndPoint="0.5,1"
                     StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
                  Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
                  Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
                  Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
                     StartPoint="0,0"
                     EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                          Offset="0" />
            <GradientStop Color="#600000FF"
                          Offset="0.4" />
            <GradientStop Color="#600000FF"
                          Offset="0.6" />
            <GradientStop Color="#000000FF"
                          Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

ToolTip 样式和模板

项目 • 2023/02/06

本主题介绍 [ToolTip 控件](#) 的样式和模板。可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。有关详细信息，请参阅[为控件创建模板](#)。

ToolTip 部件

[ToolTip](#) 控件没有任何命名部件。

ToolTip 状态

下表列出了 [ToolTip](#) 控件的可视状态。

VisualState 名称 名称	VisualStateGroup	描述
已关闭	OpenStates	默认状态。
打开	OpenStates	ToolTip 可见。
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	如果 Validation.HasError 附加属性为 <code>true</code> ，该控件没有焦点。

ToolTip ControlTemplate 示例

以下示例演示如何定义 [ToolTip](#) 控件的 [ControlTemplate](#)。

XAML

```
<Style x:Key="{x:Type ToolTip}"  
       TargetType="ToolTip">  
    <Setter Property="OverridesDefaultStyle"  
           Value="true" />  
    <Setter Property="HasDropShadow"  
           Value="True" />  
    <Setter Property="Template">  
        <Setter.Value>
```

```

<ControlTemplate TargetType="ToolTip">
    <Border Name="Border"
        BorderThickness="1"
        Width="{TemplateBinding Width}"
        Height="{TemplateBinding Height}">
        <Border.Background>
            <LinearGradientBrush StartPoint="0,0"
                EndPoint="0,1">
                <LinearGradientBrush.GradientStops>
                    <GradientStopCollection>
                        <GradientStop Color="{DynamicResource ControlLightColor}"
                            Offset="0.0" />
                        <GradientStop Color="{DynamicResource ControlMediumColor}"
                            Offset="1.0" />
                    </GradientStopCollection>
                </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
        </Border.Background>
        <Border.BorderBrush>
            <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
        </Border.BorderBrush>
        <ContentPresenter Margin="4"
            HorizontalAlignment="Left"
            VerticalAlignment="Top" />
    </Border>
    <ControlTemplate.Triggers>
        <Trigger Property="HasDropShadow"
            Value="true">
            <Setter TargetName="Border"
                Property="CornerRadius"
                Value="4" />
            <Setter TargetName="Border"
                Property="SnapsToDevicePixels"
                Value="true" />
        </Trigger>
    </ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>

```

```
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />

```

```
<GradientStop Color="#600000FF"
               Offset="0.4" />
<GradientStop Color="#600000FF"
               Offset="0.6" />
<GradientStop Color="#000000FF"
               Offset="1" />
</GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

TreeView 样式和模板

项目 • 2023/02/06

本主题介绍 [TreeView](#) 控件的样式和模板。 可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。 有关详细信息，请参阅[为控件创建模板](#)。

TreeView 部件

[TreeView](#) 控件没有任何已命名的部件。

为 [TreeView](#) 创建 [ControlTemplate](#) 时，模板的 [ScrollViewer](#) 中可能包含 [ItemsPresenter](#)。（[ItemsPresenter](#) 显示 [TreeView](#) 中的每个项；[ScrollViewer](#) 支持在控件内滚动）。如果 [ItemsPresenter](#) 不是 [ScrollViewer](#) 的直接子级，则必须将 [ItemsPresenter](#) 命名为 `ItemsPresenter`。

TreeView 状态

下表列出了 [TreeView](#) 控件的视觉对象状态。

VisualStyle 名称 名称	VisualStyleGroup 名称	描述
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	如果控件不具有焦点，则 Validation.HasError 附加属性为 <code>true</code> 。

TreeViewItem 部件

下表列出了 [TreeViewItem](#) 控件的已命名部件。

组成部分	类型	描述
PART_Header	FrameworkElement	包含 TreeView 控件的标头内容的视觉元素。

TreeViewItem 状态

下表列出了 TreeViewItem 控件的视觉对象状态。

VisualState 名称 名称	VisualStateGroup	描述
普通	CommonStates	默认状态。
MouseOver	CommonStates	鼠标指针悬停在 TreeViewItem 上方。
已禁用	CommonStates	禁用 TreeViewItem。
已设定焦点	FocusStates	TreeViewItem 具有焦点。
失去焦点	FocusStates	TreeViewItem 没有焦点。
展开	ExpansionStates	TreeViewItem 控件已展开。
Collapsed	ExpansionStates	TreeViewItem 控件已折叠。
HasItems	HasItemsStates	TreeViewItem 具有项。
NoItems	HasItemsStates	TreeViewItem 没有项。
选定	SelectionStates	TreeViewItem 已选定。
SelectedInactive	SelectionStates	TreeViewItem 处于选中状态，但未处于活动状态。
未选定	SelectionStates	TreeViewItem 未选定。
有效	ValidationStates	该控件使用 Validation 类，Validation.HasError 附加属性为 false。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 true。
InvalidUnfocused	ValidationStates	如果控件不具有焦点，则 Validation.HasError 附加属性为 true。

TreeView ControlTemplate 示例

下例演示如何定义 TreeView 控件的 ControlTemplate 及其关联类型。

XAML

```
<Style x:Key="{x:Type TreeView}"  
       TargetType="TreeView">  
    <Setter Property="OverridesDefaultStyle"  
           Value="True" />  
    <Setter Property="SnapsToDevicePixels"  
           Value="True" />  
    <Setter Property="ScrollViewer.HorizontalScrollBarVisibility"
```

```
    Value="Auto" />
<Setter Property="ScrollViewer.VerticalScrollBarVisibility"
    Value="Auto" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="TreeView">
            <Border Name="Border"
                CornerRadius="1"
                BorderThickness="1">
                <Border.BorderBrush>
                    <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                </Border.BorderBrush>
                <Border.Background>
                    <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
                </Border.Background>
                <ScrollViewer Focusable="False"
                    CanContentScroll="False"
                    Padding="4">
                    <ItemsPresenter />
                </ScrollViewer>
            </Border>
        </ControlTemplate>
    </Setter.Value>
</Setter>
</Style>

<Style x:Key="ExpandCollapseToggleStyle"
    TargetType="ToggleButton">
    <Setter Property="Focusable"
        Value="False" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="ToggleButton">
                <Grid Width="15"
                    Height="13"
                    Background="Transparent">
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CheckStates">
                            <VisualState x:Name="Checked">
                                <Storyboard>
                                    <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="
(UIElement.Visibility)">
                                        Storyboard.TargetName="Collapsed">
                                            <DiscreteObjectKeyFrame KeyTime="0"
                                                Value="{x:Static
Visibility.Hidden}" />
                                            <ObjectAnimationUsingKeyFrames>
                                                <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="
(UIElement.Visibility)">
                                                    Storyboard.TargetName="Expanded">
                                                        <DiscreteObjectKeyFrame KeyTime="0"
                                                            Value="{x:Static
Visibility.Visible}" />

```

```
        </ObjectAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
<VisualState x:Name="Unchecked" />
<VisualState x:Name="Indeterminate" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Path x:Name="Collapsed"
    HorizontalAlignment="Left"
    VerticalAlignment="Center"
    Margin="1,1,1,1"
    Data="M 4 0 L 8 4 L 4 8 Z">
    <Path.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Fill>
</Path>
<Path x:Name="Expanded"
    HorizontalAlignment="Left"
    VerticalAlignment="Center"
    Margin="1,1,1,1"
    Data="M 0 4 L 8 4 L 4 8 Z"
    Visibility="Hidden">
    <Path.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Fill>
</Path>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
<Style x:Key="TreeViewItemFocusVisual">
    <Setter Property="Control.Template">
        <Setter.Value>
            <ControlTemplate>
                <Border>
                    <Rectangle Margin="0,0,0,0"
                        StrokeThickness="5"
                        Stroke="Black"
                        StrokeDashArray="1 2"
                        Opacity="0" />
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
<Style x:Key="{x:Type TreeViewItem}"
    TargetType="{x:Type TreeViewItem}">
    <Setter Property="Background"
        Value="Transparent" />
    <Setter Property="HorizontalContentAlignment"
        Value="{Binding Path=HorizontalContentAlignment,
            RelativeSource={RelativeSource AncestorType={x:Type ItemsControl}}}" />
    <Setter Property="VerticalContentAlignment"
        Value="{Binding Path=VerticalContentAlignment,>
```

```

        RelativeSource={RelativeSource AncestorType={x:Type ItemsControl}}}" />
<Setter Property="Padding"
        Value="1,0,0,0" />
<Setter Property="Foreground"
        Value="{DynamicResource {x:Static
SystemColors.ControlTextBrushKey}}" />
<Setter Property="FocusVisualStyle"
        Value="{StaticResource TreeViewItemFocusVisual}" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type TreeViewItem}">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition MinWidth="19"
                                      Width="Auto" />
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="*" />
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto" />
                    <RowDefinition />
                </Grid.RowDefinitions>
                <VisualStateManager.VisualStateGroups>
                    <VisualStateGroup x:Name="SelectionStates">
                        <VisualState x:Name="Selected">
                            <Storyboard>
                                <ColorAnimationUsingKeyFrames Storyboard.TargetName="Bd"
                                                               Storyboard.TargetProperty=""
(Panel.Background).
(SolidColorBrush.Color)">
                                <EasingColorKeyFrame KeyTime="0"
                                         Value="{StaticResource
SelectedBackgroundColor}" />
                                </ColorAnimationUsingKeyFrames>
                            </Storyboard>
                        </VisualState>
                        <VisualState x:Name="Unselected" />
                        <VisualState x:Name="SelectedInactive">
                            <Storyboard>
                                <ColorAnimationUsingKeyFrames Storyboard.TargetName="Bd"
                                                               Storyboard.TargetProperty=""
(Panel.Background).
(SolidColorBrush.Color)">
                                <EasingColorKeyFrame KeyTime="0"
                                         Value="{StaticResource
SelectedUnfocusedColor}" />
                                </ColorAnimationUsingKeyFrames>
                            </Storyboard>
                        </VisualState>
                    </VisualStateGroup>
                    <VisualStateGroup x:Name="ExpansionStates">
                        <VisualState x:Name="Expanded">
                            <Storyboard>
                                <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty=""

```

```
(UIElement.Visibility)"
```

```
Storyboard.TargetName="ItemsHost">
    <DiscreteObjectKeyFrame KeyTime="0"
        Value="{x:Static Visibility.Visible}" />
    </ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Collapsed" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<ToggleButton x:Name="Expander"
    Style="{StaticResource ExpandCollapseToggleStyle}"
    ClickMode="Press"
    IsChecked="{Binding IsExpanded,
        RelativeSource={RelativeSource TemplatedParent}}"/>
<Border x:Name="Bd"
    Grid.Column="1"
    Background="{TemplateBinding Background}"
    BorderBrush="{TemplateBinding BorderBrush}"
    BorderThickness="{TemplateBinding BorderThickness}"
    Padding="{TemplateBinding Padding}">
    <ContentPresenter x:Name="PART_Header"
        ContentSource="Header"
        HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}"/>
</Border>
<ItemsPresenter x:Name="ItemsHost"
    Grid.Row="1"
    Grid.Column="1"
    Grid.ColumnSpan="2"
    Visibility="Collapsed" />
</Grid>
<ControlTemplate.Triggers>
    <Trigger Property="HasItems"
        Value="false">
        <Setter TargetName="Expander"
            Property="Visibility"
            Value="Hidden" />
    </Trigger>
    <MultiTrigger>
        <MultiTrigger.Conditions>
            <Condition Property="HasHeader"
                Value="false" />
            <Condition Property="Width"
                Value="Auto" />
        </MultiTrigger.Conditions>
        <Setter TargetName="PART_Header"
            Property="MinWidth"
            Value="75" />
    </MultiTrigger>
    <MultiTrigger>
        <MultiTrigger.Conditions>
            <Condition Property="HasHeader"
```

```

        Value="false" />
    <Condition Property="Height"
        Value="Auto" />
</MultiTrigger.Conditions>
<Setter TargetName="PART_Header"
    Property="MinHeight"
    Value="19" />
</MultiTrigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

上一示例使用了一个或多个以下资源。

XAML

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

```

```
<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

Window 样式和模板

项目 • 2023/02/06

本主题介绍 [Window](#) 控件的样式和模板。 可以修改默认 [ControlTemplate](#) 以赋予控件独特的外观。 有关详细信息，请参阅[为控件创建模板](#)。

Window 部件

[Window](#) 控件没有任何已命名的部件。

Window 状态

下表列出了 [Window](#) 控件的视觉对象状态。

VisualState 名称 名称	VisualStateGroup	描述
有效	ValidationStates	该控件使用 Validation 类， Validation.HasError 附加属性为 <code>false</code> 。
InvalidFocused	ValidationStates	Validation.HasError 附加属性是让控件具有焦点的 <code>true</code> 。
InvalidUnfocused	ValidationStates	Validation.HasError 附加属性是让控件不具有焦点的 <code>true</code> 。

Window ControlTemplate 示例

下例演示如何定义 [Window](#) 控件的 [ControlTemplate](#)。

XAML

```
<Style x:Key="{x:Type Window}"  
       TargetType="{x:Type Window}">  
    <Setter Property="SnapsToDevicePixels"  
           Value="true" />  
    <Setter Property="Template">  
        <Setter.Value>  
            <ControlTemplate TargetType="{x:Type Window}">  
                <Grid>  
                    <Grid.Background>  
                        <SolidColorBrush Color="{DynamicResource WindowColor}" />  
                    </Grid.Background>  
                    <AdornerDecorator>
```

```
<ContentPresenter />
</AdornerDecorator>
<ResizeGrip x:Name="WindowResizeGrip"
            HorizontalAlignment="Right"
            VerticalAlignment="Bottom"
            Visibility="Collapsed"
            IsTabStop="false" />
</Grid>
<ControlTemplate.Triggers>
    <Trigger Property="ResizeMode"
              Value="CanResizeWithGrip">
        <Setter TargetName="WindowResizeGrip"
                Property="Visibility"
                Value="Visible" />
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
```

ControlTemplate 使用了一个或多个以下资源。

XAML

```
<Style x:Key="{x:Type ResizeGrip}"
       TargetType="{x:Type ResizeGrip}">
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type ResizeGrip}">
                <Border Background="Transparent"
                        SnapsToDevicePixels="True"
                        Width="16"
                        Height="16">
                    <Rectangle Margin="2">
                        <Rectangle.Fill>
                            <DrawingBrush Viewport="0,0,4,4"
                                          ViewportUnits="Absolute"
                                          Viewbox="0,0,8,8"
                                          ViewboxUnits="Absolute"
                                          TileMode="Tile">
                                <DrawingBrush.Drawing>
                                    <DrawingGroup>
                                        <DrawingGroup.Children>
                                            <GeometryDrawing Brush="#FFE8EDF9"
                                                               Geometry="M 4 4 L 4 8 L
                                                               8 8 L 8 4 z" />
                                        </DrawingGroup.Children>
                                    </DrawingGroup>
                                </DrawingBrush.Drawing>
                            </DrawingBrush>
                        </Rectangle.Fill>
                    </Rectangle>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

```
        </Rectangle.Fill>
    </Rectangle>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
```

XAML

```
<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215,
0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>
```

```
<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

有关完整示例，请参阅[使用 ControlTemplates 设置样式示例](#)。

另请参阅

- [Style](#)
- [ControlTemplate](#)
- [控件样式和模板](#)
- [控件自定义](#)
- [样式设置和模板化](#)
- [创建控件模板](#)

WPF 自定义控件的 UI 自动化

项目 • 2023/02/06

UI 自动化使质量保证（测试）代码和具有辅助功能的应用程序（如屏幕阅读器）能够检查用户界面元素，以及能够模拟与其他代码中的用户元素进行的用户交互。有关跨所有平台的 UI 自动化的信息，请参阅“[辅助功能](#)”。

本主题介绍如何实现在 WPF 应用程序中运行的自定义控件的服务器端 UI 自动化提供程序。WPF 通过等同于用户界面元素树的对等自动化对象树来支持 UI 自动化。测试代码和提供辅助功能的应用程序可以直接使用自动化对等对象（适用于进程内代码），也可以通过 UI 自动化提供的通用接口使用自动化对等对象。

自动化对等类

WPF 控件通过派生自 [AutomationPeer](#) 的对等类树支持 UI 自动化。按照约定，对等类名以控件类名开头，以“AutomationPeer”结尾。例如，[ButtonAutomationPeer](#) 是 [Button](#) 控制类的对等类。对等类大致相当于 UI 自动化控件类型，但特定于 WPF 元素。通过 UI 自动化接口访问 WPF 应用程序的自动化代码不直接使用自动化对等，但同一进程空间中的自动化代码可以直接使用自动化对等。

内置的自动化对等类

如果元素接受用户的接口活动，或者如果元素包含屏幕阅读器应用程序的用户所需的信息，则这些元素会实现一个自动化对等类。并非所有 WPF 视觉元素都具有自动化对等。实现自动化对等的类的示例包括 [Button](#)、[TextBox](#) 和 [Label](#)。不实现自动化对等的类的示例是从 [Decorator](#) 派生的类（例如 [Border](#)）以及基于 [Panel](#) 的类（例如 [Grid](#) 和 [Canvas](#)）。

基本 [Control](#) 类没有相应的对等类。如果需要一个对等类来对应从 [Control](#) 派生的自定义控件，应从 [FrameworkElementAutomationPeer](#) 派生自定义对等类。

派生对等的安全注意事项

自动化对等必须在部分信任环境中运行。UIAutomationClient 程序集中的代码未配置为在部分信任环境中运行，自动化对等代码不应引用该程序集。应改用 UIAutomationTypes 程序集中的类。例如，应使用 UIAutomationTypes 程序集中的 [AutomationElementIdentifiers](#) 类，该类对应于 UIAutomationClient 程序集中的 [AutomationElement](#) 类。可以安全地在自动化对等代码中引用 UIAutomationTypes 程序集。

对等导航

定位自动化对等点后，进程内代码可通过调用对象的 [GetChildren](#) 和 [GetParent](#) 方法导航对等点树。 可通过在对等中实现 [GetChildrenCore](#) 方法来支持在控件中的各个 WPF 元素之间进行导航。 UI 自动化系统调用此方法生成控件内包含的子元素树（例如，列表框中的列表项）。 默认的 [UIElementAutomationPeer.GetChildrenCore](#) 方法遍历元素的可视化树以构建自动化对等树。 自定义控件重写此方法以向自动化客户端公开子元素，这会返回传达信息或允许用户交互的元素自动化对等。

派生对等中的自定义项

从 [UIElement](#) 和 [ContentElement](#) 派生的所有类都包含受保护的虚拟方法 [OnCreateAutomationPeer](#)。 WPF 调用 [OnCreateAutomationPeer](#) 以获取每个控件的自动化对等对象。 自动化代码可使用对等来获取有关控件特征和功能的信息，并模拟交互使用。 支持自动化的自定义控件必须替代 [OnCreateAutomationPeer](#) 并返回从 [AutomationPeer](#) 派生的类的实例。 例如，如果自定义控件派生自 [ButtonBase](#) 类，则 [OnCreateAutomationPeer](#) 返回的对象应派生自 [ButtonBaseAutomationPeer](#)。

实现自定义控件时，必须重写自动化对等基类中的“Core”方法，这些方法描述了特定于自定义控件的唯一行为。

重写 OnCreateAutomationPeer

替代自定义控件的 [OnCreateAutomationPeer](#) 方法，使其返回提供程序对象，该对象必须直接或间接从 [AutomationPeer](#) 派生。

重写 GetPattern

自动化对等简化了服务器端 UI 自动化提供程序的一部分实现，但自定义控件自动化对等仍必须处理模式接口。 与非 WPF 提供程序一样，对等点通过在 [System.Windows.Automation.Provider](#) 命名空间（例如 [IInvokeProvider](#)）中提供接口实现来支持控制模式。 控件模式接口可以由对等本身或其他对象实现。 对等的 [GetPattern](#) 实现返回支持指定模式的对象。 UI 自动化代码调用 [GetPattern](#) 方法并指定 [PatternInterface](#) 枚举值。 你替代的 [GetPattern](#) 应当返回一个实现特定模式的对象。 如果控件没有模式的自定义实现，可以调用基类型的 [GetPattern](#) 实现来检索其实现或返回 null（如果此控件类型不支持该模式）。 例如，[NumericUpDown](#) 自定义控件可设置为某个范围内的值，以便该控件的 UI 自动化对等实现 [IRangeValueProvider](#) 接口。 下面的示例演示了如何替代对等的 [GetPattern](#) 方法以响应 [PatternInterface.RangeValue](#) 值。

C#

```
public override object GetPattern(PatternInterface patternInterface)
{
    if (patternInterface == PatternInterface.RangeValue)
    {
        return this;
    }
    return base.GetPattern(patternInterface);
}
```

`GetPattern` 方法还可以将子元素指定为模式提供程序。下面的代码显示 `ItemsControl` 如何将滚动模式处理转移到其内部 `ScrollViewer` 控件的对等。

C#

```
public override object GetPattern(PatternInterface patternInterface)
{
    if (patternInterface == PatternInterface.Scroll)
    {
        ItemsControl owner = (ItemsControl) base.Owner;

        // ScrollHost is internal to the ItemsControl class
        if (owner.ScrollHost != null)
        {
            AutomationPeer peer =
UIElementAutomationPeer.CreatePeerForElement(owner.ScrollHost);
            if ((peer != null) && (peer is IScrollProvider))
            {
                peer.EventsSource = this;
                return (IScrollProvider) peer;
            }
        }
    }
    return base.GetPattern(patternInterface);
}
```

为指定用于模式处理的子元素，此代码会获取子元素对象，使用 `CreatePeerForElement` 方法创建对等点，将新对等点的 `EventsSource` 属性设置为当前对等点，并返回新对等点。对子元素设置 `EventsSource` 可防止子元素出现在自动化对等树中，并将子元素引发的所有事件指定为源自 `EventsSource` 中指定的控件。`ScrollViewer` 控件不会出现在自动化树中，并且它生成的滚动事件似乎源自 `ItemsControl` 对象。

重写“Core”方法

自动化代码通过调用对等类的公共方法来获取控件的相关信息。在控件实现不同于自动化对等基类提供的实现的情况下，若要提供控件的相关信息，请重写名称以“Core”结尾的每个方法。控件至少必须实现 `GetClassNameCore` 和 `GetAutomationControlTypeCore` 方法，如以下示例所示。

C#

```
protected override string GetClassNameCore()
{
    return "NumericUpDown";
}

protected override AutomationControlType GetAutomationControlTypeCore()
{
    return AutomationControlType.Spinner;
}
```

`GetAutomationControlTypeCore` 的实现通过返回 `ControlType` 值来描述你的控件。 尽管你可以返回 `ControlType.Custom`，但如果它准确地描述了你的控件，你应返回更具体的控件类型之一。`ControlType.Custom` 的返回值要求提供程序额外实现 UI 自动化，UI 自动化客户端产品预见不到控件结构、键盘交互和可能的控件模式。

实现 `IsContentElementCore` 和 `IsControlElementCore` 方法以指示你的控件是否包含数据内容或者/而且是否满足用户界面中的交互角色。 默认情况下，这两个方法返回 `true`。 这些设置提高了屏幕阅读器等自动化工具的可用性，此类工具可使用这些方法筛选自动化树。 如果 `GetPattern` 方法将模式处理转移到子元素对等点，则子元素对等点的 `IsControlElementCore` 方法可以返回 `false` 以从自动化树中隐藏子元素对等点。 例如，`ListBox` 中的滚动由 `ScrollViewer` 处理，`PatternInterface.Scroll` 的自动化对等点由与 `ListBoxAutomationPeer` 关联的 `ScrollViewerAutomationPeer` 的 `GetPattern` 方法返回。 因此，`ScrollViewerAutomationPeer` 的 `IsControlElementCore` 方法会返回 `false`，使 `ScrollViewerAutomationPeer` 不出现在自动化树中。

自动化对等应为控件提供合适的默认值。 请注意，引用你的控件的 XAML 可以通过包括 `AutomationProperties` 属性来替代 Core 方法的对等实现。 例如，下面的 XAML 创建一个按钮，该按钮具有两个自定义的 UI 自动化属性。

XAML

```
<Button AutomationProperties.Name="Special"
        AutomationProperties.HelpText="This is a special button." />
```

实现模式提供程序

如果所属的元素直接派生自 `Control`，会显式声明自定义提供程序实现的接口。 例如，以下代码为实现一个范围值的 `Control` 声明了一个对等。

C#

```
public class RangePeer1 : FrameworkElementAutomationPeer,
    IRangeValueProvider { }
```

如果所属的控件派生自特定类型的控件（如 RangeBase），则对等可派生自等效的派生对等类。在此示例中，对等将从 RangeBaseAutomationPeer 派生，它提供了 IRangeValueProvider 的基本实现。以下代码演示了此类对等的声明。

C#

```
public class RangePeer2 : RangeBaseAutomationPeer { }
```

有关示例实现，请参阅实现和使用 NumericUpDown 自定义控件的 [C#](#) 或 [Visual Basic](#) 源代码。

引发事件

自动化客户端可订阅自动化事件。自定义控件必须通过调用 [RaiseAutomationEvent](#) 方法来报告控件状态的更改。同样，更改属性值时，调用 [RaisePropertyChangedEvent](#) 方法。以下代码演示了如何在控件代码内获取对等对象，并调用一个方法来引发事件。作为一种优化，该代码会确定是否有适用于此事件类型的任何侦听器。仅当有侦听器时才引发事件，可避免不必要的开销，有助于控件保持响应状态。

C#

```
if (AutomationPeer.ListenerExists(AutomationEvents.PropertyChanged))
{
    NumericUpDownAutomationPeer peer =
        UIElementAutomationPeer.FromElement(nudCtrl) as
    NumericUpDownAutomationPeer;

    if (peer != null)
    {
        peer.RaisePropertyChangedEvent(
            RangeValuePatternIdentifiers.ValueProperty,
            (double)oldValue,
            (double)newValue);
    }
}
```

请参阅

- [UI 自动化概述](#)
- [服务器端 UI 自动化提供程序的实现](#)
- [GitHub 上的NumericUpDown 自定义控件 \(C#\)](#)
- [GitHub 上的NumericUpDown 自定义控件 \(Visual Basic\)](#)

演练：创建自定义的动画按钮

项目 • 2023/02/06

顾名思义，Windows Presentation Foundation (WPF) 非常适合为客户提供丰富的演示体验。这些演练将展示如何自定义按钮的外观和行为（包括动画）。此自定义通过使用样式和模板完成，这样你可以将此自定义按钮轻松应用于应用程序中的任何按钮。下图显示了将要创建的自定义按钮。



自定义按钮

构成按钮外观的矢量图形使用 XAML 创建（与使用 HTML 创建的类似），但更强大且更具可扩展性。Extensible Application Markup Language (XAML) 可以使用 Visual Studio 或记事本手动键入，也可以使用 Blend for Visual Studio 等可视化设计工具。Blend 的工作原理是创建基础 XAML 代码，因此两种方法都创建相同的图形。

本节内容

[使用 Microsoft Expression Blend 创建按钮](#)演示了如何使用 Expression Blend 的设计器功能创建具有自定义行为的按钮。

[使用 XAML 创建按钮](#)演示了如何使用 XAML 和 Visual Studio 创建具有自定义行为的按钮。

相关章节

[样式和模板](#)介绍了如何使用样式和模板来确定控件的外观和行为。

[动画概述](#)介绍了如何使用 WPF 动画和计时系统为对象设置动画。

[使用纯色和渐变进行绘制的概述](#)介绍了如何通过画笔对象使用纯色、线性渐变和径向渐变进行绘制。

[位图效果概述](#)介绍了 WPF 支持的位图效果，并说明如何应用这些效果。

演练：使用 Microsoft Expression Blend 创建按钮

项目 • 2022/09/27

本演练引导你完成使用 Microsoft Expression Blend 创建 WPF 自定义按钮的过程。

① 重要

Microsoft Expression Blend 的工作方式是生成 Extensible Application Markup Language (XAML) , 随后进行编译以生成可执行程序。如果宁愿直接使用 XAML , 则还有另一个演练 , 它使用 XAML 和 Visual Studio (而不是 Blend) 创建与此相同的应用程序。有关详细信息 , 请参阅[使用 XAML 创建按钮](#)。

下图显示了将要创建的自定义按钮。



将形状转换为按钮

在本演练的第一部分 , 将创建自定义按钮的自定义外观。为此 , 首先将矩形转换为按钮。然后将其他形状添加到按钮的模板 , 从而创建具有更复杂外观的按钮。为什么不从常规按钮开始并对其进行自定义 ? 因为按钮具有不需要的内置功能 ; 对于自定义按钮 , 从矩形开始会更加轻松。

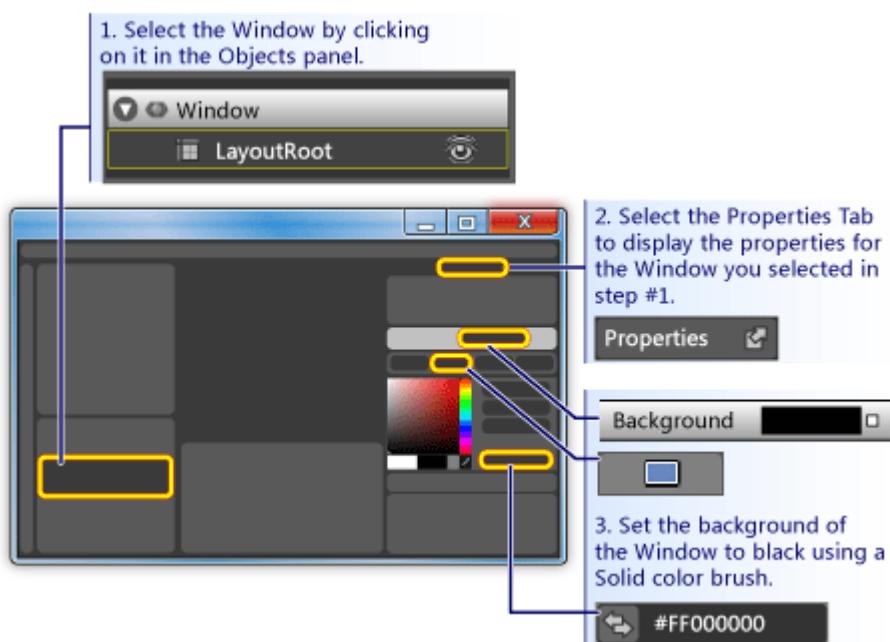
在 Expression Blend 中创建新项目

1. 启动 Expression Blend。（单击“开始”，指向“所有程序”，指向“Microsoft Expression”，然后单击“Microsoft Expression Blend”。）
2. 在需要时使应用程序最大化。
3. 在“文件”菜单上，单击“新建项目”。
4. 选择“标准应用程序(.exe)”。
5. 将项目命名为 `CustomButton`，然后按“确定”。

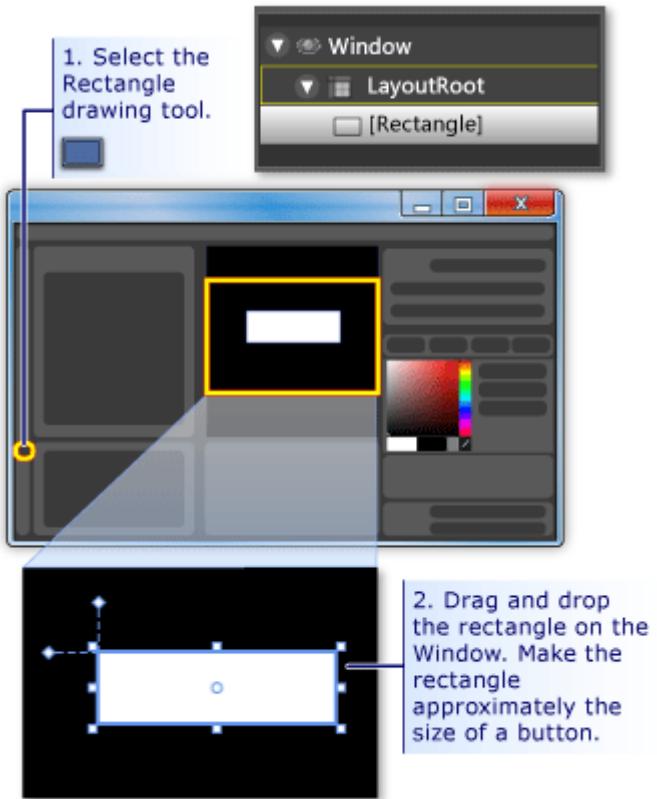
此时，你有一个空白 WPF 项目。可以按 F5 运行该应用程序。如你所料，应用程序仅包含一个空白窗口。接下来，创建一个圆角矩形并将其转换为按钮。

将矩形转换为按钮

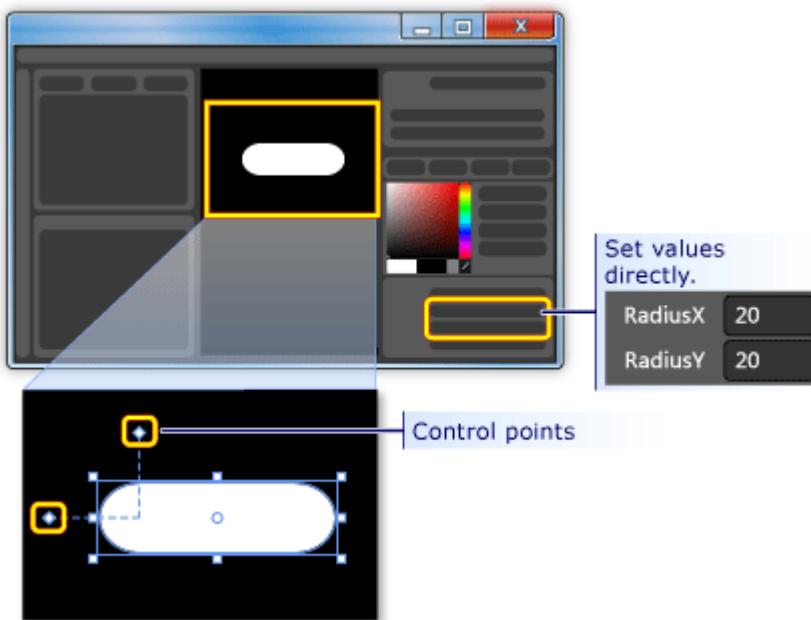
1. 将窗口背景属性设置为黑色：选择窗口，单击“属性”选项卡，然后将 `Background` 属性设置为 `Black`。



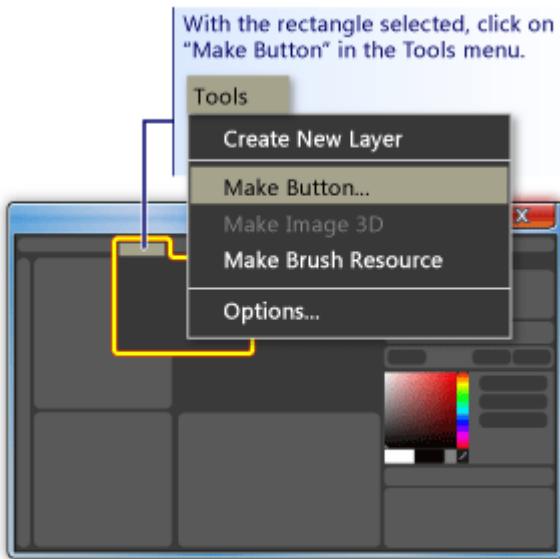
2. 在窗口上绘制一个大约为按钮大小的矩形：选择左侧工具面板中的矩形工具，然后将矩形拖到窗口上。



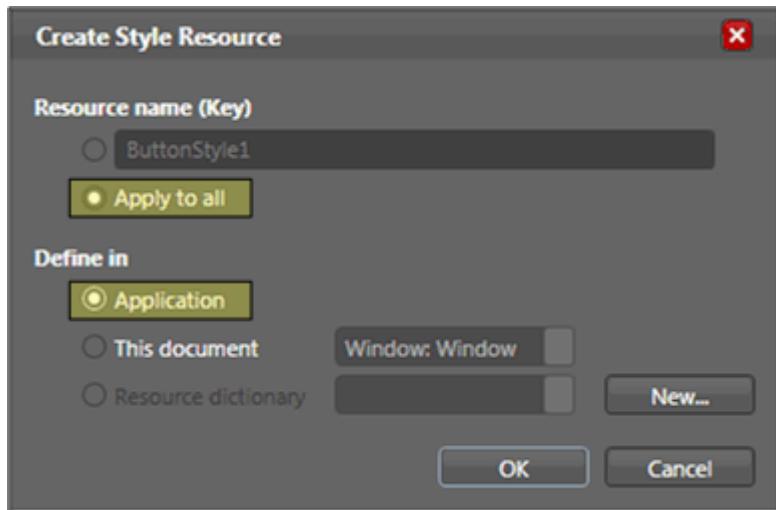
- 使矩形的各个角成为圆角：拖动矩形的控制点或直接设置 RadiusX 和 RadiusY 属性。将 RadiusX 和 RadiusY 的值设置为 20。



- 将矩形更改为按钮：选择矩形。在“工具”菜单上，单击“生成按钮”。



5. 指定样式/模板的范围：会显示如下所示的对话框。



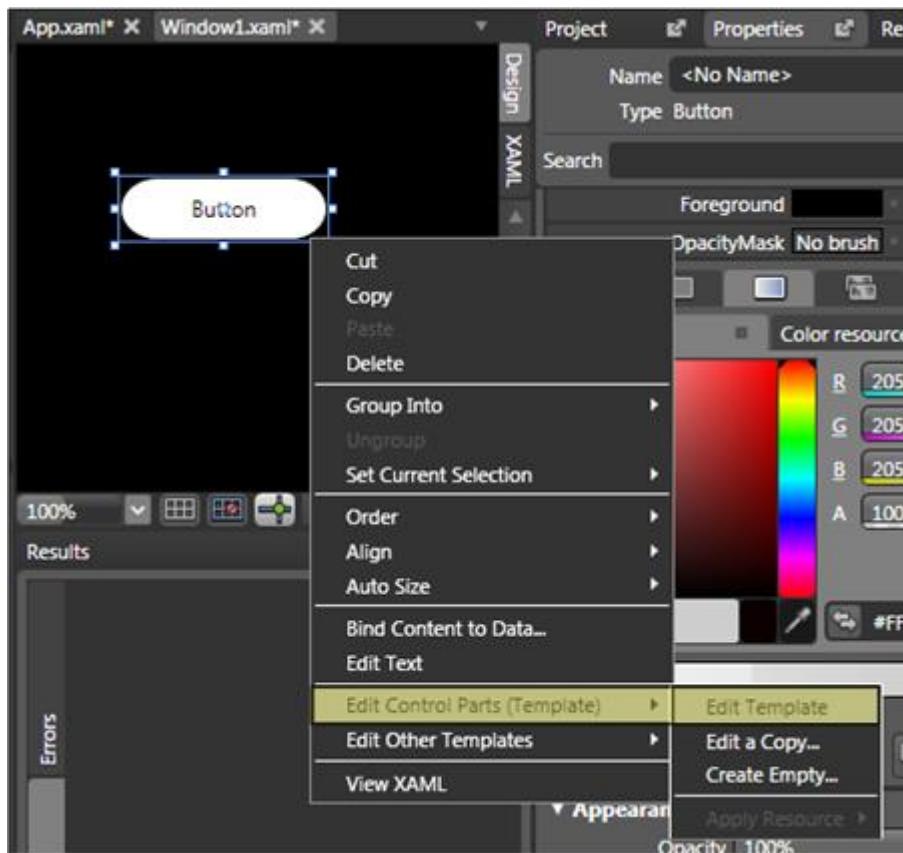
对于“资源名(关键字)”，请选择“全部应用”。这会使生成的样式和按钮模板应用于所有按钮对象。对于“定义位置”，请选择“应用程序”。这会使生成的样式和按钮模板的作用域是整个应用程序。在这两个框中设置值时，按钮样式和模板会应用于整个应用程序中的所有按钮，在应用程序中创建的任何按钮都会默认使用此模板。

编辑按钮模板

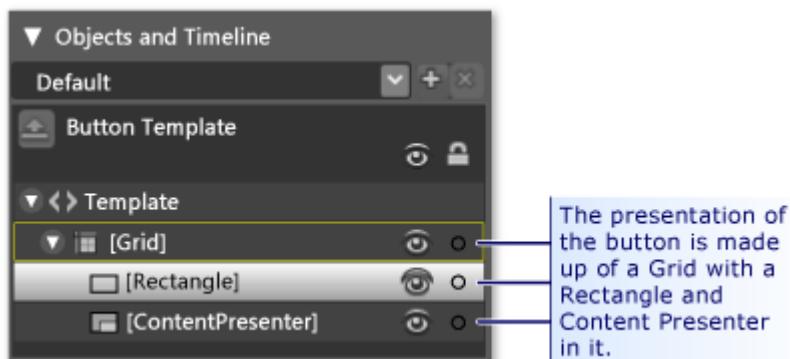
你现在有一个已更改为按钮的矩形。在本部分中，你将修改按钮的模板，并进一步自定义其外观。

编辑按钮模板以更改按钮外观

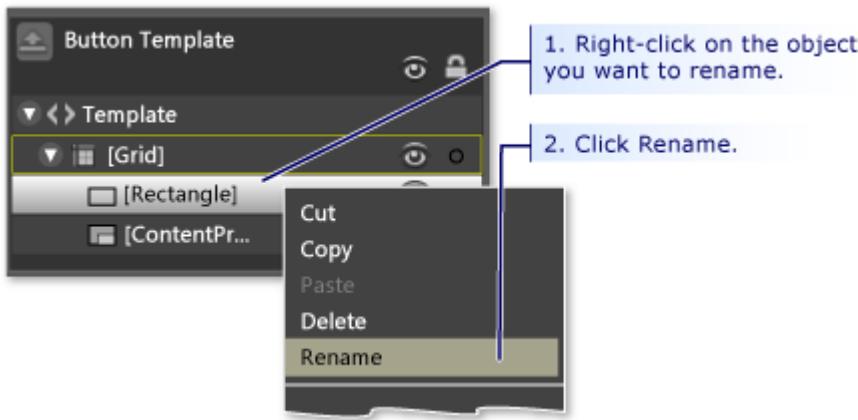
1. 进入编辑模板视图：若要进一步自定义按钮的外观，需要编辑按钮模板。此模板是在将矩形转换为按钮时创建的。若要编辑按钮模板，请右键单击按钮，选择“编辑控件部件(模板)”，然后选择“编辑模板”。



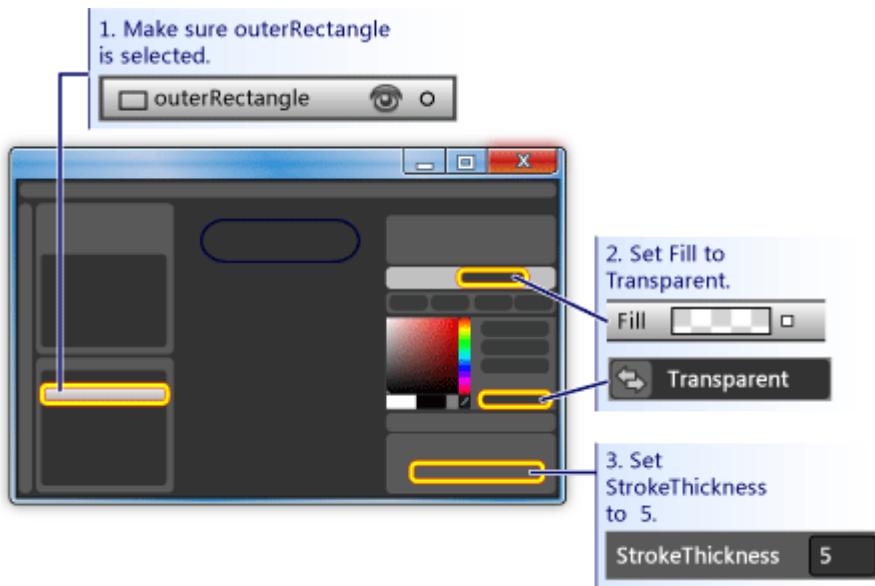
在模板编辑器中，请注意，按钮现在分隔为 [Rectangle](#) 和 [ContentPresenter](#)。
[ContentPresenter](#) 用于在按钮中呈现内容（例如字符串“Button”）。矩形和 [ContentPresenter](#) 都处于 [Grid](#) 内部。



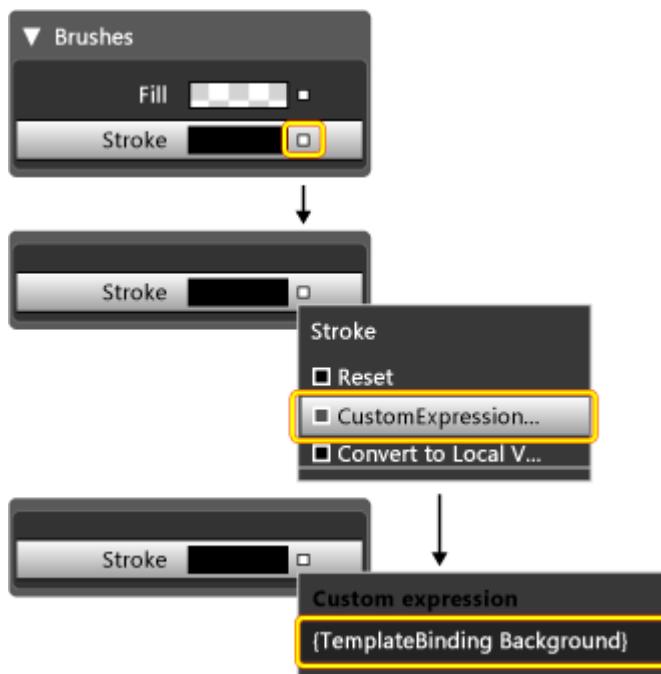
2. 更改模板组件的名称：右键单击模板清单中的矩形，将 [Rectangle](#) 名称从 “[Rectangle]”更改为“outerRectangle”，并将 “[ContentPresenter]”更改为 “myContentPresenter”。



3. 更改矩形，使其内部为空（类似于圆环）：选择“outerRectangle”，将 Fill 设置为“Transparent”，将 StrokeThickness 设置为 5。



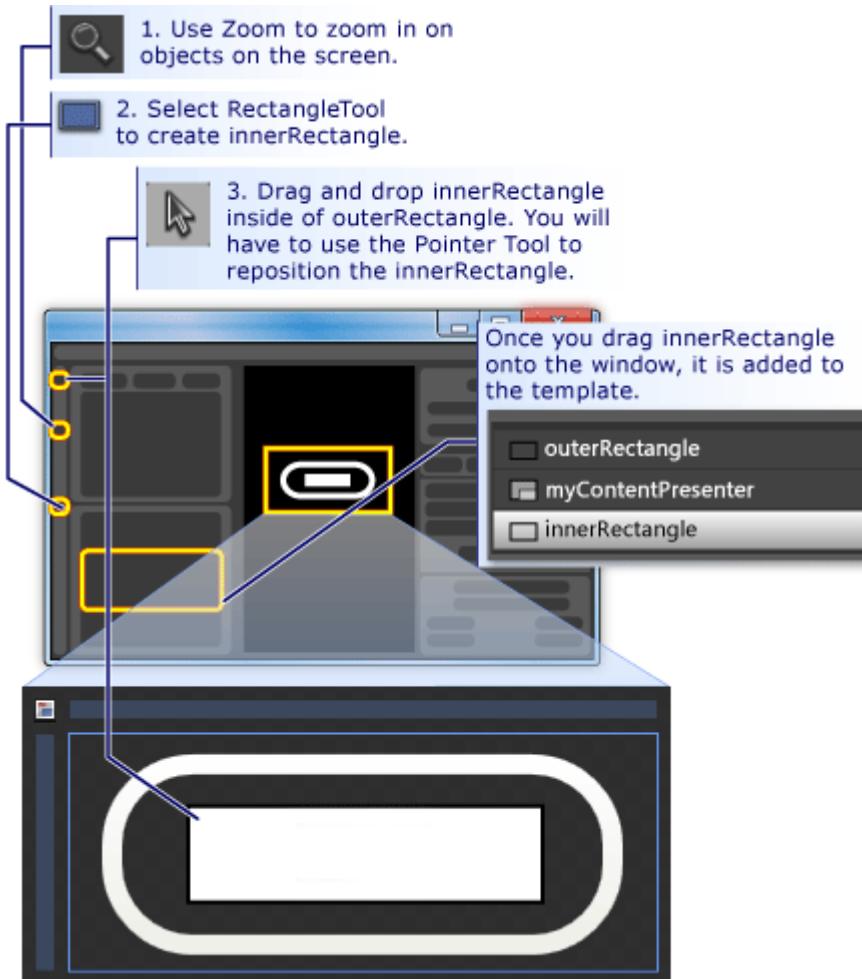
然后将 Stroke 设置为模板将采用的任何颜色。为此，请单击“笔划”旁的白色小框，选择“CustomExpression”，然后在对话框中键入“{TemplateBinding Background}”。



4. 创建内部矩形：现在，创建另一个矩形（将它命名为“innerRectangle”），并将它对称地放置在 outerRectangle 内部。对于此类工作，你可能希望进行缩放以使按钮在编辑区域中更大。

① 备注

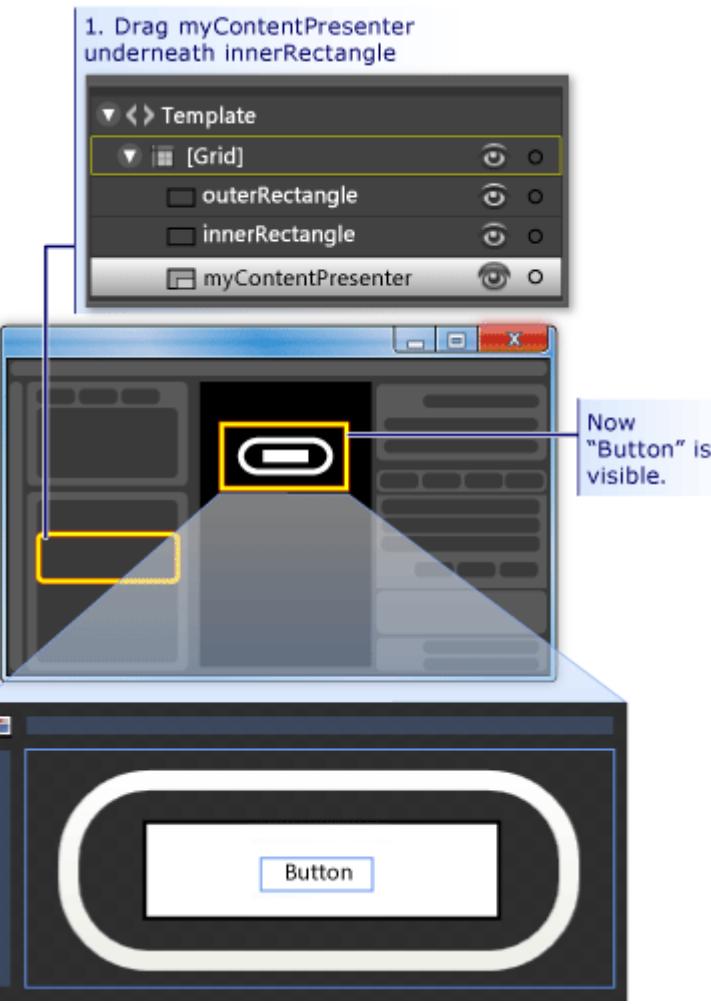
你的矩形看起来可能与图中的矩形不同（例如，它可能具有圆角）。



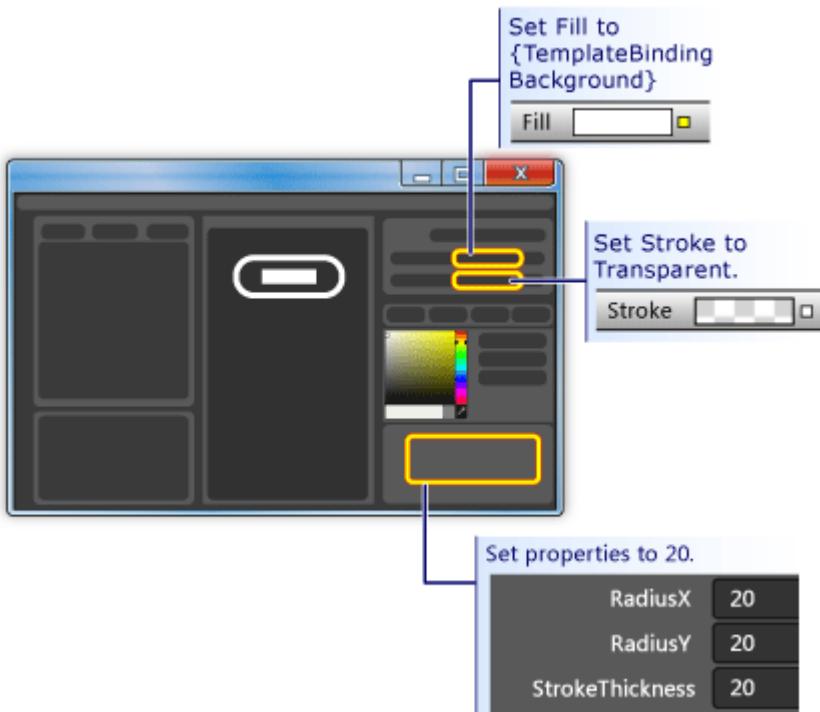
5. 将 ContentPresenter 移动到上层：此时，文本“Button”可能不再可见。如果是这样，这是因为 innerRectangle 处于 myContentPresenter 的上层。若要解决此问题，请拖动 innerRectangle 下层的 myContentPresenter。重新定位矩形和 myContentPresenter，使其类似于下面这样。

① 备注

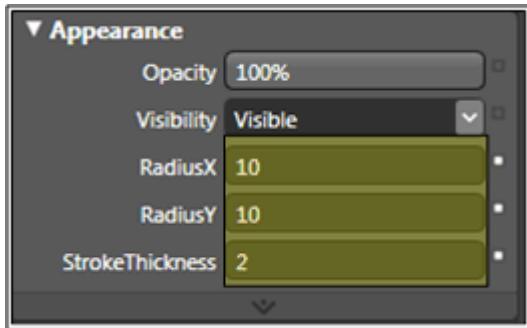
或者，也可以通过右键单击 myContentPresenter 并按“向前移动”将其置于顶层。



6. 更改 innerRectangle 的外观：将 RadiusX、RadiusY 和 StrokeThickness 值设置为 20。此外，使用自定义表达式“{TemplateBinding Background}”将 Fill 设置为模板的背景，并将 Stroke 设置为“transparent”。请注意，innerRectangle 的 Fill 和 Stroke 设置与 outerRectangle 的这些设置相反。



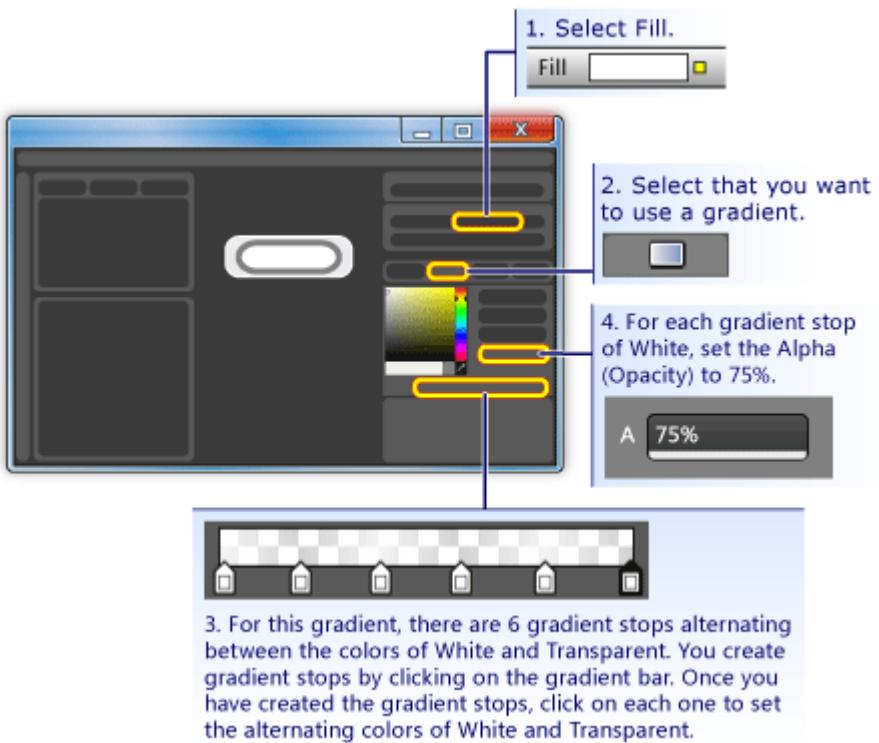
7. 在顶层添加玻璃层：自定义按钮外观的最后一部分是在顶层添加玻璃层。此玻璃层由第三个矩形组成。由于玻璃会覆盖整个按钮，因此玻璃矩形在尺寸上类似于 outerRectangle。因此，只需创建 outerRectangle 的副本即可创建矩形。突出显示 outerRectangle 并使用 CTRL+C 和 CTRL+V 创建副本。将此新矩形命名为“glassCube”。
8. 在需要时重新定位 glassCube：如果 glassCube 的位置尚未使其覆盖整个按钮，请将其拖动到位。
9. 为 glassCube 提供与 outerRectangle 略有不同的形状：更改 glassCube 的属性。首先将 RadiusX 和 RadiusY 属性更改为 10，并将 StrokeThickness 更改为 2。



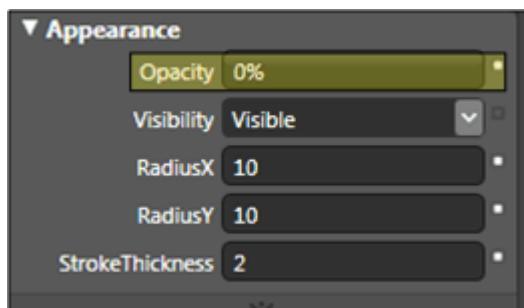
10. 使 glassCube 看起来像玻璃：使用线性渐变将 Fill 设置为玻璃外观，该渐变为 75% 不透明，在 6 个大致均匀的间隔之间交替使用白色和透明颜色。下面是渐变停止点的设置内容：

- 渐变停止点 1：Alpha 值为 75% 的白色
- 渐变停止点 2：透明
- 渐变停止点 3：Alpha 值为 75% 的白色
- 渐变停止点 4：透明
- 渐变停止点 5：Alpha 值为 75% 的白色
- 渐变停止点 6：透明

这会创建“波浪”玻璃外观。



11. 隐藏玻璃层：现在可看到玻璃层的外观，进入“属性”面板的“外观”窗格，将不透明度设置为 0% 以隐藏它。在接下来的部分中，我们将使用属性触发器和事件来显示和操作玻璃层。

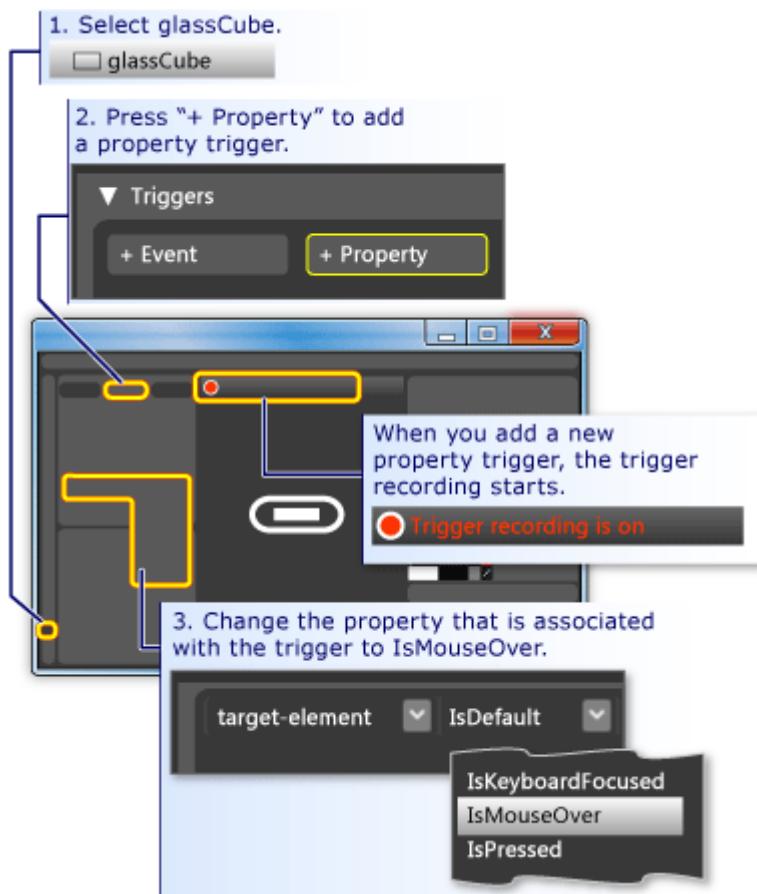


自定义按钮行为

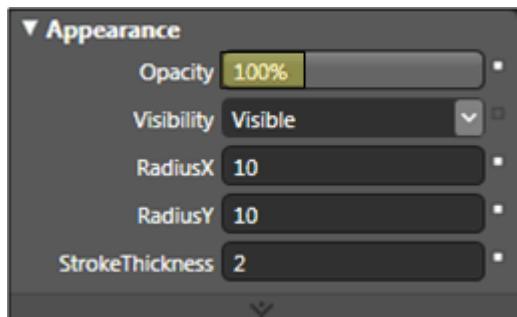
此时，你已通过编辑按钮模板自定义了按钮的呈现，但按钮不会如同典型按钮一样响应用户操作（例如，在鼠标悬停在上方、获得焦点和单击时更改外观。）接下来的两个过程演示了如何在自定义按钮中构建此类行为。我们将从简单的属性触发器开始，然后添加事件触发器和动画。

设置属性触发器

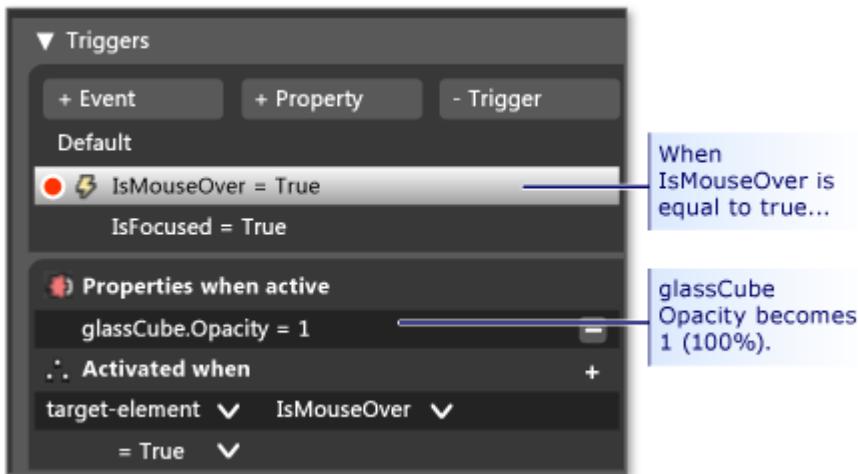
- 创建新的属性触发器：选中 glassCube 后，单击“触发器”面板中的“+ 属性”（请参阅下一步后面的图）。这会创建具有默认属性触发器的属性触发器。
- 使 `IsMouseOver` 成为触发器使用的属性：将属性更改为 `IsMouseOver`。这可在 `IsMouseOver` 属性为 `true` 时（用户使用鼠标指向按钮时）使属性触发器激活。



3. IsMouseOver 为 glassCube 触发 100% 的不透明度：请注意“触发器记录已开启”（请参阅上图）。这意味着，在记录开启期间对 glassCube 的属性值进行的任何更改都会成为在 IsMouseOver 为 true 时进行的操作。在记录期间，将 glassCube 的 Opacity 更改为 100%。

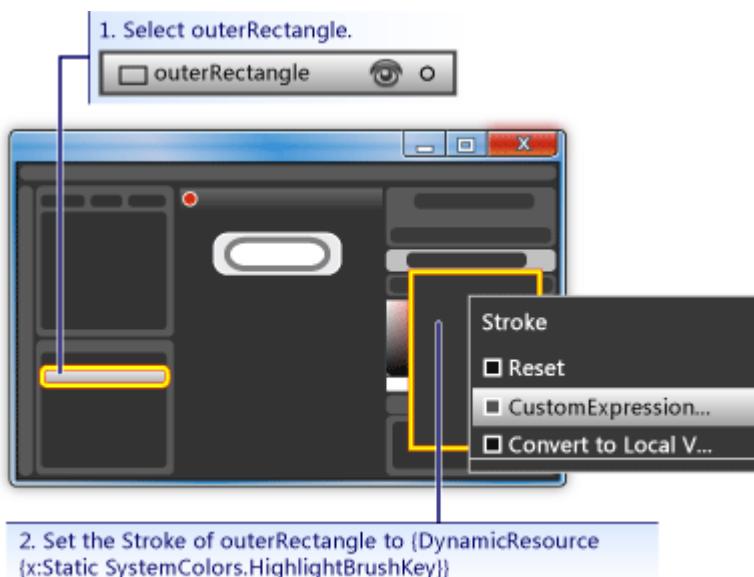


现在创建了第一个属性触发器。请注意，编辑器的“触发器”面板 记录了 Opacity 已更改为 100%。

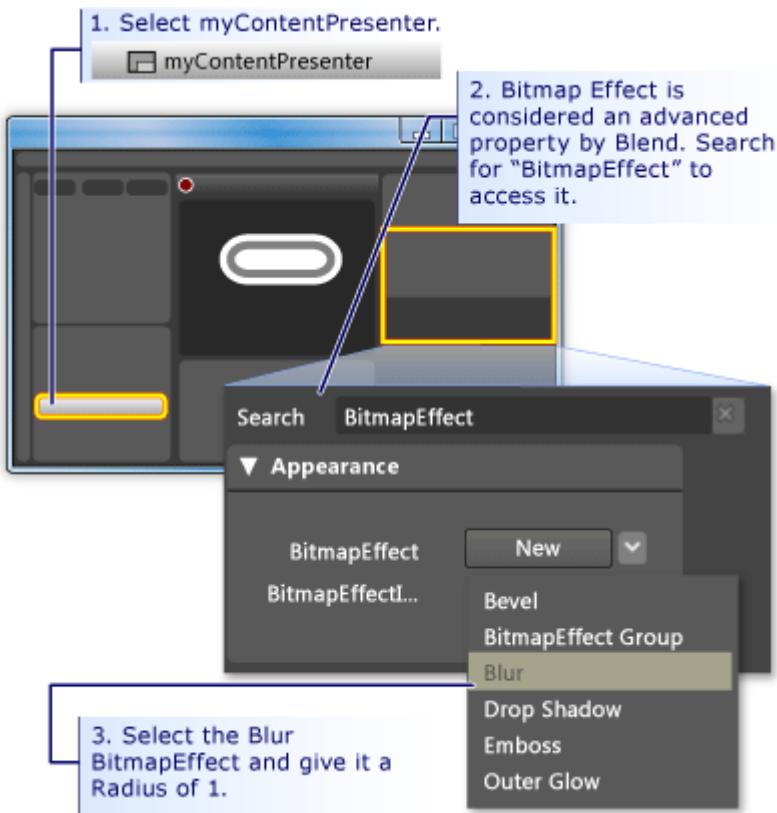


按 F5 运行应用程序，然后将鼠标指针移到按钮上方并移走。你应看到，玻璃层会在鼠标悬停在按钮上方出现，在鼠标离开时消失。

4. IsMouseOver 触发笔划值更改：让我们将其他一些操作与 `IsMouseOver` 触发器关联。在记录继续进行期间，将所选内容从 `glassCube` 切换到 `outerRectangle`。然后将 `outerRectangle` 的 `Stroke` 设置为自定义表达式“`{DynamicResource {x:Static SystemColors.HighlightBrushKey}}`”。这会将 `Stroke` 设置为按钮使用的典型突出显示颜色。按 F5 以查看鼠标位于按钮上方时的效果。



5. IsMouseOver 触发模糊文本：让我们将另一个操作关联到 `IsMouseOver` 属性触发器。当玻璃出现在按钮上时，使按钮的内容显得有点模糊。为此，我们可以将模糊 `BitmapEffect` 应用于 `ContentPresenter` (`myContentPresenter`)。



① 备注

若要使“属性”面板恢复为对 `BitmapEffect` 进行搜索之前的内容，请清除“搜索”框中的文本。

此时，我们使用具有多个关联操作的属性触发器创建了鼠标指针进入并离开按钮区域时的突出显示行为。按钮的另一个典型行为是在它获得焦点时（如单击它后）突出显示。可以通过为 `IsFocused` 属性添加另一个属性触发器来添加此类行为。

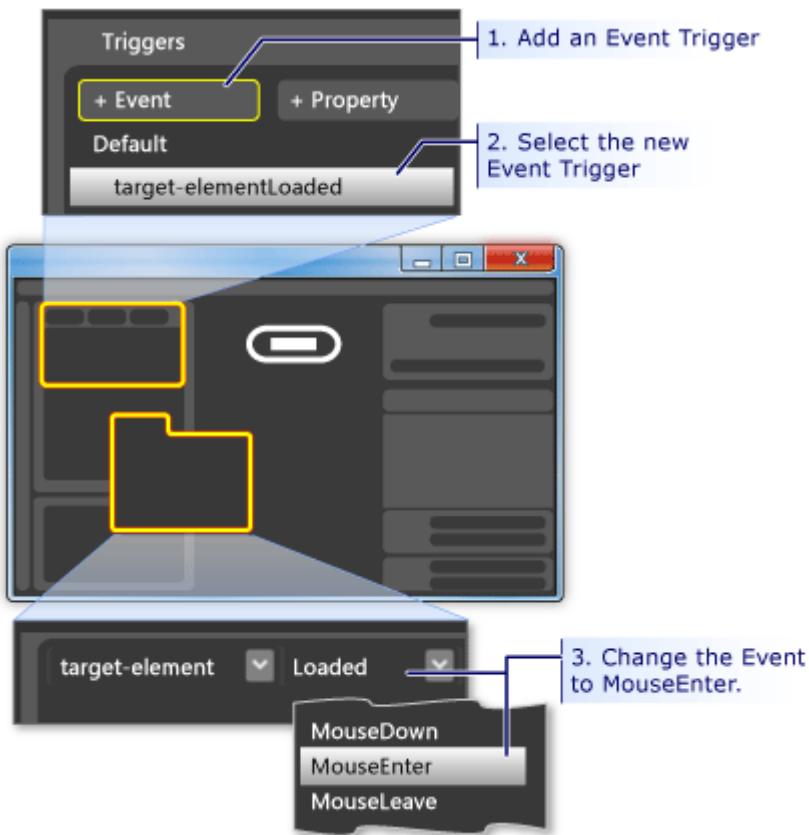
6. 为 `IsFocused` 创建属性触发器：使用与 `IsMouseOver` 相同的过程（请参阅本部分的第一步），为 `IsFocused` 属性创建另一个属性触发器。在“触发器记录已开启”期间，将以下操作添加到触发器：

- `glassCube` 获取 100% 的 `Opacity`。
- `outerRectangle` 获取 `Stroke` 自定义表达式值“`{DynamicResource {x:Static SystemColors.HighlightBrushKey}}`”。

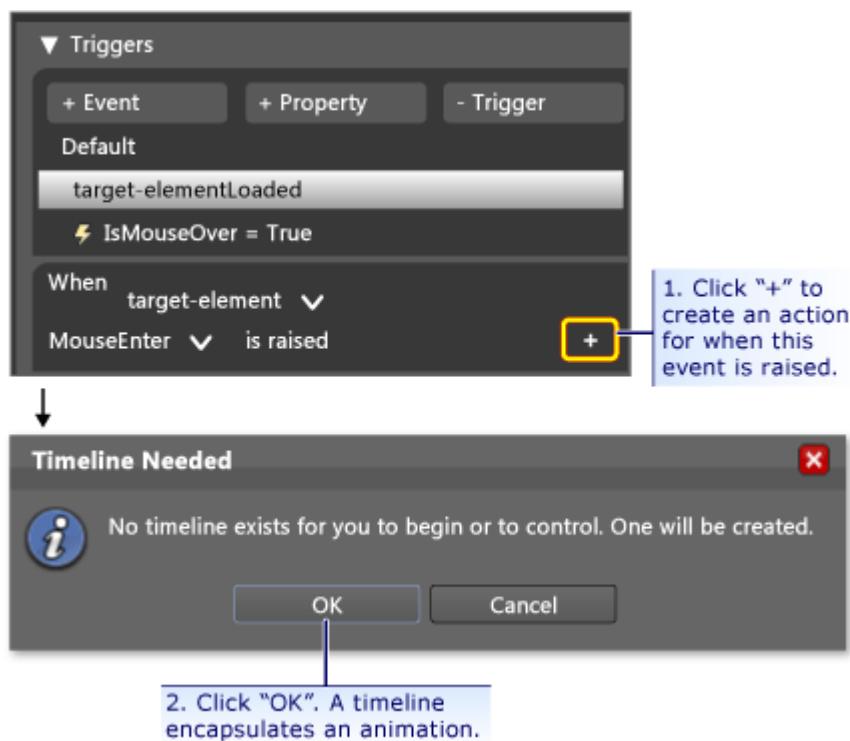
在本演练的最后一步中，我们将向按钮添加动画。这些动画将由事件触发（特别是 `MouseEnter` 和 `Click` 事件）。

使用事件触发器和动画添加交互性

1. 创建 `MouseEnter` 事件触发器：添加新事件触发器，然后选择 `MouseEnter` 作为要在触发器中使用的事件。



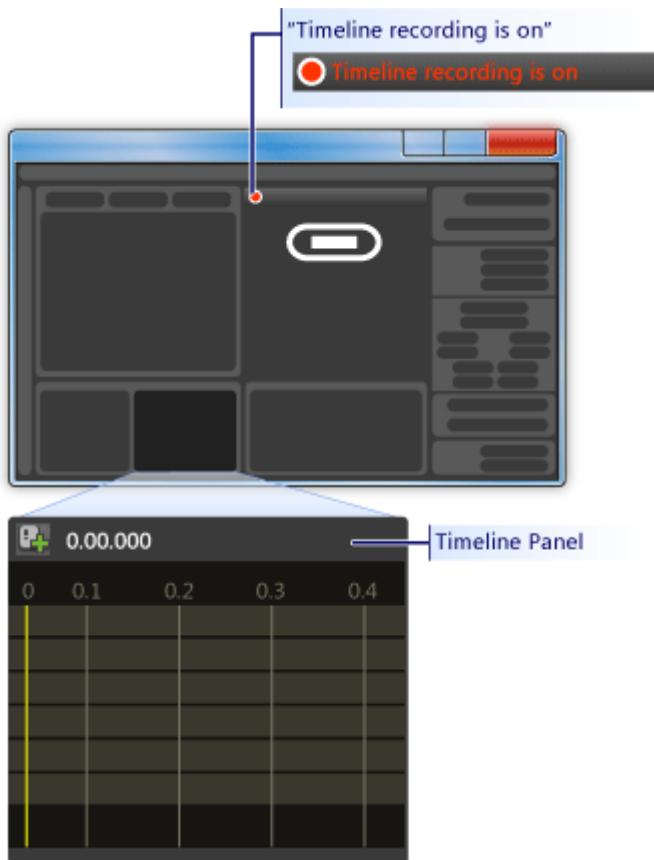
2. 创建动画时间线：接下来，将动画时间线关联到 [MouseEnter](#) 事件。



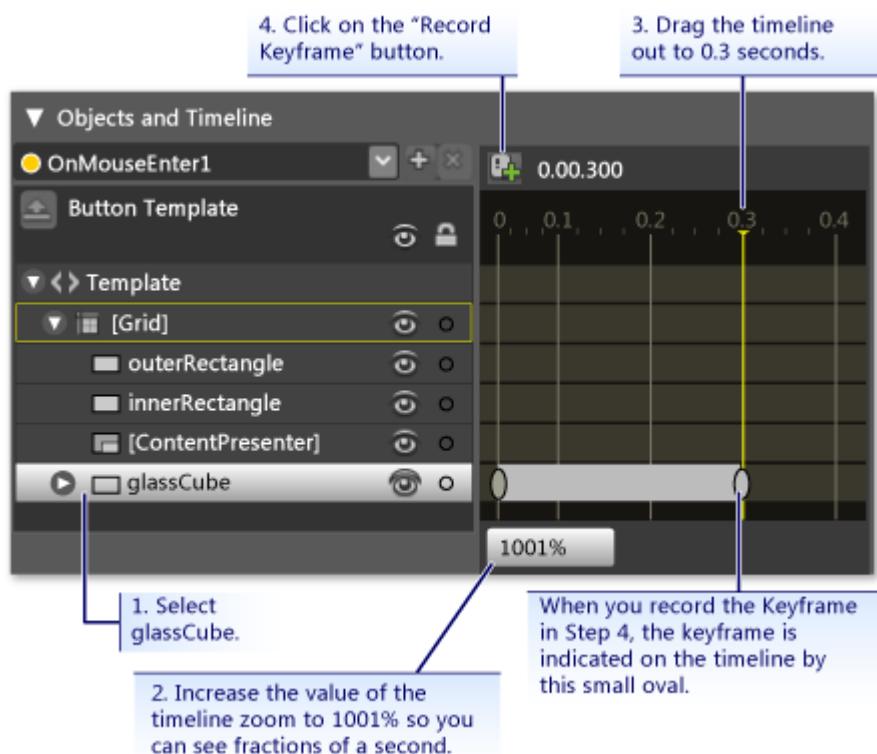
按“确定”创建新时间线后，将显示时间线面板，“时间线记录已开启”会在设计面板中可见。这意味着我们可以开始在时间线中记录属性更改（对属性更改进行动画处理）。

① 备注

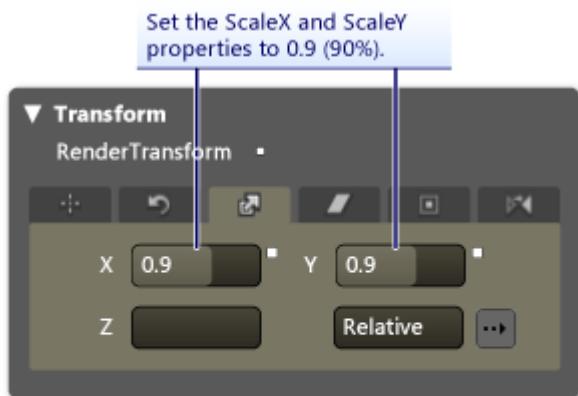
可能需要重设窗口和/或面板的大小才能查看显示。



3. 创建关键帧：若要创建动画，请选择要进行动画处理的对象，在时间线上创建两个或更多关键帧，并为这些关键帧设置希望动画在它们之间内插的属性值。下图指导你创建关键帧。



4. 在此关键帧处缩小 glassCube：选择第二个关键帧后，使用“大小转换”将 glassCube 的大小缩小到其完整大小的 90%。

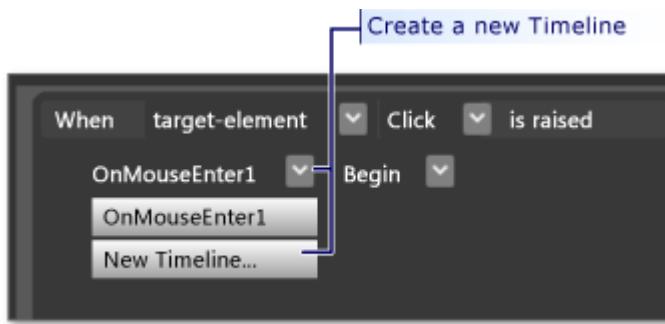


Note: you will probably have to expand the Transform section of the Properties panel.

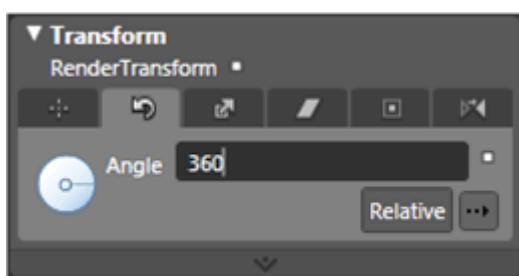
按 F5 运行该应用程序。 将鼠标指针移到按钮上。 请注意，玻璃层在按钮上层缩小。

5. 创建另一个事件触发器并将其他动画与它关联：让我们添加另一个动画。 使用与用于创建上一个事件触发器动画类似的过程：

- 使用 Click 事件创建新的事件触发器。
- 将新时间线与 Click 事件关联。



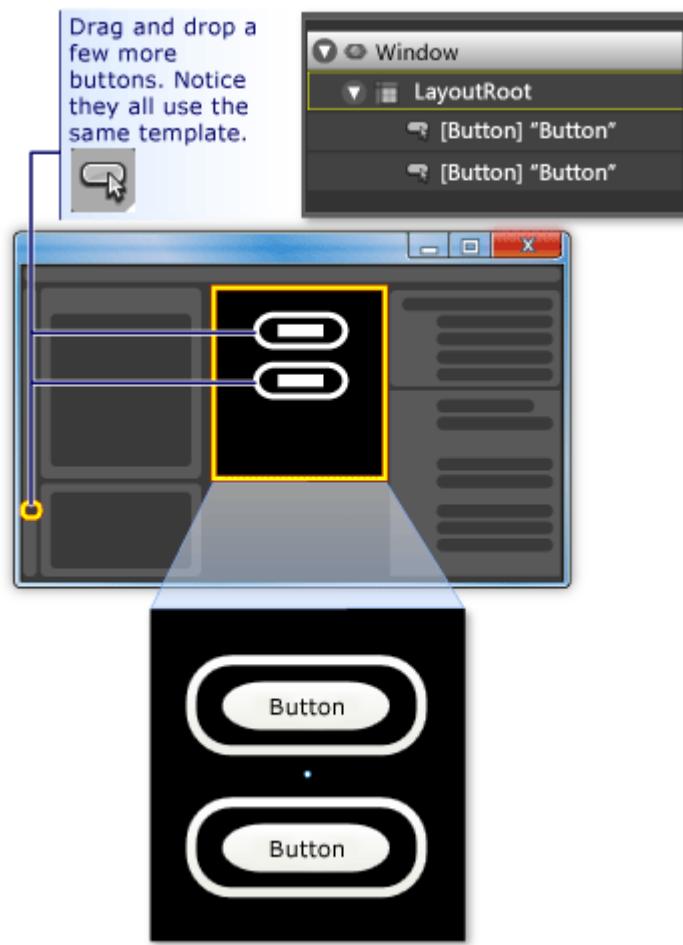
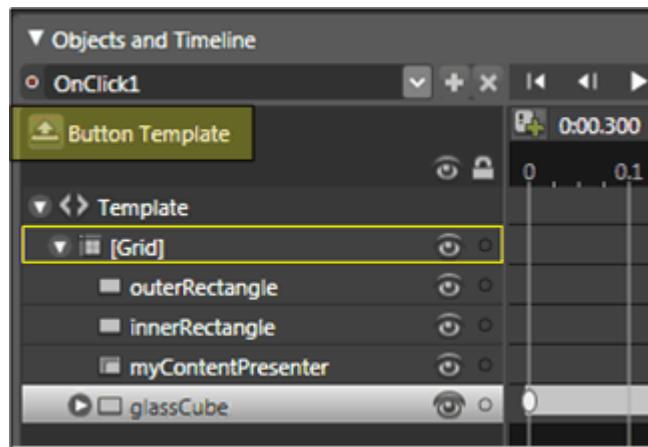
- 对于此时间线，创建两个关键帧，一个帧在 0.0 秒，第二个帧在 0.3 秒。
- 在 0.3 秒的关键帧突出显示后，将“旋转转换角度”设置为 360 度。



- 按 F5 运行该应用程序。 单击按钮。 请注意，玻璃层会旋转。

结论

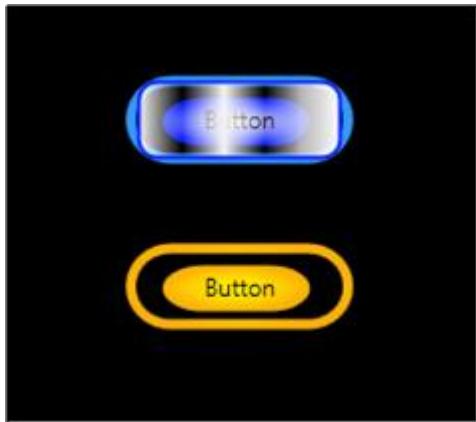
你已完成了自定义按钮。你是使用应用于应用程序中所有按钮的按钮模板完成了此任务。如果离开模板编辑模式（请参阅下图）并创建更多按钮，你会看到其外观和行为类似于自定义按钮，而不是默认按钮。



按 F5 运行该应用程序。单击按钮并注意到它们的行为方式全都相同。

请记住，在自定义模板期间，将 innerRectangle 的 Fill 属性和 outerRectangle 的 Stroke 属性设置为模板背景 ({TemplateBinding Background})。因此，当你设置各个按钮的背景时，你设置的背景将用于这些各自的属性。现在尝试更改背景。在下图中，使用了不

同的渐变。因此，尽管模板可用于控件（如按钮）的整体自定义，但仍可修改使用模板的控件，使其外观彼此不同。



总之，在自定义按钮模板的过程中，你已了解如何在 Microsoft Expression Blend 中执行以下操作：

- 自定义控件的外观。
- 设置属性触发器。属性触发器非常有用，因为它们可用于大多数对象，而不仅仅是控件。
- 设置事件触发器。事件触发器非常有用，因为它们可用于大多数对象，而不仅仅是控件。
- 创建动画。
- 其他：创建渐变、添加 BitmapEffect、使用转换以及设置对象的基本属性。

另请参阅

- [使用 XAML 创建按钮](#)
- [样式设置和模板化](#)
- [动画概述](#)
- [使用纯色和渐变进行绘制概述](#)
- [位图效果概述](#)

演练：使用 XAML 创建按钮

项目 • 2023/02/06

本演练的目的是了解如何创建用于 Windows Presentation Foundation (WPF) 应用程序的动画按钮。本演练使用样式和模板创建一个自定义按钮资源，该资源允许重用代码并将按钮逻辑与按钮声明分离。本演练完全使用 Extensible Application Markup Language (XAML) 编写。

① 重要

本演练将指导你通过将 Extensible Application Markup Language (XAML) 键入或复制并粘贴到 Visual Studio 来创建应用程序。如果你想了解如何使用设计器创建相同的应用程序，请参阅[使用 Microsoft Expression Blend 创建按钮](#)。

下图显示了完成的按钮。



创建基本按钮

首先创建一个新项目并向窗口中添加几个按钮。

创建新的 WPF 项目并向窗口添加按钮

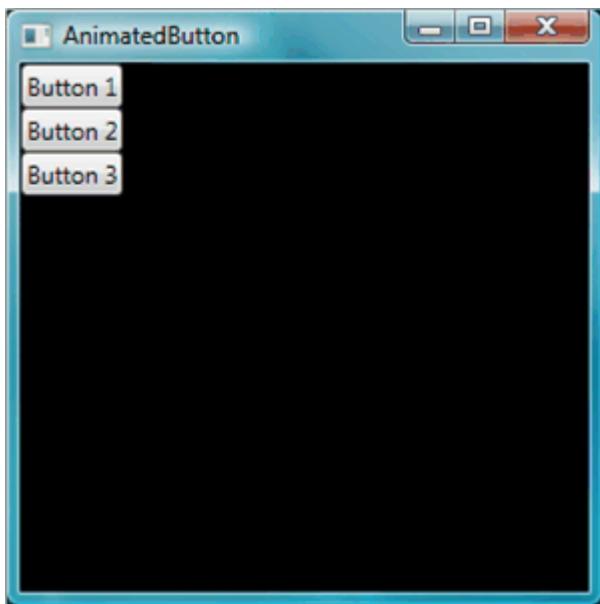
1. 启动 Visual Studio。
2. **创建新的 WPF 项目**：在“文件”菜单上，指向“新建”，然后单击“项目”。找到“Windows 应用程序(WPF)”模板并将项目命名为“AnimatedButton”。这将为应用程序创建框架。

3. **添加基本默认按钮**：本演练所需的所有文件均由模板提供。在解决方案资源管理器中双击打开 Window1.xaml 文件。默认情况下，Window1.xaml 中有一个 Grid 元素。删除 Grid 元素并向 Extensible Application Markup Language (XAML) 页面添加几个按钮，方法是将以下突出显示的代码键入或复制并粘贴到 Window1.xaml 中：

```
XAML

<Window x:Class="AnimatedButton.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="AnimatedButton" Height="300" Width="300"
    Background="Black">
    <!-- Buttons arranged vertically inside a StackPanel. -->
    <StackPanel HorizontalAlignment="Left">
        <Button>Button 1</Button>
        <Button>Button 2</Button>
        <Button>Button 3</Button>
    </StackPanel>
</Window>
```

按 F5 运行应用程序；你应该会看到一组如下图所示的按钮。



基本按钮创建完毕，你已经完成了 Window1.xaml 文件中的工作。本演练的其余部分侧重于 app.xaml 文件、定义样式和按钮模板。

设置基本属性

接下来，让我们在这些按钮上设置一些属性来控制按钮的外观和布局。你将使用资源为整个应用程序定义按钮属性，而不是在按钮上单独设置属性。应用程序资源在概念上类

似于网页的外部级联样式表 (CSS)；但是，如本演练结束时所示，资源比级联样式表 (CSS) 更强大。若要详细了解资源，请参阅 [XAML 资源](#)。

使用样式设置按钮的基本属性

1. 定义 Application.Resources 块：打开 app.xaml 并添加以下突出显示的标记（如果尚不存在）：

XAML

```
<Application x:Class="AnimatedButton.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="Window1.xaml"
    >
    <Application.Resources>
        <!-- Resources for the entire application can be defined here. -->
    </Application.Resources>
</Application>
```

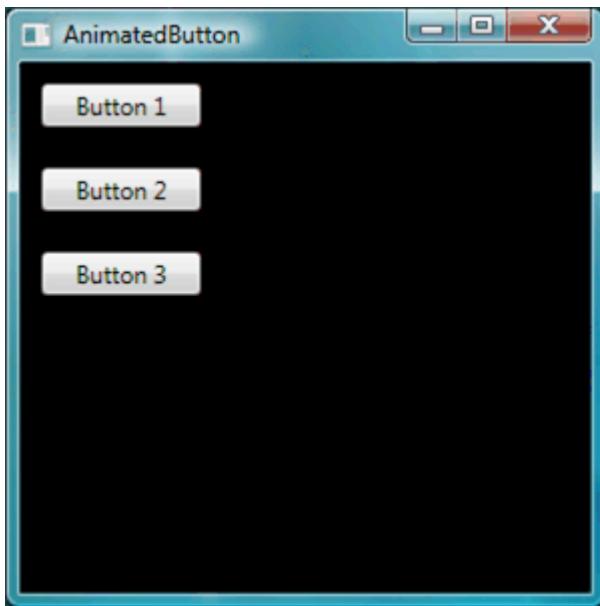
资源范围取决于定义资源的位置。通过在 app.xaml 文件的 Application.Resources 中定义资源，可以在应用程序的任何位置使用该资源。若要详细了解如何定义资源范围，请参阅 [XAML 资源](#)。

2. 创建样式并使用它定义基本属性值：将以下标记添加到 Application.Resources 块。此标记会创建一个适用于应用程序中所有按钮的 Style，并将按钮的 Width 设置为 90，将 Margin 设置为 10：

XAML

```
<Application.Resources>
    <Style TargetType="Button">
        <Setter Property="Width" Value="90" />
        <Setter Property="Margin" Value="10" />
    </Style>
</Application.Resources>
```

TargetType 属性指定样式适用于 Button 类型的所有对象。每个 Setter 为 Style 设置一个不同的属性值。因此，此时应用程序中每个按钮的宽度为 90，边距为 10。如果按 F5 运行应用程序，你会看到以下窗口。



你可以使用样式执行更多操作，包括各种微调目标对象的方法、指定复杂属性值，甚至使用样式作为其他样式的输入。有关详细信息，请参阅[样式设置和模板化](#)。

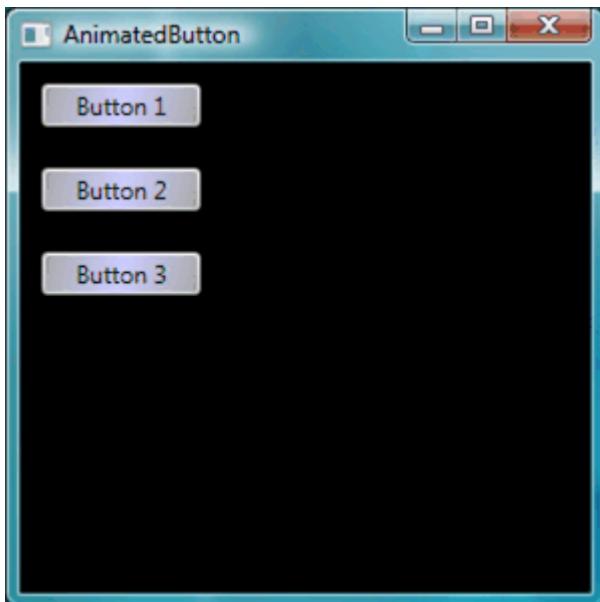
3. 将样式属性值设置为资源：资源提供一种简单的方法来重用通常定义的对象和值。使用资源定义复杂值以使代码更加模块化特别有用。将以下突出显示的标记添加到 app.xaml。

XAML

```
<Application.Resources>
    <LinearGradientBrush x:Key="GrayBlueGradientBrush" StartPoint="0,0"
EndPoint="1,1">
        <GradientStop Color="DarkGray" Offset="0" />
        <GradientStop Color="#CCCCFF" Offset="0.5" />
        <GradientStop Color="DarkGray" Offset="1" />
    </LinearGradientBrush>
    <Style TargetType="{x:Type Button}">
        <Setter Property="Background" Value="{StaticResource
GrayBlueGradientBrush}" />
        <Setter Property="Width" Value="80" />
        <Setter Property="Margin" Value="10" />
    </Style>
</Application.Resources>
```

在 `Application.Resources` 块的正下方，你创建了一个名为“`GrayBlueGradientBrush`”的资源。此资源定义水平渐变。此资源可用作应用程序中任何位置（包括 `Background` 属性的按钮样式资源库内部）的属性值。现在，所有按钮都具有此渐变的 `Background` 属性值。

按 F5 运行该应用程序。该消息应如下所示。



创建一个定义按钮外观的模板

在本部分中，你将创建一个自定义按钮外观（呈现）的模板。按钮呈现由多个对象组成，包括矩形和其他赋予按钮独特外观的组件。

到目前为止，对应用程序中按钮外观的控制仅限于更改按钮的属性。如果要对按钮外观进行更彻底的更改，该怎么办？模板支持对对象的呈现进行强大的控制。因为模板可用于样式，所以你可以将模板应用于应用了样式的所有对象（在本演练中为按钮）。

使用模板定义按钮的外观

1. **设置模板**：由于像 `Button` 这样的控件具有 `Template` 属性，因此可以像在 `Style` 中使用 `Setter` 设置的其他属性值一样定义模板属性值。将以下突出显示的标记添加到按钮样式。

XAML

```
<Application.Resources>
    <LinearGradientBrush x:Key="GrayBlueGradientBrush"
        StartPoint="0,0" EndPoint="1,1">
        <GradientStop Color="DarkGray" Offset="0" />
        <GradientStop Color="#CCCCFF" Offset="0.5" />
        <GradientStop Color="DarkGray" Offset="1" />
    </LinearGradientBrush>
    <Style TargetType="{x:Type Button}">
        <Setter Property="Background" Value="{StaticResource
GrayBlueGradientBrush}" />
        <Setter Property="Width" Value="80" />
        <Setter Property="Margin" Value="10" />
        <Setter Property="Template">
            <Setter.Value>
                <!-- The button template is defined here. -->
            </Setter.Value>
        </Setter>
    </Style>
</Application.Resources>
```

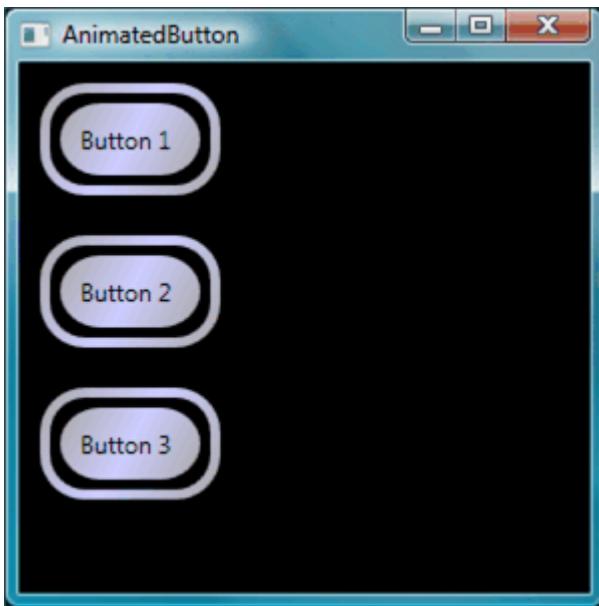
```
</Setter.Value>
</Setter>
</Style>
</Application.Resources>
```

2. **更改按钮呈现**：此时，你需要定义模板。添加以下突出显示的标记。此标记指定两个具有圆角的 Rectangle 元素，后跟一个 DockPanel。DockPanel 用于托管按钮的 ContentPresenter。ContentPresenter 显示按钮的内容。在本演练中，内容为文本（“Button 1”、“Button 2”、“Button 3”）。所有模板组件（矩形和 DockPanel）都布置在 Grid 内。

XAML

```
<Setter.Value>
<ControlTemplate TargetType="Button">
    <Grid Width="{TemplateBinding Width}" Height="{TemplateBinding
Height}" ClipToBounds="True">
        <!-- Outer Rectangle with rounded corners. -->
        <Rectangle x:Name="outerRectangle" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" Stroke="{TemplateBinding Background}"
RadiusX="20" RadiusY="20" StrokeThickness="5" Fill="Transparent" />
        <!-- Inner Rectangle with rounded corners. -->
        <Rectangle x:Name="innerRectangle" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" Stroke="Transparent" StrokeThickness="20"
Fill="{TemplateBinding Background}" RadiusX="20" RadiusY="20" />
        <!-- Present Content (text) of the button. -->
        <DockPanel Name="myContentPresenterDockPanel">
            <ContentPresenter x:Name="myContentPresenter" Margin="20"
Content="{TemplateBinding Content}" TextBlock.Foreground="Black" />
        </DockPanel>
    </Grid>
</ControlTemplate>
</Setter.Value>
```

按 F5 运行该应用程序。该消息应如下所示。



3. **向模板添加玻璃效果**：接下来添加玻璃效果。首先，创建一些可呈现玻璃渐变效果的资源。在 `Application.Resources` 块中的任意位置添加这些渐变资源：

XAML

```
<Application.Resources>
    <GradientStopCollection x:Key="MyGlassGradientStopsResource">
        <GradientStop Color="WhiteSmoke" Offset="0.2" />
        <GradientStop Color="Transparent" Offset="0.4" />
        <GradientStop Color="WhiteSmoke" Offset="0.5" />
        <GradientStop Color="Transparent" Offset="0.75" />
        <GradientStop Color="WhiteSmoke" Offset="0.9" />
        <GradientStop Color="Transparent" Offset="1" />
    </GradientStopCollection>
    <LinearGradientBrush x:Key="MyGlassBrushResource"
        StartPoint="0,0" EndPoint="1,1" Opacity="0.75"
        GradientStops="{StaticResource MyGlassGradientStopsResource}" />
    <!-- Styles and other resources below here. -->
```

这些资源用作插入到按钮模板的 `Grid` 中的矩形的 `Fill`。将以下突出显示的标记添加到模板。

XAML

```
<Setter.Value>
    <ControlTemplate TargetType="{x:Type Button}">
        <Grid Width="{TemplateBinding Width}" Height="{TemplateBinding Height}"
            ClipToBounds="True">

            <!-- Outer Rectangle with rounded corners. -->
            <Rectangle x:Name="outerRectangle" HorizontalAlignment="Stretch"
                VerticalAlignment="Stretch" Stroke="{TemplateBinding Background}"
                RadiusX="20" RadiusY="20" StrokeThickness="5" Fill="Transparent"
            />
```

```

<!-- Inner Rectangle with rounded corners. -->
<Rectangle x:Name="innerRectangle" HorizontalAlignment="Stretch"
    VerticalAlignment="Stretch" Stroke="Transparent"
StrokeThickness="20"
    Fill="{TemplateBinding Background}" RadiusX="20" RadiusY="20" />

<!-- Glass Rectangle -->
<Rectangle x:Name="glassCube" HorizontalAlignment="Stretch"
    VerticalAlignment="Stretch"
    StrokeThickness="2" RadiusX="10" RadiusY="10" Opacity="0"
    Fill="{StaticResource MyGlassBrushResource}"
    RenderTransformOrigin="0.5,0.5">
    <Rectangle.Stroke>
        <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
            <LinearGradientBrush.GradientStops>
                <GradientStop Offset="0.0" Color="LightBlue" />
                <GradientStop Offset="1.0" Color="Gray" />
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Rectangle.Stroke>
    <!-- These transforms have no effect as they are declared here.
The reason the transforms are included is to be targets
for animation (see later). -->
    <Rectangle.RenderTransform>
        <TransformGroup>
            <ScaleTransform />
            <RotateTransform />
        </TransformGroup>
    </Rectangle.RenderTransform>
    <!-- A BevelBitmapEffect is applied to give the button a
"Beveled" look. -->
    <Rectangle.BitmapEffect>
        <BevelBitmapEffect />
    </Rectangle.BitmapEffect>
</Rectangle>

<!-- Present Text of the button. -->
<DockPanel Name="myContentPresenterDockPanel">
    <ContentPresenter x:Name="myContentPresenter" Margin="20"
        Content="{TemplateBinding Content}"
TextBlock.Foreground="Black" />
</DockPanel>
</Grid>
</ControlTemplate>
</Setter.Value>

```

请注意，`x:Name` 属性为“glassCube”的矩形的 `Opacity` 为 0，因此当你运行示例时，你看不到覆盖在顶部的玻璃矩形。这是因为我们稍后会在用户与按钮交互时将触发器添加到模板中。但是，通过将 `Opacity` 值更改为 1 并运行应用程序，可以看到按钮现在的外观。请参阅下图。在继续下一个步骤之前，请将 `Opacity` 改回为 0。



创建按钮交互性

在本部分中，你将创建属性触发器和事件触发器来更改属性值，并运行动画来响应用户操作，例如将鼠标指针移到按钮上并单击。

添加交互性（鼠标悬停、鼠标离开、单击等）的一种简单方法是在模板或样式中定义触发器。若要创建 Trigger，可定义属性“condition”，例如：按钮 `IsMouseOver` 属性值等于 `true`。然后，定义触发器条件为 `true` 时发生的资源库（操作）。

创建按钮交互性

1. **添加模板触发器**：将突出显示的标记添加到模板。

```
XAML

<Setter.Value>
  <ControlTemplate TargetType="{x:Type Button}">
    <Grid Width="{TemplateBinding Width}"
          Height="{TemplateBinding Height}" ClipToBounds="True">

      <!-- Outer Rectangle with rounded corners. -->
      <Rectangle x:Name="outerRectangle" HorizontalAlignment="Stretch"
                 VerticalAlignment="Stretch" Stroke="{TemplateBinding Background}"
                 RadiusX="20" RadiusY="20" StrokeThickness="5" Fill="Transparent"
      />

      <!-- Inner Rectangle with rounded corners. -->
      <Rectangle x:Name="innerRectangle" HorizontalAlignment="Stretch"
                 VerticalAlignment="Stretch" Stroke="Transparent"
                 StrokeThickness="20"
                 Fill="{TemplateBinding Background}" RadiusX="20" RadiusY="20"
      />
```

```

<!-- Glass Rectangle -->
<Rectangle x:Name="glassCube" HorizontalAlignment="Stretch"
    VerticalAlignment="Stretch"
    StrokeThickness="2" RadiusX="10" RadiusY="10" Opacity="0"
    Fill="{StaticResource MyGlassBrushResource}"
    RenderTransformOrigin="0.5,0.5">
    <Rectangle.Stroke>
        <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
            <LinearGradientBrush.GradientStops>
                <GradientStop Offset="0.0" Color="LightBlue" />
                <GradientStop Offset="1.0" Color="Gray" />
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Rectangle.Stroke>

    <!-- These transforms have no effect as they
        are declared here.
        The reason the transforms are included is to be targets
        for animation (see later). -->
    <Rectangle.RenderTransform>
        <TransformGroup>
            <ScaleTransform />
            <RotateTransform />
        </TransformGroup>
    </Rectangle.RenderTransform>

    <!-- A BevelBitmapEffect is applied to give the button a
        "Beveled" look. -->
    <Rectangle.BitmapEffect>
        <BevelBitmapEffect />
    </Rectangle.BitmapEffect>
</Rectangle>

<!-- Present Text of the button. -->
<DockPanel Name="myContentPresenterDockPanel">
    <ContentPresenter x:Name="myContentPresenter" Margin="20"
        Content="{TemplateBinding Content}"
        TextBlock.Foreground="Black" />
</DockPanel>
</Grid>

    <ControlTemplate.Triggers>      <!-- Set action triggers for the
        buttons and define           what the button does in response to those
        triggers. -->      </ControlTemplate.Triggers>
    </ControlTemplate>
</Setter.Value>

```

2. **添加属性触发器**：将突出显示的标记添加到 `ControlTemplate.Triggers` 块：

XAML

```
<ControlTemplate.Triggers>
```

```

<!-- Set properties when mouse pointer is over the button. -->
<Trigger Property="IsMouseOver" Value="True">      <!-- Below are three
property settings that occur when the condition is met (user
mouses over button). -->      <!-- Change the color of the outer
rectangle when user mouses over it. -->      <Setter Property
="Rectangle.Stroke" TargetName="outerRectangle" Value="
{DynamicResource {x:Static SystemColors.HighlightBrushKey}}" />      <!--
- Sets the glass opacity to 1, therefore, the glass "appears"
when user mouses over it. -->      <Setter Property="Rectangle.Opacity"
Value="1" TargetName="glassCube" />      <!-- Makes the text slightly
blurry as though you were looking at it through blurry glass.
-->      <Setter Property="ContentPresenter.BitmapEffect"
TargetName="myContentPresenter">      <Setter.Value>
<BlurBitmapEffect Radius="1" />      </Setter.Value>      </Setter>
</Trigger>

<ControlTemplate.Triggers/>

```

按 F5 运行应用程序，并查看在按钮上运行鼠标指针时的效果。

- 添加焦点触发器**：接下来，添加一些类似的资源库来处理按钮具有焦点的情况（例如，在用户单击它之后）。

XAML

```

<ControlTemplate.Triggers>

<!-- Set properties when mouse pointer is over the button. -->
<Trigger Property="IsMouseOver" Value="True">

    <!-- Below are three property settings that occur when the
        condition is met (user mouses over button). -->
    <!-- Change the color of the outer rectangle when user
        mouses over it. -->
    <Setter Property ="Rectangle.Stroke" TargetName="outerRectangle"
        Value="{DynamicResource {x:Static
SystemColors.HighlightBrushKey}}" />

    <!-- Sets the glass opacity to 1, therefore, the glass
        "appears" when user mouses over it. -->
    <Setter Property="Rectangle.Opacity" Value="1"
        TargetName="glassCube" />

    <!-- Makes the text slightly blurry as though you were
        looking at it through blurry glass. -->
    <Setter Property="ContentPresenter.BitmapEffect"
        TargetName="myContentPresenter">
        <Setter.Value>
            <BlurBitmapEffect Radius="1" />
        </Setter.Value>
    </Setter>
</Trigger>

<!-- Set properties when button has focus. -->      <Trigger

```

```

        Property="IsFocused" Value="true">>      <Setter
        Property="Rectangle.Opacity" Value="1"      TargetName="glassCube" />
        <Setter Property="Rectangle.Stroke" TargetName="outerRectangle"
        Value="{DynamicResource {x:Static SystemColors.HighlightBrushKey}}}" />
        <Setter Property="Rectangle.Opacity" Value="1" TargetName="glassCube"
        />    </Trigger>

    </ControlTemplate.Triggers>

```

按 F5 运行应用程序并单击其中一个按钮。请注意，单击按钮后，按钮保持突出显示状态，因为它仍具有焦点。如果单击另一个按钮，新按钮将获得焦点，而上一个按钮将失去焦点。

4. 为 `MouseEnter` 和 `MouseLeave` 添加动画：接下来为触发器添加一些动画。在 `ControlTemplate.Triggers` 块内的任意位置添加以下标记。

XAML

```

<!-- Animations that start when mouse enters and leaves button. -->
<EventTrigger RoutedEvent="Mouse.MouseEnter">
    <EventTrigger.Actions>
        <BeginStoryboard Name="mouseEnterBeginStoryboard">
            <Storyboard>
                <!-- This animation makes the glass rectangle shrink in the X
                direction. -->
                <DoubleAnimation Storyboard.TargetName="glassCube"
                    Storyboard.TargetProperty=
                    "(Rectangle.RenderTransform).(TransformGroup.Children)[0].
                    (ScaleTransform.ScaleX)"
                    By="-0.1" Duration="0:0:0.5" />
                <!-- This animation makes the glass rectangle shrink in the Y
                direction. -->
                <DoubleAnimation
                    Storyboard.TargetName="glassCube"
                    Storyboard.TargetProperty=
                    "(Rectangle.RenderTransform).(TransformGroup.Children)[0].
                    (ScaleTransform.ScaleY)"
                    By="-0.1" Duration="0:0:0.5" />
            </Storyboard>
        </BeginStoryboard>
    </EventTrigger.Actions>
</EventTrigger>
<EventTrigger RoutedEvent="Mouse.MouseLeave">
    <EventTrigger.Actions>
        <!-- Stopping the storyboard sets all animated properties back to
        default. -->
        <StopStoryboard BeginStoryboardName="mouseEnterBeginStoryboard" />
    </EventTrigger.Actions>
</EventTrigger>

```

当鼠标指针移到按钮上时，玻璃矩形会缩小；当指针离开时，玻璃矩形会恢复到正常大小。

当指针移到按钮上时（引发 [MouseEnter](#) 事件），会触发两个动画。这些动画沿 X 和 Y 轴缩小玻璃矩形。注意 [DoubleAnimation](#) 元素的属性 - [Duration](#) 和 [By](#)。
[Duration](#) 指定动画效果持续半秒，[By](#) 指定玻璃矩形缩小 10%。

第二个事件触发器 ([MouseLeave](#)) 仅用于停止第一个事件触发器。当你停止 [Storyboard](#) 时，所有动画属性都会恢复为默认值。因此，当用户将指针从按钮上移开时，按钮会回到鼠标指针移到按钮上之前的状态。有关动画的详细信息，请参阅 [动画概述](#)。

5. **添加单击按钮时的动画**：最后一步是添加用户单击按钮时的触发器。在 `ControlTemplate.Triggers` 块内的任意位置添加以下标记：

XAML

```
<!-- Animation fires when button is clicked, causing glass to spin. -->
<EventTrigger RoutedEvent="Button.Click">
    <EventTrigger.Actions>
        <BeginStoryboard>
            <Storyboard>
                <DoubleAnimation Storyboard.TargetName="glassCube"
                    Storyboard.TargetProperty=
                        "(Rectangle.RenderTransform).(TransformGroup.Children)[1].
                        (RotateTransform.Angle)"
                    By="360" Duration="0:0:0.5" />
            </Storyboard>
        </BeginStoryboard>
    </EventTrigger.Actions>
</EventTrigger>
```

按 F5 运行应用程序并单击其中一个按钮。单击按钮时，玻璃矩形会旋转。

总结

在本演练中，你执行了以下练习：

- 将 [Style](#) 的目标设置为对象类型 ([Button](#))。
- 使用 [Style](#) 控制整个应用程序中按钮的基本属性。
- 创建渐变等资源，用于 [Style](#) 资源库的属性值。
- 通过将模板应用于按钮来自定义整个应用程序中按钮的外观。

- 自定义按钮的行为（包括动画效果），以响应用户操作（例如 `MouseEnter`、`MouseLeave` 和 `Click` ）。

另请参阅

- [使用 Microsoft Expression Blend 创建按钮](#)
- [样式设置和模板化](#)
- [动画概述](#)
- [使用纯色和渐变进行绘制概述](#)
- [位图效果概述](#)