1 WPF及Silverlight版Toolbar概述

通过**WPF及Silverlight版Toolbar**，您可以在应用程序中创建自定义工具栏并提供个性化导航。**WPF及Silverlight版Toolbar**还提供项（如连接，自定义目录及分隔符），可以使您灵活地将任意控件放置在工具栏上。

1.1 WPF及Silverlight版帮助文档

入门

- 关于 **ComponentOne Studio WPF Edition**安装，授权，技术支持，命名空间和创建附有控件的项目，请参阅 Getting Started with WPF Edition.
- 关于 **ComponentOne Studio Silverlight Edition**安装，授权，技术支持，命名空间和创建附有控件的项目，请参阅 Getting Started with Silverlight Edition.

2 主要特性

WPF及Silverlight版Toolbar 可以使您创建自定义的，功能丰富的应用程序，通过了解它的主要特性您可以充分的使用**WPF及Silverlight版Toolbar**：

- 标签和分组
  通过使用标签和分组来组织相关的指令，例如，分组功能与工具栏按钮类似，office中的字体或者剪切板工具栏可以看做是分组功能的实现，您可以对每个标签和分组进行标记，从而尽可能的描述其中的指令及它所执行的操作。
  更多详细信息请参阅**Toolbar元素 (Section 5)**。
- 带状功能
  C1Toolbar 内置了Microsoft 的带状功能包括折叠，帮助及对话框开启按钮。
- 轻量级ToolbarStrip
  创建一个轻量级的**C1ToolbarStrip** 并可以单独的使用，例如一个简单的场景。
- 样式化按钮和下拉菜单
  C1Toolbar 控件是一个容器且支持任何的UIElements，为了更容易创建工具栏，**C1Toolbar** 包含了不同的类型并内置于**C1ToolbarStrip**中，如按钮、切换按钮、分隔符及下拉菜单控件，同时也可以添加事件处理或者利用指令框架为每个控件添加行为，更多详细信息请参阅**Toolbar元素 (Section 5)**。
- 支持WPF指令框架
  工具栏支持WPF指令框架，可以为标题或者大、中、小的图像资源创建一个单独的指令，并在您的工具栏中显示。
  更多详细信息请参阅**C1Toolbar指令教程（WPF版）。 (Section 9)**
- 支持溢出
  根据在Strip面板上的可用空间，项可以在Strip面板和下拉Overflow面板之间跳转。默认情况下它会自动执行，也可以设置为从不，总是或者按需要。
- 更改方向
  可以选择您工具栏的方向，水平（默认）或者垂直。请多详细信息请传阅**Toolbar布局 (Section 6.1)**.
- 支持ClearStyle 技术
  C1Toolbar 支持ComponentOne's ClearStyle 技术，您只需要使用不同的控件模板就可以轻松的改变控件的颜色，通过设置几个颜色的属性，您就可以快速的制定**C1Toolbar**元素的样式，更多关于ClearStyle技术的相关信息请参阅**ComponentOne ClearStyle技术 (Section 6.4)**。
- Silverlight Toolkit主题
  通过支持内置的Microsoft Silverlight Toolkit主题可以添加样式到您的UI中，包括ExpressionDark, ExpressionLight, WhistlerBlue, RainerOrange, ShinyBlue和BureauBlack。

3 C1Toolbar快速入门

快速入门可以帮助您顺利运行**WPF及Sliverlight版Toolbar**，在快速入门中您需要在Visual Studio中创建一个新的工程，添加**WPF及Silverlight版Toolbar**控件到您的应用程序中，并添加项**C1ToolbarGroup**, **C1ToolbarStrip**, **C1ToolbarButton**和**C1ToolbarToggleButton** 到您的**C1Toolbar**中，请多详细信息请参阅**Toolbar元素 (Section 5)**。

3.1 第一步：添加WPF版Toolbar到您的项目中

创建您的项目并添加一个**C1Toolbar** 控件到您的应用程序中，完成如下步骤：

1. 在Visual Studio中创建一个新的**WPF或Silverlight项目**。
2. 添加一个引用到**C1.WPF及C1.WPF.C1Toolbar** 或者 **C1.Silverlight及C1.Silverlight.C1Toolbar** 配置文件，在解决方案管理器中右键点击**References** 并选择**Add Reference**，在**Add Reference** 对话框中选择浏览标签，浏览**C1.WPF.C1Toolbar.dll及C1.WPF.dll** 或者 **C1.Silverlight.C1Toolbar.dll及C1.Silverlight.dll** 并选择**OK**。
3. 定义系统和**C1.WPF.C1Toolbar** 或 **C1.Silverlight.C1Toolbar** 前缀。

WPF

```
xmlns:System="clr-namespace:System;assembly=mscorlib"
xmlns:c1toolbar="clr-namespace:C1.WPF.Toolbar;assembly=C1.WPF.Toolbar"
```

Silverlight

```
xmlns:System="clr-namespace:System;assembly=mscorlib"
xmlns:c1toolbar="clr-
namespace:C1.Silverlight.C1Toolbar;assembly=C1.Silverlight.C1Toolbar"
```

4. 添加两行到LayoutRoot 网格并设置第一行的**Height**为**Auto**。

XAML

```
<Grid x:Name="LayoutRoot" Background="White">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
</Grid>
```

5. 在页面的第一行中拖入一个**C1Toolbar** 并移除默认属性：**Height="100" HorizontalAlignment="Left" Margin="174,34,0,0"**.，您此时的XAML将会如下：

XAML

```
<Grid.RowDefinitions>
   <RowDefinition Height="Auto"/>
   <RowDefinition Height="*"/>
</Grid.RowDefinitions>
   <c1:C1Toolbar Grid.Row="1" Name="c1Toolbar1">
   </c1:C1Toolbar>
```

3.2 第二步：添加C1ToolbarGroups到C1Toolbargroup

在本步中您将继续在Visual Studio中添加**C1ToolbarGroups**到您的**C1Toolbar**中，此时添加**C1ToolbarButtons**到**C1ToolbarGroups**中。

1. 右键点击**C1Toolbar** 控件并选择**Properties** 来打开它的属性窗口，导航到**ToolbarItems** 属性并点击省略按钮打开**Collection Editor: ToolbarItems**。
2. 点击**Add** 按钮两次添加两个**C1ToolbarGroups** 并点击**OK**，此时我们的XAML将会如下：

XAML

```
<c1:C1Toolbar Name="c1Toolbar1">
    <c1:C1ToolbarGroup />
    <c1:C1ToolbarGroup />
</c1:C1Toolbar>
```

3. 在XAML中，将每一个C1ToolbarGroup设置为: **Clipboard**和**Font**，此时您的XAML 将会如下：

XAML

```
<c1:C1Toolbar Grid.Row="1" Name="c1Toolbar1">
   <c1:C1ToolbarGroup Header="Clipboard"/>
   <c1:C1ToolbarGroup Header="Font"/>
</c1:C1Toolbar>
```

4. 选择**Clipboard C1ToolbarGroup** 并点击省略按钮，打开**Items** 集合编辑器。
5. 从**Select item** 下拉菜单中，点击**Add** 三次添加三个**C1ToolbarButtons** 到集合中。
6. 选择第一个**C1ToolbarButton** 并在它的属性窗口中扩展**Other** 节点，设置**LabelTitle** 属性为**Paste**，并分别设置其他两个为**Cut**和**Copy**。
7. 这一步是可选择的，您可以设置**LargeImageSource** 和/或 **SmallImageSource** 属性到您项目中的资源，或者添加新的图像。
8. 点击**OK** 来关闭**Items Collection** 编辑器，您的XAML将会如下：

XAML

```
<c1:C1Toolbar Name="c1Toolbar1">
    <c1:C1ToolbarGroup Header="Clipboard">
            <c1:C1ToolbarButton LabelTitle="Paste"
LargeImageSource="/ToolbarQuickstart;component/Images/Paste.png" />
            <c1:C1ToolbarButton LabelTitle="Cut"
SmallImageSource="/ToolbarQuickstart;component/Images/Cut.png" />
            <c1:C1ToolbarButton LabelTitle="Copy"
SmallImageSource="/ToolbarQuickstart;component/Images/Copy.png" />
        </c1:C1ToolbarGroup>
        <c1:C1ToolbarGroup Header="Font"/>
</c1:C1Toolbar>
```

3.3 第三步：添加一个C1ToolbarStrip和C1ToolbarToggleButtons

在这一步中您将继续在Visual Studio中添加一个**C1ToolbarStrip** 到您的'Font'**C1ToolbarGroup**，此时您将添加**C1ToolbarToggleButtons** 到**C1ToolbarGroup**。

1. 选择**Font C1ToolbarGroup** 并在XAML中添加**C1ToolbarStrip**。

XAML
```
<c1:C1ToolbarGroup Header="Font">
    <c1:C1ToolbarStrip />
</c1:C1ToolbarGroup>
```

其余的XAML 将如下：

XAML
```
<c1:C1Toolbar Grid.Row="1" Name="c1Toolbar1"
    <c1:C1ToolbarGroup Header="Clipboard">
        <c1:C1ToolbarButton LabelTitle="Paste" />
        <c1:C1ToolbarButton LabelTitle="Cut" />
        <c1:C1ToolbarButton LabelTitle="Copy" />
    </c1:C1ToolbarGroup>
    <c1:C1ToolbarGroup Header="Font">
    <c1:C1ToolbarStrip />
    </c1:C1ToolbarGroup>
</c1:C1Toolbar>
```

2. 选择**C1ToolbarStrip** 并打开它的**Items** 集合编辑器。
3. 从**Select item**下拉菜单中选择**C1ToolbarToggleButton** 并点击**Add** 三次添加三个**C1ToolbarToggleButtons**，设置每个**LabelTitle**属性为：加粗，斜体，下划线，点**OK** 关闭**Items** 集合编辑器。

XAML
```
<c1:C1ToolbarGroup Header="Font">
    <c1:C1ToolbarStrip>
        <c1:C1ToolbarToggleButton LabelTitle="Bold" />
        <c1:C1ToolbarToggleButton LabelTitle="Italic" />
        <c1:C1ToolbarToggleButton LabelTitle="Underline" />
    </c1:C1ToolbarStrip>
</c1:C1ToolbarGroup>
```

恭喜！您现在已经使用C1Toolbargroup创建了一个工具栏。

更多主题:

- **为ToolbarButton点击事件添加后台逻辑 (Section 8.3)** -该主题展示了如何使用按钮点击事件添加后台逻辑。
- **C1Toolbar指令教程（WPF版） (Section 9)** -该教程演示了如何在WPF应用程序中使用**C1Toolbar指令**。
- **C1Toolbar指令教程（Silverlight版） (Section 10)** - 该教程演示了如何在Silverlight应用程序中使用**C1Toolbar指令**。
- **C1ToolbarGroup中按钮大小和文本的位置 (Section 6.2)** -该主题展示了如何使用**C1ToolbarGroup**的**GroupSizeDefinitions**。
- **Toolbar标签项 (Section 5.6)** -该主题展示了如何添加**C1ToolbarTabItem**控件。

4 XAML快速索引

本节将给出一个例子，展示如何仅通过XAML代码使用**C1Toolbar**控件。

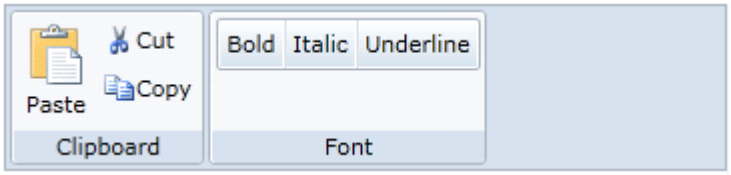
4.1 例：添加项到C1Toolbar中

以下XAML 代码将为您展示如何添加一个**C1ToolbarGroup**, **C1ToolbarStrip**和**C1ToolbarButton**到**C1Toolbar**控件中：

XAML

```
<c1:C1Toolbar Name="c1Toolbar1">
 <c1:C1ToolbarTabControl>
  <c1:C1ToolbarTabItem Header="Home">
   <c1:C1ToolbarGroup Header="Clipboard">
     <c1:C1ToolbarButton LabelTitle="Paste" LargeImageSource="/Images/Paste.png" />
     <c1:C1ToolbarButton LabelTitle="Cut" SmallImageSource="/Images/Cut.png" />
     <c1:C1ToolbarButton LabelTitle="Copy" SmallImageSource="/Images/Copy.png" />
   </c1:C1ToolbarGroup>
   <c1:C1ToolbarGroup Header="Font">
    <c1:C1ToolbarStrip>
       <c1:C1ToolbarToggleButton LabelTitle="Bold" />
       <c1:C1ToolbarToggleButton LabelTitle="Italic" />
       <c1:C1ToolbarToggleButton LabelTitle="Underline" />
    </c1:C1ToolbarStrip>
   </c1:C1ToolbarGroup>
  </c1:C1ToolbarTabItem>
 </c1:C1ToolbarTabControl>
</c1:C1Toolbar>
```

下图展示了通过添加以上XAML代码所显示的C1Toolbar图像：



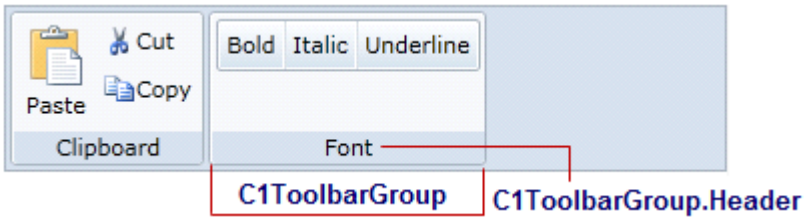
5 Toolbar元素

C1Toolbar 是一个容器控件，它可以包含其他的控件，如按钮，复选按钮，文本框，下拉菜单，拆分按钮和分隔符，**C1ToolBar** 提供七种不同的对象类型以支持各种各样类型的控件：
C1ToolbarButton, C1ToolbarDropDown, C1ToolbarGroup, C1ToolbarSplitButton, C1ToolbarStrip, C1ToolbarTabItem, C1ToolbarToggleButton。

5.1 Toolbar分组

C1ToolBarGroup 对象定义了一组工具栏元素，**C1ToolbarGroup** 可以包含以下工具栏控件：**C1ToolbarButton**, **C1ToolbarToggleButton**, **C1ToolbarDropDown**和 **C1ToolbarSplitButton**。

当您添加工具栏项到**C1ToolbarGroup**，则子项会被分组到矩形框中，如下图：



C1ToolbarGroups典型用于用户一组相互独立的功能，这就是说，一个组按钮仅表示一个功能且只能在某个时刻启动。

在**C1ToolbarGroup** 中控件(**C1ToolbarButtons**和**C1ToolbarToggleButtons**)的图像可以通过使用**GroupSizeDefinitions**类制定尺寸的大，中或小。

GroupSizeDefinition 定义了控件尺寸的大中小，以及**C1ToolbarGroup**中按钮文档的位置，如果设置为“large”，此时在组中的所有控件(**C1ToolbarButtons** 和 **C1ToolbarToggleButtons**)可以从**LargeImageSource**属性获取图像资源及将LabelTitle属性中的文本值置于图像下方，如果设置为“meidum”，此时在组中的所有控件(**C1ToolbarButtons** 和 **C1ToolbarToggleButtons**)可以从**SmallImageSource**属性获取图像资源及将LabelTitle属性中的文本值置于图像右边，如果设置为“small”，此时在组中的所有控件(**C1ToolbarButtons** 和 **C1ToolbarToggleButtons**)可以从**SmallImageSource**属性获取图像资源且不显示文本信息。

例： 带有C1ToolbarStrip 的C1ToolbarGroup，C1ToolbarStrip 包含C1ToolbarButtons

XAML

```
</c1:C1ToolbarGroup>
   <c1:C1ToolbarGroup Header="Font">
    <c1:C1ToolbarStrip>
       <c1:C1ToolbarToggleButton LabelTitle="Bold" />
       <c1:C1ToolbarToggleButton LabelTitle="Italic" />
       <c1:C1ToolbarToggleButton LabelTitle="Underline" />
    </c1:C1ToolbarStrip>
   </c1:C1ToolbarGroup>
```

C1ToolBarGroup 包含以下属性：

Property	Definition
FocusBrush	获取或设置一个指定集合索引的项。

GroupSizeDefinitions	获取或设控件中选定项的模块。
Header	获取或设置工具栏分组的标题。
MouseOverBrush	获取或设置画笔用以高亮控件，当鼠标悬置于控件上方时。
PressedBrush	获取或设置画笔用以绘制一个按钮，当它被按压时。
ShowDialogLauncher	获取或设置对话框启动的可见性。

5.2 Toolbar按钮

C1ToolBarButton 对象定义了一个工具栏按钮，可以包含本文，图像或者文本和图像，通过设置**C1ToolBarButton.LabelTitle** 属性来更改文本，以及**C1ToolBarButton**对象的**LargeImageSource**或**SmallImageSource** 来更改图像。

按钮可以使用**C1ToolbarItems** 集合编辑器在运行时添加，也可以使用**C1ToolBarItemCollection** 集合中的**C1ToolBarItemCollection.Add**或者**C1ToolBarItemCollection.Remove**方法编码实现，又或者通过XAML编码实现。

C1ToolBarButton 包含以下特有属性：

Property	**Definition**
LabelTitle	获取或设置控件标签的标题。
LargeImageSource	获取或设置控件的大图像资源。
MouseOverBrush	获取或设置画笔用以高亮控件，当鼠标悬置于控件上方时。
PressedBrush	获取或设置画笔用以绘制一个按钮，当它被按压时。
SmallImageSource	获取或设置控件的小图像资源。

5.3 Toolbar下拉菜单

C1ToolbarDropDown 控件代表在**C1ToolbarStrip**中的一个下拉按钮，当它被点击时会弹出**Content** 属性面板或者通过设置**Menu** 属性的上下文菜单，一个下拉按钮为用户提供一列选项，当您点击**C1ToolBarDropDown** 按钮时，**Click** 事件将被触发，下拉列表同时会出现。

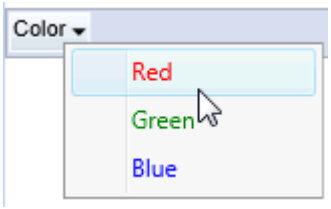
例 1: 带有弹出堆叠面板的下拉菜单，面板中包含按钮。

XAML

```
<c1:C1ToolbarDropDown Padding="2" Header="Color">
   <c1:C1ToolbarDropDown.Content>
```

```
      <StackPanel Margin="2" Orientation="Horizontal">
        <Button Margin="2" Content="Red" Foreground="Red" />
        <Button Margin="2" Content="Green" Foreground="Green" />
        <Button Margin="2" Content="Blue" Foreground="Blue" />
      </StackPanel>
    </c1:C1ToolbarDropDown.Content>
</c1:C1ToolbarDropDown>
```

例 2: 带有弹出菜单的下拉菜单



XAML

```
<c1:C1Toolbar Name="c1Toolbar1" Margin="0,127,0,160">
   <c1:C1ToolbarDropDown Padding="2" Header="Color">
     <c1:C1ToolbarDropDown.ContextMenu>
        <ContextMenu>
          <MenuItem Foreground="Red" Header="Red" IsCheckable="True" />
          <MenuItem Foreground="Green" Header="Green" IsCheckable="True" />
          <MenuItem Foreground="Blue" Header="Blue" IsCheckable="True" />
         </ContextMenu>
     </c1:C1ToolbarDropDown.ContextMenu>
   </c1:C1ToolbarDropDown>
</c1:C1Toolbar></c1:C1ToolbarDropDown>
```

C1ToolBarDropDown 包含下述特有属性:

Property	Definition
ContentBackground	获取或设置目录的背景
Menu	获取或设置当控件被点击时显示的上下文菜单。
MouseOverBrush	获取或设置画笔用以高亮控件，当鼠标悬置于控件上方时。
PressedBrush	获取或设置画笔用以绘制一个按钮，当它被按压时。

5.4 Toolbar拆分按钮

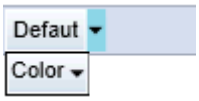
C1ToolbarSplitButton 控件代表在**C1ToobarStrip**中的一个下拉菜单的拆分按钮。

它类似于**C1ToolbarDropDown**，但它包含两个可点击的区域：按钮区域和向下箭头。

当点击下方矩形框的最右侧部分，它将显示一个带有**Content** 属性的弹出面板或者使用**ContextMenu** 属性的上下文菜

单，当点击按钮左侧部分则会触发标准按钮的**Click** 事件，当弹出可选择项时通常**Click** 事件用以执行默认或者最近一次行为。

当您将光标悬停在如下图像的按钮上，垂直实线分割的图像将会显示如下：



例: 带有弹出菜单的拆分按钮

XAML

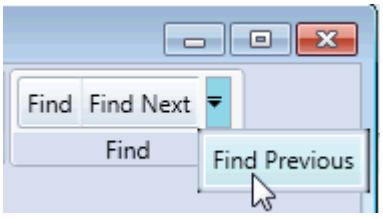
```
<c1:C1ToolbarSplitButton Padding="2" Header="Defaut"
   Click="SetDefaultStyle">
   <c1:C1ToolbarDropDown.Menu>
     <c1:C1ContextMenu>
       <c1:C1MenuItem Header="Heading 1" FontSize="14" />
       <c1:C1MenuItem Header="Heading 2" FontSize="12" />
       <c1:C1MenuItem Header="Title" FontWeight="Bold" />
       <c1:C1MenuItem Header="Subtitle" FontWeight="SemiBold"
         FontStyle="Italic"  />
       <c1:C1MenuItem Header="Quote" FontStyle="Italic" />
     </c1:C1ContextMenu>
   </c1:C1ToolbarDropDown.Menu>
</c1:C1ToolbarSplitButton>
```

5.5 Toolbar带

表示工具栏控件或者其他WPF版C1控件的一个容器，**C1ToolbarStrip** 可以包含**C1ToolbarButton**, **C1ToolbarToggleButton**, **C1ToolbarDropDown**, **C1ToolbarSplitButton**, **C1Separator**, **C1ComboBox** 和**C1TextBox** 控件。

C1ToolbarStrip 会根据在带状面板中的可用空间大小实现溢出操作，所有的项会在带状面板和下拉溢出面板之间跳转，默认情况下它会自动执行，但也可以设置为从不，总是，或者按需要。

下图演示了**C1ToolbarStrip** 所提供的溢出功能：



C1ToolbarStrip 可以使用XAML或者编程添加。

例: 带有切换按钮的Toolbar带

XAML

```
<c1:C1ToolbarGroup Header="Font">
    <c1:C1ToolbarStrip>
```

```
        <c1:C1ToolbarToggleButton LabelTitle="Bold" />
        <c1:C1ToolbarToggleButton LabelTitle="Italic" />
        <c1:C1ToolbarToggleButton LabelTitle="Underline" />
    </c1:C1ToolbarStrip>
</c1:C1ToolbarGroup>
```

C1ToolBarStrip 包含以下特有属性：

Property	Definition
ButtonBackground	获取或设置画笔，用以配置控件内按钮的背景。
ButtonForeground	获取或设置画笔，用以配置控件内按钮的前景。
FocusBrush	获取或设置画笔，用以控件聚焦时的高亮。
MouseOverBrush	获取或设置画笔用以高亮控件，当鼠标悬置于控件上方时。
Orientation	获取或设置工具栏带的方向。
Overflow	获取或设置一个值，当项不适合可用空间时，将如何处理。
OverflowMenuItems	获取包含溢出菜单元素的集合。
OverflowPanel	获取或设置溢出面板的模板。
PressedBrush	获取或设置画笔用以绘制一个按钮，当它被按压时。

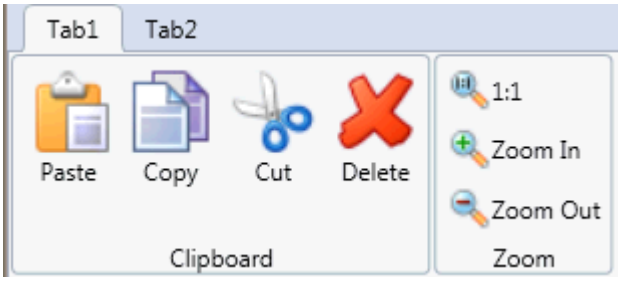
5.6 Toolbar标签项

C1ToolbarTablItem 代表在C1Toolbar中的一个标签项，当您需要在一个很小窗口空间中显示大量的信息时，**C1ToolbarTabItems** 会十分有效。

C1ToolbarTabItem 将控件分割成一个单独的页，且在同一时间只能显示一个标签。

C1ToolbarTabItem 可以包含一个**C1ToolbarGroups**集合，它可以包含**C1ToolbarButtons**, **C1ToolbarToggleButtons**, **C1ToolbarDropDowns**和**C1ToolbarSplitButtons**。

下图显示了两个**C1ToolbarTabItems**:



C1ToolbarTabItem 可以通过**ToolbarItems**集合编辑器，XAML或者编程的方式添加到**C1Toolbar** 中。

为了使用设计器添加一个**C1ToolbarTabItem**，需要完成以下步骤：

1. 添加一个 **C1Toolbar** 控件到您的页面中。

2. 选择**C1Toolbar** 控件并点击省略按钮，打开**C1Toolbar** 属性窗口中**ToolbarItems** 属性，此时出现**ToolbarItems**的集合编辑器。
3. 从下拉菜单中选择**C1ToolbarTabItem** 并点击**Add**，此时**C1ToolbarTabItem**被添加到**C1Toolbar** 控件中。
4. 设置**Header** 属性为**Tab 1**.

为了添加**C1ToolbarTabItem**，请使用如下XAML代码:

XAML

```
<c1:C1Toolbar Grid.Row="1" Name="c1Toolbar1">
            <c1:C1ToolbarTabItem Header="Tab 1">
                <c1:C1ToolbarTabItem.Content>
                    <c1:C1ToolbarPanel />
                </c1:C1ToolbarTabItem.Content>
            </c1:C1ToolbarTabItem>
</c1:C1Toolbar>
```

C1ToolBarGroups 可以通过**Groups** 集合编辑器，XAML或者使用**C1ToolbarTabItem.Groups**属性的编程方式添加到**C1ToolbarTabItem**中。

为了使用设计器将**C1ToolbarGroup** 添加到**C1ToolbarTabItem**，需要完成以下步骤：

1. 添加一个 **C1Toolbar** 控件到您的页面中。
2. 右键点击**C1ToolbarTabItem**并选择**Properties**，点击**ellipsis**按钮并打开在**C1ToolbarTabItem** 属性窗口中的**Groups**属性，此时显示**Groups** 集合编辑器。
3. 点击**Add**将一个**C1ToolbarGroup** 添加到**C1ToolbarTabItem**，**C1ToolbarGroup** 被添加到**C1ToolbarTabItem**。
4. 设置**Header** 属性为 **Tab 1**。

为了添加**C1ToolbarGroup**到**C1ToolbarTabItem** 中，请使用XAML代码：

XAML

```
<c1:C1Toolbar Grid.Row="1" Name="c1Toolbar1">
            <c1:C1ToolbarTabItem Header="Tab 1">
                <c1:C1ToolbarTabItem.Content>
                    <c1:C1ToolbarPanel />
                </c1:C1ToolbarTabItem.Content>
                <c1:C1ToolbarGroup/>
            </c1:C1ToolbarTabItem>
</c1:C1Toolbar>
```

C1ToolBarTabItem 包含以下特有属性：

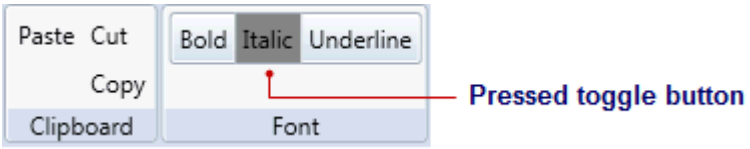
Property	Definition
Groups	获取工具栏分组集合。

5.7 Toolbar切换按钮

C1ToolbarToggleButton 代表在**C1ToolbarToolStrip**中的一个切换按钮，这是一个状态按钮可以使用户在开启和关闭状态之间切换，当点击切换按钮时，它将会保持激活或者按压状态，直到它再次被点击。

切换按钮同指令按钮一样，不但可以显示图像或者文本，而且当它被按压或者激活时，会显示为开启状态。为了指定开启状态，您可以通过**C1ToolbarToggleButton.PressedBrush** 属性设置颜色。

下图演示了三个**C1ToolbarToggleButtons**，其中**Italic**切换按钮被按压，第二个切换按钮**Italic**的**PressedBrush**属性被设置为为**Gray**。



C1ToolbarToggleButtons 可以通过**Items**集合编辑器，编写代码或者XAML添加到**C1ToolbarStrip**中。

例: **三个C1TolbarToggleButtons，其中一个处于开启状态。**

XAML

```
<c1:C1Toolbar Grid.Row="1" Name="c1Toolbar1">
  <c1:C1ToolbarGroup Header="Clipboard">
    <c1:C1ToolbarButton LabelTitle="Paste" />
    <c1:C1ToolbarButton LabelTitle="Cut" />
    <c1:C1ToolbarButton LabelTitle="Copy" />
  </c1:C1ToolbarGroup>
  <c1:C1ToolbarGroup Header="Font">
  <c1:C1ToolbarStrip>
     <c1:C1ToolbarToggleButton LabelTitle="Bold" />
     <c1:C1ToolbarToggleButton PressedBrush ="Gray" LabelTitle="Italic" />
     <c1:C1ToolbarToggleButton LabelTitle="Underline" />
     </c1:C1ToolbarStrip>
  </c1:C1ToolbarGroup>
</c1:C1Toolbar>
```

C1ToolBarToggleButton包含下列属性：

Property	Definition
GroupName	获取或设置互斥的C1ToolbarToggleButton控件的名称。
LabelTitle	获取或设置控件标签的标题。
LargeImageSource	获取或设置控件的大图像资源。
MouseOverBrush	获取或设置画笔用以高亮控件，当鼠标悬置于控件上方时。

PressedBrush	获取或设置画笔用以绘制一个按钮，当它被按压时。
SmallImageSource	获取或设置控件的大图像资源。

6 C1Toolbar布局与外观

以下主题将详细介绍如何自定义**C1Toolbar** 控件的布局和外观，您可以使用内置的布局选项在面板中配置您的布局，如网格或者画布。

您可以通过使用堆叠面板自定义您工具栏项的外观，在堆叠面板中您可以决定您工具栏项的方向，此外您还可以添加图像和文本，以及更改文本的对齐，字体的样式和颜色。

XAML

```
<c1tb:C1Toolbar>
 <c1tb:C1ToolbarGroup Header="Clipboard">
   <c1tb:C1ToolbarStrip Padding="0">
     <c1tb:C1ToolbarButton Width="60" HorizontalContentAlignment="Left">
        <StackPanel Orientation="Horizontal" >
         <Image Grid.Column="1" Source="Resources/cut.png" Margin="7 0 7 0"/>
         <TextBlock Text="Cut" Foreground= "DarkOrange" VerticalAlignment="Center"
TextAlignment="Center" />
           </StackPanel>
    </c1tb:C1ToolbarButton>
   </c1tb:C1ToolbarStrip>
   <c1tb:C1ToolbarStrip Padding="0">
      <c1tb:C1ToolbarButton Width="60" HorizontalContentAlignment="Left" >
        <StackPanel Orientation="Horizontal">
      <Image Source="Resources/copy.png"  Margin="4 0 4 0"/>
   <TextBlock Text="Copy" VerticalAlignment="Center" TextAlignment="Center"/>
        </StackPanel>
      </c1tb:C1ToolbarButton>
    </c1tb:C1ToolbarStrip>
  <c1tb:C1ToolbarStrip Padding="0">
     <c1tb:C1ToolbarButton Width="60" HorizontalContentAlignment="Left">
          <StackPanel Orientation="Horizontal">
            <Image Source="Resources/paste.png"  Margin="4 0 4 0"/>
           <TextBlock Text="Paste" VerticalAlignment="Center"
TextAlignment="Center"/>
        </StackPanel>
      </c1tb:C1ToolbarButton>
    </c1tb:C1ToolbarStrip>
  </c1tb:C1ToolbarGroup>
</c1tb:C1Toolbar>
```

6.1 Toolbar布局

当您使用**C1ToolbarStrip**时可以选择您工具栏的方向，水平（默认）或者垂直。

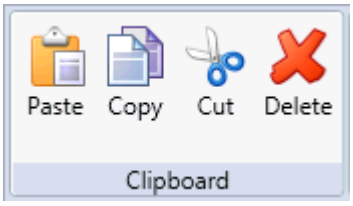
在**C1Toolbar**控件中的带状面板可以通过设置**C1ToolbarStrip.Orientation** 属性水平或者垂直呈现，在C1Toolbar中独立子元素的布局也可以使用**StackPanel.Orientation** 属性来控制。

6.2

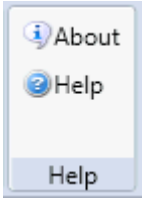
您可以使用**GroupSizeDefinition**对**C1ToolbarGroup**中所有的**C1ToolbarButtons**和**C1ToolbarToggleButtons** 定义统一的尺寸（小，中，或者大）。

GroupSizeDefinition 定义了按钮的大中小尺寸以及在**C1ToolbarGroup**中按钮文本的位置，如果设置为“large”，此时在组中的所有按钮可以从**LargeImageSource**属性获取图像资源及将LabelTitle属性中的文本值置于图像下方，如果设置为“meidum”，此时在组中的所有按钮可以从**SmallImageSource**属性获取图像资源及将LabelTitle属性中的文本值置于图像右边，如果设置为“small”，此时在组中的所有按钮可以从**SmallImageSource**属性获取图像资源且不显示文本信息。

下图使用**LargeImageResource** 属性的按钮及**LabelTitle** 属性的文本在图片下方显示：



下图使用**SmallImageResource** 属性的按钮及**LabelTitle** 属性的文本在图片右方显示：



下图仅显示使用**SmallImageResource** 属性的按钮：

6.3 C1Toolbar模板

WPF及Silverlight版控件的一个优势在于控件具有完全自定义的“无外观”用户界面，例如您在为WPF或Silverlight应用设计用户界面（UI）时，可以通过**WPF及Silverlight版Toolbar**来实现适用于数据管理的UI。可扩展标记语言（XAML;发音为“Zammel”）是一种基于XML的声明式语言，它提供了一种简易的方式可以不用编写代码就可以完成您的UI设计。

可访问模板

您可以访问Microsoft Expression Blend获取模板，选择**C1Toolbar**控件并在对象菜单中选择**编辑模板**，通过选择编辑拷贝来创建当前模板可编辑的一个拷贝，或者选择创建一个新的**空白模板**。

一旦您创建了一个新的模板，该模板将会出现在**Objects and Timeline** 窗口中，注意您可以使用**Template** 属性自定义模板。

> **注意：**如果您通过菜单方式创建了一个新的模板，模板将会自动的连接模板的属性，但如果您用XAML手动创建了一个模板，您就必须为您创建的模板提供适宜的模板属性。

附加模板

除了默认模板外，**C1Toolbar** 控件还包括几个附加模板，这些模板可以通过Microsoft Expression Blend 来访问–在Blend中选择**C1Toolbar** 控件以及在菜单中选择 **Edit Additional Templates**，选择一个模板并选择**Create Empty**。

6.4 ComponentOne ClearStyle技术

ComponentOne ClearStyle技术是一项新颖，快速且简捷的方法，它为Silverlight及WPF版控件提供样式支持，ClearStyle可以帮助您避免处理大量繁琐的XAML模板和样式资源，可以为控件创建一个自定义样式。

目前，如果您需要在所有标准的WPF及Silverlight控件上添加一个主题，就必须要创建一个样式资源模板，然而这个过程在Microsoft Visual Studio上完成是困难的，这就是为什么Microsoft要引入Expression Blend来简化此任务，但是对于那些不熟悉Blend平台或者没有时间学习的开发人员来说，两个平台的转换对于他们来说是一项挑战，此外您还需要一名设计人员，然而您的设计人员和开发人员需要共享XAML文件，这无疑也增加了开发的复杂性。

因此我们将ClearStyle技术引入，想以尽可能直观的方式引入到Visual Studio开发中。在大多数情况下，您可能只需要对应用中的控件做出简单的改变，而这个过程也需要尽可能简化。例如，你仅想改变数据网格的行的颜色，这里就只需要简单地设置一项属性，而不是创建一个完整而复杂的模板对颜色做出改变。

6.4.1 如何使用ClearStyle

控件样式的每个关键部分都应该是可以直接操作的，如一个简单的颜色属性，这就导致需要对每个控件制定一系列独特的样式属性。例如，一个Gauge有**PointerFill**和**PointerStroke**两种属性，而一个DataGrid具有**SelectedBrush**和**MouseOverBrush**两种行特性。

比方说，在您的窗体里有一个控件不支持ClearStyle，您可以通过ClearStyle创建XAML资源并以此匹配您窗体中的其他其他控件（如抓取确切的颜色），或者您想要利用ClearStyle覆盖样式的部分内容（如自定义的滚动条），这都可以通过ClearStyle实现，因为ClearStyle是可扩展并可以覆盖您需要的样式区域。

ClearStyle旨在成为一个快速、便捷的样式改变解决方案，但您依然可以采取老式的ComponentOne控件样式获取需方法，而且ClearStyle不会干预那些不常用的属性，那些属性通常需要完整的自定义设计。

6.4.2 Toolbar ClearStyle属性

C1Toolbar 支持ComponentOne's ClearStyle技术，可以帮助您不用改变控件的模板就可以轻松的对控件的颜色做出更改。通过设置几个颜色的属性，您就可以快速的设置**C1Toolbar**元素的样式，**C1Toolbar** 支持的属性如下表：

Property	Description
Background	获取或设置用以填充**C1Toolbar**的背景。
MouseOverBrush	获取或设置画笔用以高亮控件，当鼠标悬置于控件上方时。
PressedBrush	获取或设置画笔用以绘制一个按钮，当它被按压时。
FocusBrush	获取或设置画笔，用以控件聚焦时的高亮。
Header (仅适用于C1ToolbarGroup)	获取或设置画笔用以绘制标题。

7 C1Toolbar示例

请注意，本ComponentOne软件工具附有不同的示例工程或者demo，您可以利用包含在ComponentOne Studio中的开发工具来完成。

您可以参考下列路径中的产品示例：

Documents\ComponentOne Samples\WPF

或

Documents\ComponentOne Samples\Silverlight

以下ControlExplorer中的页面将详细介绍**C1Toolbar** 控件：

C# Samples

Sample	**Description**
Toolstrip	显示轻量级**C1ToolbarStrip**控件以支持水平和垂直方向，以及溢出处理。
Toolbar	代表**C1Toolbar**控件，支持分组、分隔符及不同的按钮类型。
Commands (仅适用于WPF)	显示如何使用**C1Toolbar** 指令，以及不同的工具栏按钮尺寸。

8 C1Toolbar任务帮助

如果您熟悉Visual Studio编程且了解在一般情况下如何使用**C1Toolbar**，请直接参考任务帮助，如果您是初次使用**WPF及Silverlight版Toolbar**产品，请先参阅C1Toolbar快速入门。

在使用**WPF及Silverlight版Toolbar**产品时，每个任务帮助主题都提供一个特定任务的解决方案。

在您参阅任务帮助主题前，请确定您已经创建了一个新的WPF或Silverlight项目。

8.1 调整工具栏按钮

下述XAML将展示如何在分组面板中央对齐工具栏按钮:

XAML

```
<c1:C1ToolbarGroup Header="Group" >

  <c1:C1ToolbarGroup.ItemsPanel>

   <ItemsPanelTemplate>

      <c1:C1ToolbarGroupPanel HorizontalAlignment="Center" />

    </ItemsPanelTemplate>

  </c1:C1ToolbarGroup.ItemsPanel>
```

8.2 添加一个图像到Toolbar按钮

您可以使用**LargeImageSource** 或 **SmallImageSource** 属性为您的**C1ToolbarButtons** 添加大或者小图像。

使用设计器

1. 在您的页面中右键点击**C1ToolbarButton** 控件，并选择**Properties**。
2. 定位**LargelImageSource** 属性并点击**ellipsis** 按钮，此时显示**Choose Image** 对话框。
3. 点击**Add** 按钮并选择您想要加入的图像，并点击**OK**。

8.3 为ToolbarButton点击事件添加后台逻辑

本主题展示了如何通过XAML为C1ToolbarButton点击事件添加后台逻辑，此时可以在代码编辑器中添加方法代码。

1. 在XAML编辑器中添加一个**C1ToolbarButton**到**C1Toolbar**元素，并设置**LabelTitle** 属性为**Search**。

   XAML

   ```
   <c1:C1Toolbar Grid.Row="1" Height="100" Name="c1Toolbar1">
      <c1:C1ToolbarButton LabelTitle="Search"
   </c1:C1Toolbar>
   ```

2. 在XAML编辑器中添加一个名为**Click**的属性到**C1ToolbarButton** 元素中，将它的值设置为**New Click**，这将为您

在代码中提供事件处理程序的名称，同时也为**C1ToolbarButton** 元素提供唯一名称并设置它的值为：Search。

XAML

```
<c1:C1Toolbar Grid.Row="1" Height="100" Name="c1Toolbar1">
    <c1:C1ToolbarButton LabelTitle="Search" Click="New_Click" Name="Search"/>
</c1:C1Toolbar>
```

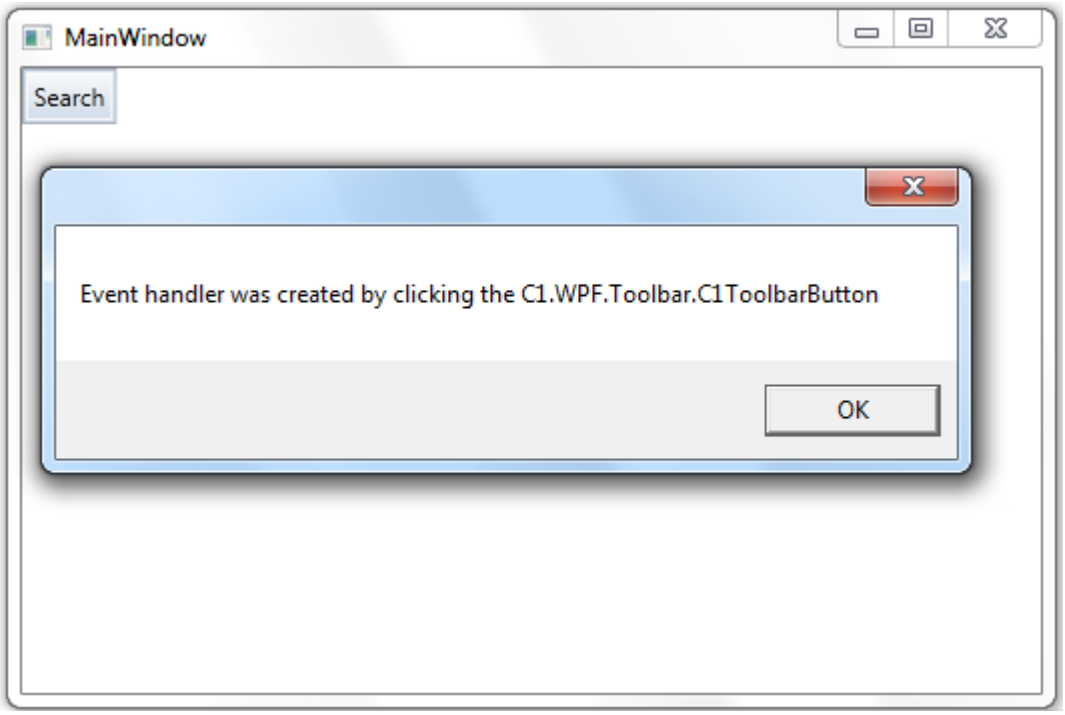
3. 右键点击设计器并点击**View Code**。
4. 添加以下事件处理到Window1类中，这段代码无论您何时点击按钮都会显示一条信息。

C#

```
private void New_Click(object sender, RoutedEventArgs e)
{
  MessageBox.Show("Event handler was created by clicking the " + Search);
}
```

运行应用程序并观察：

工具栏按钮的名称将在消息框中显示：



9 C1Toolbar指令教程（WPF版）

WPF版Toolbar 支持指令框架，在WPF中使用指令有几个目的，其最主要的目的在于分离对象来调用逻辑指令，指令可以降低原始界面与行为处理之间的耦合度。

以下将演示如何在WPF应用程序中使用**C1Toolbar**指令。

9.1 第一部分：使用指令库

WPF 提供了预定义指令库如下：

- 应用指令(剪切，复制，粘贴)
- 导航指令(前进，后退)
- 媒体指令(播放，停止，暂停)
- 编辑指令
- 控件指令

许多WPF控件都内置这些支持的指令，因此您可以不用编写任何代码实现它们。

以下步骤介绍了如何使用**C1Toolbar**实现常用的剪切板功能。

1. 打开或创建一个新的WPF应用程序。
2. 添加两个**RowDefinitions** 到默认的**Grid**元素，为第一行设置为自动高度，如下:

XAML

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

3. 添加一个**C1Toolbar** 以填充第一行。
4. 粘贴以下XAML并使用标签，分组和按钮来填充**C1Toolbar**:

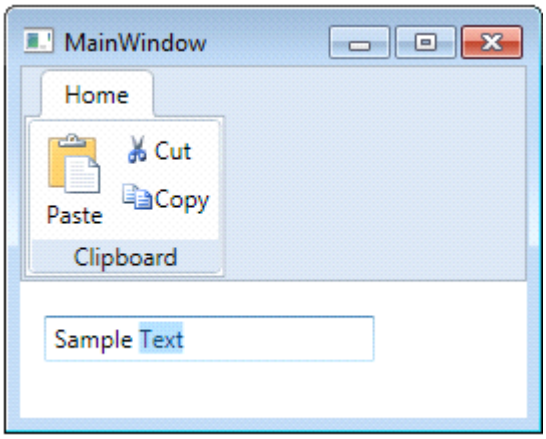
XAML

```
<c1:C1Toolbar Name="c1Toolbar1" FocusManager.IsFocusScope="True">
    <c1:C1ToolbarTabControl>
        <c1:C1ToolbarTabItem Header="Home">
            <c1:C1ToolbarGroup Header="Clipboard">
                <c1:C1ToolbarButton LabelTitle="Paste"
Command="ApplicationCommands.Paste" LargeImageSource="/Resources/paste.png" />
                <c1:C1ToolbarButton LabelTitle="Cut"
Command="ApplicationCommands.Cut" SmallImageSource="/Resources/cut.png" />
                <c1:C1ToolbarButton LabelTitle="Copy"
Command="ApplicationCommands.Copy" SmallImageSource="/Resources/copy.png" />
            </c1:C1ToolbarGroup>
        </c1:C1ToolbarTabItem>
    </c1:C1ToolbarTabControl>
</c1:C1Toolbar>
```

该标记在C1Toolbar中创建一个包含单个分组的标签，**Clipboard** 分组包含三个**C1ToolbarButtons** 并为每一个从WPF应用指令库中分配一个常用的剪切板指令，注意C1Toolbar的**IsFocusScope** 属性被设置为**True** ，这可以使**C1Toolbar** 跟踪其范围内所有聚焦的元素，小图和大图都可以指定给**C1ToolbarButtons**，尽管这些并不是必

要的。

5. 在**C1Toolbar**下添加一个**TextBox** 控件。
6. 运行应用程序并观察通过点击**C1ToolbarButtons**以执行常用的剪切板指令(粘贴，拷贝和剪切) 。



9.2 第二部分：创建自定义指令

如果在指令库中的指令不能满足您的需求，此时您可以创建您自己的指令，您仅需通过实现ICommand接口实现自定义指令，WPF 提供了一个名为**RoutedCommand**(及 RoutedUICommand)的具体实现，通过元素树来定义一个指令路由。

以下步骤将介绍如何使用**RoutedCommand**s来添加一个自定义指令到**C1Toolbar**。

1. 使用第一部分的示例，打开隐藏代码文件。
2. 定义一个名为**ClearCommand**的RoutedCommand，该指令将用于清除页面中**TextBox** 的文本。

```
public static RoutedCommand ClearCommand = new RoutedCommand();
```

3. 创建一个事件处理以定义指令逻辑。

C#

```
//执行ClearCommand逻辑
private void ExecutedClearCommand(object sender, ExecutedRoutedEventArgs e)
{
    textBox1.Clear();
}
```

4. 创建另一个事件处理以决定指令是否能被执行，当一个指令不能被执行时，与指令相关联的Toolbar按钮将变成灰色并禁用。

C#

```
// 若文本框中有文本则返回true。
private void CanExecuteClearCommand(object sender, CanExecuteRoutedEventArgs e)
{
    if (textBox1.Text.Length > 0)
    {
        e.CanExecute = true;
    }
    else
```

```
    {
        e.CanExecute = false;
    }
}
```

5. 下一步，创建一个CommandBinding以关联事件处理的指令，当指令调用时，元素树将被遍历以查找CommandBinding对象，将以下代码放到InitializeComponent 后来调用您的页面：

C#

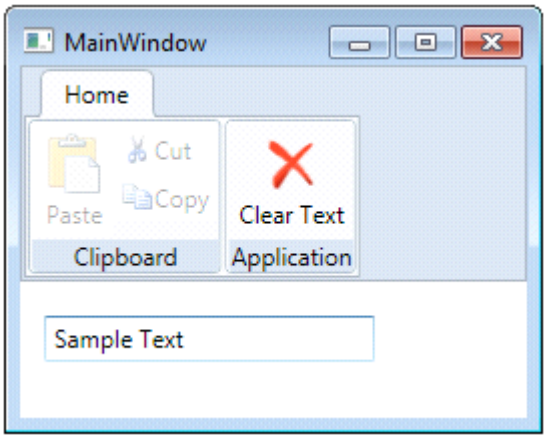
```
CommandBinding customCommandBinding = new CommandBinding(ClearCommand,
ExecutedClearCommand, CanExecuteClearCommand);

// 添加CommandBinding到根元素
this.CommandBindings.Add(customCommandBinding);
```

6. 在包含**C1Toolbar**的XAML中，在分组中添加一个新的**C1ToolbarButton**并名为**Application**。
7. 在代码中设置**Command** 属性为静态指令。

XAML

```
<c1:C1ToolbarGroup Header="Application">
    <c1:C1ToolbarButton LabelTitle="Clear Text" Command="{x:Static
local:MainWindow.ClearCommand}" LargeImageSource="/Resources/clear.png"/>
</c1:C1ToolbarGroup>
```

8. 运行应用程序并观察当工具栏按钮被点击时，**ClearCommand** 将通过遍历元素树查找项关联的**CommandBinding**，如果找到则相关联的事件处理将调用您的指令逻辑并执行。



9.3 第三部分：MVVM中使用指令

指令操作是MVVM（Model-View-ViewModel）设计模式的重要部分，它将UI从业务逻辑中分离开来，**WPF版Toolbar**支持指令框架并应用于广泛使用的MVVM模式中，在MVVM设计的应用程序中，指令目标常常在ViewModel中完成，其并不是UI元素树的部分，因此，在MVVM中使用RoutedCommand并不是一个好的ICommand实现，您可以使用一个特别的实现如**DelegateCommand** 或 **RelayCommand**，可以使您的View 绑定到非UI元素树的对象。

以下步骤介绍如何使用**RelayCommand** 类和**C1Toolbar** 创建第二部分相同的自定义指令。

1. 创建一个名为**MainViewModel**的新类以实现**INotifyPropertyChanged** 接口，该类将作为我们包含C1Toolbar View的ViewModel。

C#

```
class MainViewModel : System.ComponentModel.INotifyPropertyChanged
{
    private string textValue = "";
    public string TextValue
    {
        get
        {
            return textValue;
        }
        set
        {
            textValue = value;
            OnPropertyChanged("TextValue");
        }
    }

    private RelayCommand clearCommand;
    public ICommand ClearCommand
    {
        get
        {
            if (clearCommand == null)
            {
                clearCommand = new RelayCommand(param => this.Clear(), param =>
this.CanClear());
            }
            return clearCommand;
        }
    }

    private bool CanClear()
    {
        return textValue.Length > 0;
    }

    private void Clear()
    {
        TextValue = "";
    }

    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged(string propertyName)
    {
        PropertyChangedEventHandler handler = PropertyChanged;

        if (handler != null)
        {
            handler(this, new PropertyChangedEventArgs(propertyName));
```

```
        }
    }

}
```

该类使用**RelayCommand**，它是**DelegateCommand**轻量级的变体，两个ICommand的实现允许您将指令逻辑作为参数委托给方法进行传递，指令目标不是UI元素树的部分，**RelayCommand** 和 **DelegateCommand** 类不是WPF框架的组成部分，且不能通过MVVM工具箱在线查找。

2. 添加**RelayCommand** 类到您的项目中。

C#

```
/// <summary>
///     该类允许将指令逻辑作为参数委托给方法传递，
///     并使一个View绑定指令到非元素树的对象。
/// </summary>
public class RelayCommand : ICommand
{
    #region Fields

    readonly Action<object> _execute;
    readonly Predicate<object> _canExecute;

    #endregion // Fields

    #region Constructors

    public RelayCommand(Action<object> execute)
        : this(execute, null)
    {
    }

    public RelayCommand(Action<object> execute, Predicate<object> canExecute)
    {
        if (execute == null)
            throw new ArgumentNullException("execute");

        _execute = execute;
        _canExecute = canExecute;
    }
    #endregion // 构造函数

    #region ICommand Members

    [DebuggerStepThrough]
    public bool CanExecute(object parameter)
    {
        return _canExecute == null ? true : _canExecute(parameter);
    }

    public event EventHandler CanExecuteChanged
```

```
    {
        add { CommandManager.RequerySuggested += value; }
        remove { CommandManager.RequerySuggested -= value; }
    }

    public void Execute(object parameter)
    {
        _execute(parameter);
    }

    #endregion // ICommand成员
}
```

3. 为了绑定到ViewModel，需要添加以下XAML 到您包含**C1Toolbar** 的View上方:

XAML

```
<Window.Resources>
    <local:MainViewModel x:Key="viewModel" />
</Window.Resources>
<Window.DataContext>
    <Binding Source="{StaticResource viewModel}"/>
</Window.DataContext>
```

添加到您ViewModel 的XAML将作为资源绑定到Window或UserControl的**DataContext**中。

4. 此时添加以下工具栏分组到您在第一部分创建的**C1Toolbar** 。

XAML

```
<c1:C1ToolbarGroup Header="Application">
    <c1:C1ToolbarButton LabelTitle="Clear Text" Command="{Binding ClearCommand}"
LargeImageSource="/Resources/delete.png"/>
</c1:C1ToolbarGroup>
```

该工具栏分组包含一个带有Command属性的**C1ToolbarButton**，Command绑定到ViewModel定义的**ClearCommand**。

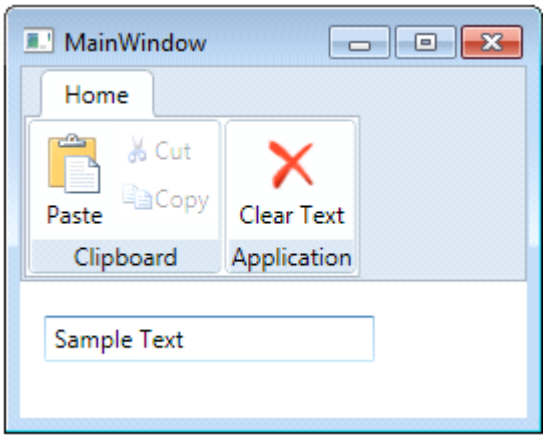
5. 由于指令遵循MVVM优秀的特性，文本框的 **Text** 属性也应该绑定到ViewModel的值，如果我们希望在上面应用业务逻辑，则需要绑定**Text** 属性到ViewModel中定义的**TextValue**。

XAML

```
<TextBox Grid.Row="1" Text="{Binding TextValue,
UpdateSourceTrigger=PropertyChanged}" Height="23" HorizontalAlignment="Left"
Margin="12,17,0,0" Name="textBox1" VerticalAlignment="Top" Width="165" />
```

通过将**UpdateSourceTrigger** 设置为 **PropertyChanged**，在ViewModel中的TextValue属性将可以在任何时刻更新Text值，而不是只有当文本框失去焦点时。

6. 运行应用程序并观察行为中的自定义**RelayCommand** 。



通过MVVM设计模式来降低UI与业务逻辑之间的耦合度，对于您的应用开发是非常实用的。

10 C1Toolbar指令教程（Silverlight版）

Silverlight 4 支持在任意继承**ButtonBase**类的对象中使用ICommand接口，包括**C1ToolbarButton**在内的任意按钮可以通过控件的**Command**属性关联实现ICommand的对象，尽管支持**ICommand**接口，但Silverlight并不提供任何内置的实现，**Silverlight Edition**包括**C1Command** 及**C1ToolbarCommand** 两个实现，因此您不需要自己来编写。

C1Command 类被包含在**C1.Silverlight** 配置中，**C1ToolbarCommand**被包含在**C1.Silverlight.Toolbar**配置中，您可以通过添加几个额外的工具栏，并关联相应的标签和图片属性以扩展 **C1Command** 。
以下步骤演示了如何使用**C1Toolbar**指令。

使用C1ToolbarCommand

1. 打开或创建一个新的Silverlight 应用程序。
2. 添加两个**RowDefinitions** 到默认的**Grid** 元素，将第一行设置为自动高度，如：

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

3. 添加一个C1Toolbar以填充第一行。
4. 在第二行中添加一个**TextBox**。
5. 粘贴下述**C1ToolbarCommand** 到UserControl资源：

XAML
```
<UserControl.Resources>
    <c1:C1ToolbarCommand x:Key="cmdClear" LabelTitle="Clear Text"
LargeImageSource="/Resources/delete.png" />
</UserControl.Resources>
```

6. 该指令将用于清除**TextBox**中内容，您可以为**C1ToolbarCommand**指定一个**LargeImageSource**和**SmallImageSource**，但这不是必须的。
7. 粘贴以下XAML 来填充带有标签或者分组按钮的C1Toolbar:

XAML
```
<c1:C1Toolbar Name="c1Toolbar1">
    <c1:C1ToolbarTabControl>
        <c1:C1ToolbarTabItem Header="Home">
            <c1:C1ToolbarGroup Header="Application">
                <c1:C1ToolbarButton
c1:CommandExtensions.Command="{StaticResource cmdClear}" />
            </c1:C1ToolbarGroup>
        </c1:C1ToolbarTabItem>
    </c1:C1ToolbarTabControl>
</c1:C1Toolbar>
```

8. 这里我们设置**c1:CommandExtensions.Command** 属性到我们的指令中，注意**C1ToolbarButton** 也有一个继承的**Command**属性，当使用C1Commands时最好使用附加属性**CommandExtensions.Command** 来设置指令。
9. 添加一个**TextBox** 到页面下方名为“textBox1”的工具栏。
10. 接下来，您需要注册该指令后，在页面初始化使用此代码：

C#

```
// 注册指令方法
CommandManager.RegisterClassCommandBinding(GetType(), new
CommandBinding((C1ToolbarCommand)Resources["cmdClear"], Clear, CanClear));
// 检测已注册指令的执行能力
CommandManager.InvalidateRequerySuggested();
```

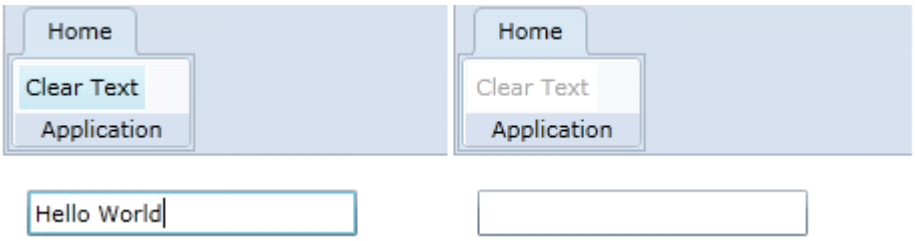
11. **C1.Silverlight.CommandManager** 提供用于注册指令的实用方法。
12. 粘贴以下**Clear** 及 **CanClear** 事件处理以执行指令逻辑：

C#

```
private void Clear(object sender, ExecutedRoutedEventArgs e)
{
    textBox1.Text = "";
}

private void CanClear(object sender, CanExecuteRoutedEventArgs e)
{
    if (textBox1.Text.Length > 0)
    {
        e.CanExecute = true;
    }
    else
        e.CanExecute = false;

}
```

13. 在Silverlight版中您需要显式的在代码中检查已注册指令的能力，需要在**TextBox** 的**TextChanged** 事件中添加以下代码：

C#

```
private void textBox1_TextChanged(object sender, TextChangedEventArgs e)
{
    CommandManager.InvalidateRequerySuggested();
}
```

14. 运行示例，您现在可以通过在C1Toolbar中使用C1Command来清除文本框中的内容。