Toolbar for WPF/Silverlight

2019.04.03 更新

グレープシティ株式会社

目次

製品の概要

Toolbar for WPF/Silverlight を使用すると、アプリケーションにナビゲーション機能を追加するためのカスタムツールバーを作成できます。**Toolbar for WPF/Silverlight** には1つのコントロール **C1Toolbar** が含まれます。このコントロールにはさまざまな項目（リンク、カスタムコンテンツ、セパレータなど）やグループを格納できるため、このツールバーには、ほとんどすべての種類のコントロールを配置できる柔軟性があります。

> **メモ**：説明内に含まれるクラスおよびメンバーに対するリファレンスへのリンクは、原則としてWPF版のリファレンスページを参照します。Silverlight版については、目次から同名のメンバーを参照してください。

主な特長

Toolbar for WPF/Silverlight を使用すると、機能豊富でカスタマイズされたアプリケーションを作成できます。次のような主要機能を利用して、**Toolbar for WPF/Silverlight** を最大限に活用してください。

- **タブとグループ**
  タブやグループを使用して、関連するコマンドをまとめて整理することができます。たとえば、関連するツールバーボタン同士を「フォント」や「クリップボード」などのグループにまとめることができます。タブやグループには、それに含まれるコマンドやアクションを表すラベルを付けることができます。
  詳細については、「ツールバーの要素」を参照してください。

- **リボンに似た機能**
  C1Toolbar には、折りたたみ、ヘルプ、ダイアログランチャーのボタンなど、Microsoft の Ribbon に似た機能が組み込まれています。

- **軽量の ToolbarStrip**
  簡単なシナリオで別途使用できる軽量の **C1ToolbarStrip** を作成します。

- **スタイルが設定されたボタンおよびドロップダウン**
  C1ToolBar コントロールは、任意の UIElement をサポートするコンテナです。ツールバーを簡単に作成できるように、**Toolbar for WPF/Silverlight** には、ボタン、トグルボタン、セパレータ、ドロップダウンコントロールなど、ツールバーに配置できるさまざまなコントロールが用意されています。イベントハンドラをアタッチしたり、コマンドフレームワークを活用して、各コントロールに動作を追加できます。

- **WPF コマンドフレームワークのサポート**
  ツールバーは、WPF コマンドフレームワークをサポートします。それぞれタイトルと小さな/大きな画像ソースを含むコマンド群を一度作成し、それらを複数のツールバーで共有できます。
  詳細については、「コマンドチュートリアル」を参照してください。

- **オーバーフローのサポート**
  ストリップパネルの空きスペースに応じて、ストリップパネルとドロップダウンオーバーフローパネルの間で項目が入れ替わります。デフォルトでは自動的に入れ替わりますが、必要に応じていつでも無効に設定できます。
  詳細については、「ツールバーストリップ」を参照してください。

- **方向の変更**
  ツールバーの方向を水平（デフォルト）と垂直から選択します。
  詳細については、「ツールバーのレイアウト」を参照してください。

- **ClearStyle 技術のサポート**
  C1Toolbarは、コントロールのテンプレートを変更することなくコントロールの色を簡単に変更できる ComponentOne の新しい ClearStyle 技術をサポートします。色のプロパティをいくつか設定するだけで、**C1Toolbar** 要素のスタイルを簡単に設定できます。ClearStyle 技術の詳細については、「Toolbar の ClearStyle プロパティ」を参照してください。

- **Silverlight Toolkit テーマのサポート**
  ExpressionDark、ExpressionLight、WhistlerBlue、RainierOrange、ShinyBlue、BureauBlack など、最もよく使用されている Microsoft Silverlight Toolkit テーマが組み込みでサポートされており、それを使用して UI にスタイルを追加できます。

クイックスタート

このクイックスタートは、**Toolbar for WPF/Silverlight** を初めて使用するユーザーのために用意されています。このクイックスタートでは、Visual Studio で新しいプロジェクトを作成し、アプリケーションに **Toolbar for WPF/Silverlight** を追加して、**C1ToolbarGroup**、**C1ToolbarStrip**、**C1ToolbarButton**、**C1ToolbarToggleButton** などの **C1Toolbar** 項目を追加します。詳細については、「ツールバーの要素」を参照してください。

手順 1:コントロールの追加

プロジェクトを設定し、**C1Toolbar** コントロールをアプリケーションに追加するには、次の手順に従います。

1. Visual Studio で新しい WPF/Silverlight プロジェクトを作成します。
2. **C1.WPF** および **C1.WPF.C1Toolbar**または **C1.Silverlight** および **C1.Silverlight.C1Toolbar** アセンブリへの参照を追加します。ソリューションエクスプローラで、**[参照]**を右クリックし、**[参照の追加]**を選択します。**[参照の追加]**ダイアログボックスで、**[参照]**タブを選択します。**C1.WPF.C1Toolbar.4.dll** および **C1.WPF**または**C1.Silverlight.C1Toolbar.dll** および **C1.Silverlight** を参照し、**[OK]**を選択します。
3. System および **C1.WPF.C1Toolbar**または**C1.Silverlight.C1Toolbar** プリフィックスを定義します。

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. LayoutRoot グリッドに2つの行を追加し、最初の行の**高さ**を **Auto** に設定します。

XAML

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

5. ページの最初の行内に **C1Toolbar** をドロップし、デフォルトプロパティ **Height="100" HorizontalAlignment="Left" Margin="174,34,0,0"** を削除します。XAML は、次のようになります。

XAML

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

```
   <c1:C1Toolbar Grid.Row="1" Name="c1Toolbar1">
   </c1:C1Toolbar>
```

手順 2：C1ToolbarGroup の追加

この手順では、引き続き Visual Studio で作業し、**C1Toolbar** に **C1ToolbarGroup** を追加します。次に、**C1ToolbarGroup** に **C1ToolbarButton** を追加します。

1. **C1Toolbar** コントロールを右クリックし、**[プロパティ]**を選択して**[プロパティ]**ウィンドウを開きます。**ToolbarItems** プロパティに移動し、省略符ボタンをクリックして**[コレクションエディター:Items]**を開きます。
2. **[追加]**ボタンを2回クリックして、2つの **C1ToolbarGroup** を追加し、**[OK]**をクリックします。XAML は、次のようになります。

XAML

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

3. XAML で、各 **C1ToolbarGroup** の **Header** プロパティをそれぞれ **クリップボード** と **フォント** に設定します。XAML は、次のようになります。

XAML

```
<c1:C1Toolbar Grid.Row="1" Name="c1Toolbar1">
   <c1:C1ToolbarGroup Header="クリップボード"/>
   <c1:C1ToolbarGroup Header="フォント"/>
</c1:C1Toolbar>
```

4. **クリップボード C1ToolbarGroup** を選択し、**Items** コレクションエディタの横にある省略符ボタンをクリックします。
5. **[項目の選択]**ドロップダウンから、**[追加]**を3回クリックして、コレクションに3つの **C1ToolbarButton** を追加します。
6. 最初の **C1ToolbarButton** を選択し、プロパティウィンドウで **その他** ノードを展開します。**LabelTitle** プロパティを「貼り付け」に設定し、他の2つのボタンのプロパティをそれぞれ「切り取り」と「コピー」に設定します。
7. オプションの手順として、**LargeImageSource** または **SmallImageSource** プロパティをプロジェクト内にあるリソースに設定したり、新しい画像を追加することができます。
8. **[OK]**をクリックして**[コレクションエディター:Items]**を閉じます。XAML は次のようになります。

XAML

```
<c1:C1Toolbar Name="c1Toolbar1">
    <c1:C1ToolbarGroup Header="クリップボード">
            <c1:C1ToolbarButton LabelTitle="貼り付け"
LargeImageSource="/C1Toolbar;component/Resources/Paste.png" />
            <c1:C1ToolbarButton LabelTitle="切り取り"
SmallImageSource="/C1Toolbar;component/Resources/Cut.png" />
            <c1:C1ToolbarButton LabelTitle="コピー"
SmallImageSource="/C1Toolbar;component/Resources/Copy.png" />
        </c1:C1ToolbarGroup>
        <c1:C1ToolbarGroup Header="フォント"/>
</c1:C1Toolbar>
```

手順 3：C1ToolbarStrip および C1ToolbarToggleButton の追加

この手順では、引き続き Visual Studio で作業し、「フォント」 **C1ToolbarGroup** に **C1ToolbarStrip** を追加してから、この **C1ToolbarGroup** にいくつかの **C1ToolbarToggleButton** を追加します。

1. XAML で **フォント C1ToolbarGroup** を選択し、**C1ToolbarStrip** を追加します。

```
XAML
<c1:C1ToolbarGroup Header="フォント">
    <c1:C1ToolbarStrip />
</c1:C1ToolbarGroup>
```

残りの XAML は次のようになります。

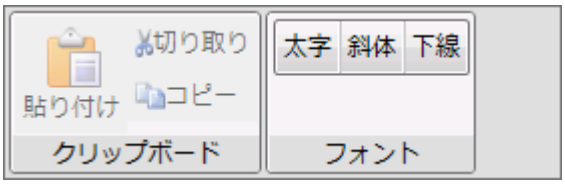
```
XAML
<c1:C1Toolbar Grid.Row="1" Name="c1Toolbar1">
    <c1:C1ToolbarGroup Header="クリップボード">
        <c1:C1ToolbarButton LabelTitle="貼り付け" />
        <c1:C1ToolbarButton LabelTitle="切り取り" />
        <c1:C1ToolbarButton LabelTitle="コピー" />
    </c1:C1ToolbarGroup>
    <c1:C1ToolbarGroup Header="フォント">
    <c1:C1ToolbarStrip />
    </c1:C1ToolbarGroup>
</c1:C1Toolbar>
```

2. **C1ToolbarStrip** を選択し、**[コレクションエディター:Items]**を開きます。

3. **[項目の選択]**ドロップダウンから **C1ToolbarToggleButton** を選択し、**[追加]**を3回クリックして3つの C1ToolbarToggleButton を追加します。**LabelTitle** プロパティをそれぞれ*「太字」*、*「斜体」*、および*「下線」*に設定します。[OK]をクリックして **[コレクションエディター:Items]**を閉じます。

```
XAML
<c1:C1ToolbarGroup Header="フォント">
    <c1:C1ToolbarStrip>
        <c1:C1ToolbarToggleButton LabelTitle="太字" />
        <c1:C1ToolbarToggleButton LabelTitle="斜体" />
        <c1:C1ToolbarToggleButton LabelTitle="下線" />
    </c1:C1ToolbarStrip>
</c1:C1ToolbarGroup>
```

おめでとうございます。これで、**C1Toolbar** を使用してツールバー UI が作成されました。



関連トピック:

- ToolbarButton クリックイベントにロジックを追加する – このトピックは、ボタンクリックイベントを使用して、ボタンにロジックを追加する方法を説明します。
- コマンドチュートリアル – このチュートリアルは、WPF アプリケーションで **C1Toolbar** をコマンドと組み合わせて使用する方法を示します。
- C1Toolbar for Silverlight を使用したコマンド – このチュートリアルは、Silverlight アプリケーションで **C1Toolbar** をコマンドと組み合わせて使用する方法を示します。
- ツールバーグループ内のボタンのサイズとテキストの位置 – このトピックは、**C1ToolbarGroup** の **GroupSizeDefinition** を使用する方法を示します。
- ツールバータブ項目 – このトピックは、**C1ToolbarTabItem** コントロールの追加方法を示します。

簡略化リボン

In general, a ribbon has a standard three line appearance of items. Toolbar for WPF offers SimplifiedRibbon, similar to the Ribbon tool in MS Office 365, which provides you an option to work with a single line collapsed view or an expanded view containing three line appearances. This allows you to allocate more screen space to the working area of your application. In addition, SimplifiedRibbon automatically combines items in a group into a dropdown when it doesn't have enough space to display all group items across the window width.

The SimplifiedRibbon control is represented by the **C1SimplifiedRibbon** class that implements C1ToolbarBase class. The **C1SimplifiedRibbon** class lets you map the items of SimplifiedRibbon using the **RibbonItems** property. In addition, **IsCollapsed** property of the **C1SimplifiedRibbon** class lets you check whether the tab panel displaying the ribbon items is collapsed or expanded. It also allows you to choose marquee brush for the toolbar using the **MarqueeBrush** property of the C1SimplifiedRibbon class.

Toolbar for WPF offers another control, C1SimplifiedTabControl, which is a part of C1SimplifiedRibbon control. This control is represented by the **C1SimplifiedTabControl** class which inherits the **C1TabContol** class. It is similar to the ToolbarTabControl used in C1Toolbar control template. The control handles marquee logic and toggle collapse state of tab control. You can use this control inside the C1SimplifiedRibbon control to create a custom control template for C1SimplifiedRibbon.

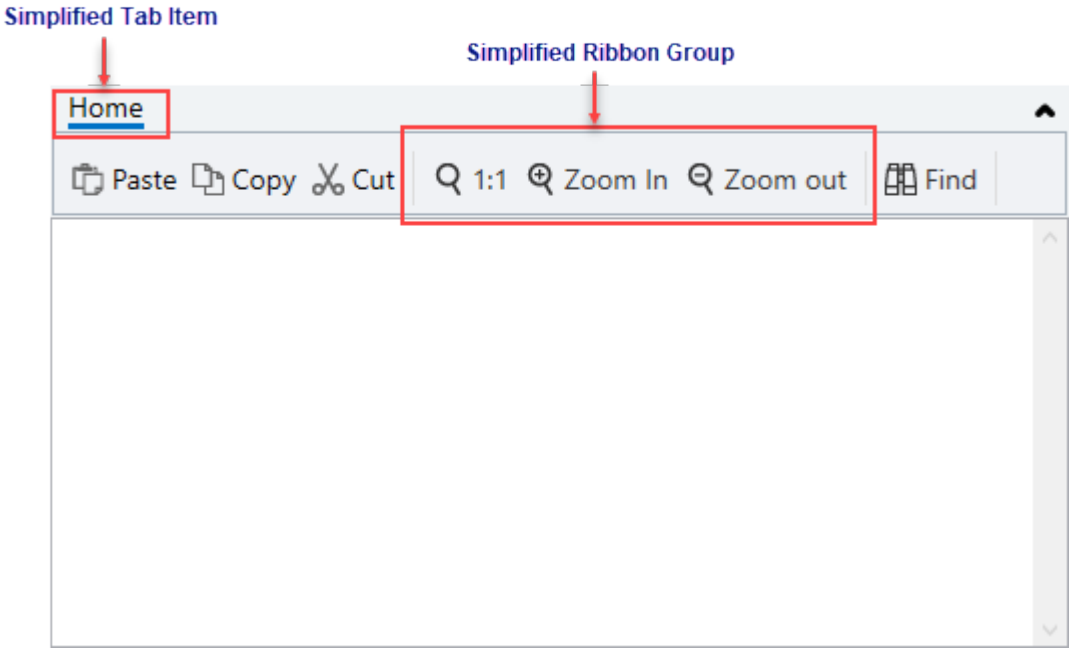
簡略化リボンツールバーの概要

The Simplified Ribbon control acts as a container to hold other controls, such as buttons, check boxes, text boxes, drop-down lists, etc. It lets you organize related commands and buttons together using the following elements which can contain all the mentioned controls to be used in a simplified ribbon.

- **Simplified Tab Item**– It wraps the tab header and simplified ribbon group and holds the collection of SimplifiedRibbon group. The simplified tab item is represented by the **C1SimplifiedTabItem** class which is inherited from the **C1TabItem** class.

  The **C1SimplifiedTabItem** class provides **Groups** property which lets the user get or set different group of options in the toolbar.

- **Simplified Ribbon Group** - It contains collection of toolbar elements and handles the simplified ribbon control layout which represent toolbar options of a specific type. The simplified ribbon group is represented by the **C1SimplifiedRibbonGroup** class which inherits the **C1ToolbarGroup** class.

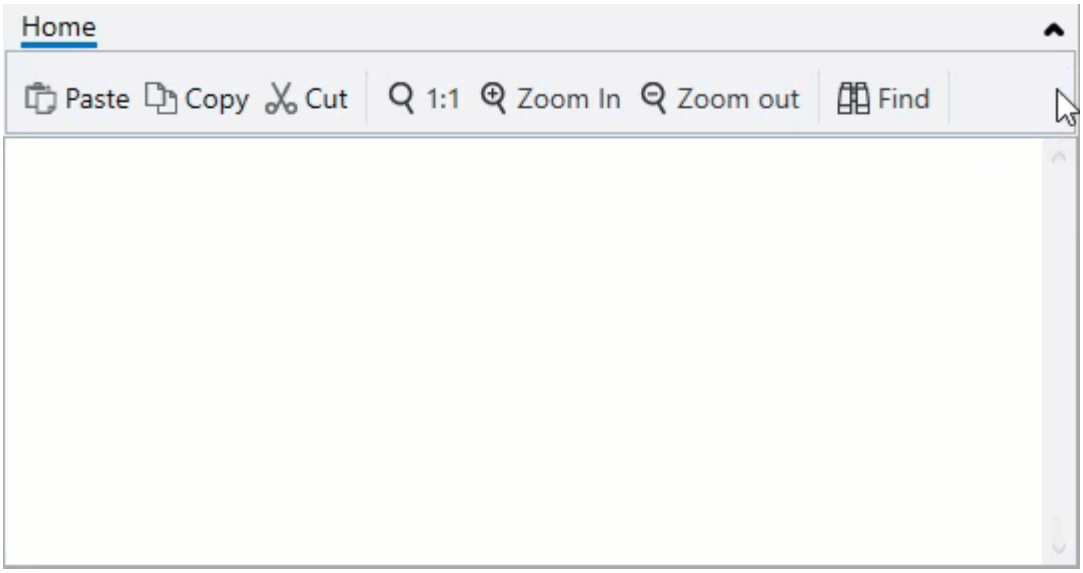


In addition to the elements listed above, you can also add the toolbar elements to the SimplifiedRibbon. For information on Toolbar elements, see Toolbar Elements.

クイックスタート

This quick start will guide you through the steps of adding C1SimplifiedRibbon control to a project, adding different elements and toolbar commands.

The following GIF shows how the simplified ribbon toolbar appears on collapsing and expanding the ribbon.



To create a simple toolbar with simplified ribbon style, follow these steps:

1. Create a WPF App.
2. Add a Resources folder to the application and add images of the toolbar commands you want to add to it. For example, we have added small and big images for all the edit, scale and search options in the Resources folder.
3. Add the SimplifiedRibbon control to the XAML designer using the following code.

C#

```
<c1:C1SimplifiedRibbon FontSize="14" />
```

4. Define the following instances of C1ToolbarCommand inside the <Window.Resources></ Window.Resources> tag to define commands to be used in the toolbar.

C# copyCode

```
<c1:C1ToolbarCommand x:Key="cmdPaste" LabelTitle=" Paste"

LargeImageSource="/SimplifiedRibbon;component/Resources/paste32.png"

SmallImageSource="/SimplifiedRibbon;component/Resources/paste16.png" />
<c1:C1ToolbarCommand x:Key="cmdCut" LabelTitle=" Cut"

LargeImageSource="/SimplifiedRibbon;component/Resources/cut32.png"

SmallImageSource="/SimplifiedRibbon;component/Resources/cut16.png"/>
<c1:C1ToolbarCommand x:Key="cmdCopy" LabelTitle=" Copy"

LargeImageSource="/SimplifiedRibbon;component/Resources/copy32.png"

SmallImageSource="/SimplifiedRibbon;component/Resources/copy16.png"/>
<c1:C1ToolbarCommand x:Key="cmdZoomOriginal" LabelTitle=" 1:1" />
<c1:C1ToolbarCommand x:Key="cmdZoomIn" LabelTitle=" Zoom In" />
<c1:C1ToolbarCommand x:Key="cmdZoomOut" LabelTitle=" Zoom Out" />
<c1:C1ToolbarCommand x:Key="cmdFind" LabelTitle=" Find" />
```

5. Add SimplifiedTabItem and SimplifiedRibbonGroup elements to the simplified ribbon to define the toolbar tabs and the groups inside a tab.
Observe: A tab named Home is added with three groups named Edit, Scale and Search containing different commands, such as Cut, Copy, Paste, Zoom Original, Zoom In, Zoom out, and Find.

C# copyCode

```
    <c1:C1SimplifiedTabItem Header="Home">
    <c1:C1SimplifiedRibbonGroup Header="Edit">
        <c1:C1SimplifiedRibbonGroup.GroupSizeDefinitions>
            <c1:C1ToolbarGroupSizeDefinition>
                <c1:C1ToolbarControlSize>Large</c1:C1ToolbarControlSize>
            </c1:C1ToolbarGroupSizeDefinition>
        </c1:C1SimplifiedRibbonGroup.GroupSizeDefinitions>
            <c1:C1ToolbarButton Command="{StaticResource cmdPaste}" />
            <c1:C1ToolbarButton Command="{StaticResource cmdCopy}" />
            <c1:C1ToolbarButton Command="{StaticResource cmdCut}" />
        </c1:C1SimplifiedRibbonGroup>

    <c1:C1SimplifiedRibbonGroup VerticalContentAlignment="Center"
Header="Scale">
            <c1:C1SimplifiedRibbonGroup.GroupSizeDefinitions>
                <c1:C1ToolbarGroupSizeDefinition>
                    <c1:C1ToolbarControlSize>Medium</c1:C1ToolbarControlSize>
                </c1:C1ToolbarGroupSizeDefinition>
            </c1:C1SimplifiedRibbonGroup.GroupSizeDefinitions>
            <c1:C1ToolbarItem Command="{StaticResource cmdZoomOriginal}"
Content=" 1:1">
                <c1:C1ToolbarItem.LargeIcon>
                    <c1:C1BitmapIcon
```

```
Source="/SimplifiedRibbon;component/Resources/zoom_original32.png"
                                    Width="32" Height="32" />
                </c1:C1ToolbarItem.LargeIcon>
                <c1:C1ToolbarItem.MediumIcon>
                    <c1:C1BitmapIcon
Source="/SimplifiedRibbon;component/Resources/zoom_original16.png"
                                    Width="16" Height="16" />
                </c1:C1ToolbarItem.MediumIcon>
            </c1:C1ToolbarItem>
            <c1:C1ToolbarItem Command="{StaticResource cmdZoomIn}" Content="
Zoom In">
                <c1:C1ToolbarItem.LargeIcon>
                    <c1:C1BitmapIcon
Source="/SimplifiedRibbon;component/Resources/zoom_in32.png"
                                    Width="32" Height="32" />
                </c1:C1ToolbarItem.LargeIcon>
                <c1:C1ToolbarItem.MediumIcon>
                    <c1:C1BitmapIcon
Source="/SimplifiedRibbon;component/Resources/zoom_in16.png"
                                    Width="16" Height="16" />
                </c1:C1ToolbarItem.MediumIcon>
            </c1:C1ToolbarItem>
            <c1:C1ToolbarItem Command="{StaticResource cmdZoomOut}" Content="
Zoom out">
            <c1:C1ToolbarItem.LargeIcon>
                <c1:C1BitmapIcon
Source="/SimplifiedRibbon;component/Resources/zoom_out32.png"
                                Width="32" Height="32" />
            </c1:C1ToolbarItem.LargeIcon>
            <c1:C1ToolbarItem.MediumIcon>
                <c1:C1BitmapIcon
Source="/SimplifiedRibbon;component/Resources/zoom_out16.png"
                                Width="16" Height="16" />
            </c1:C1ToolbarItem.MediumIcon>
        </c1:C1ToolbarItem>
    </c1:C1SimplifiedRibbonGroup>

    <c1:C1SimplifiedRibbonGroup Header="Search">
        <c1:C1SimplifiedRibbonGroup.GroupSizeDefinitions>
            <c1:C1ToolbarGroupSizeDefinition>
                <c1:C1ToolbarControlSize>Large</c1:C1ToolbarControlSize>
                <c1:C1ToolbarControlSize>Medium</c1:C1ToolbarControlSize>
            </c1:C1ToolbarGroupSizeDefinition>
        </c1:C1SimplifiedRibbonGroup.GroupSizeDefinitions>
        <c1:C1ToolbarItem Command="{StaticResource cmdFind}" Content=" Find">
            <c1:C1ToolbarItem.LargeIcon>
                <c1:C1BitmapIcon
Source="/SimplifiedRibbon;component/Resources/find32.png"
                                Width="32" Height="32" />
            </c1:C1ToolbarItem.LargeIcon>
            <c1:C1ToolbarItem.MediumIcon>
                <c1:C1BitmapIcon
Source="/SimplifiedRibbon;component/Resources/find16.png"
```

```
                                    Width="16" Height="16" />
                </c1:C1ToolbarItem.MediumIcon>
            </c1:C1ToolbarItem>
        </c1:C1SimplifiedRibbonGroup>
</c1:C1SimplifiedTabItem>
```

Similarly, you can add more tabs, groups, and commands using C1SimplifiedTabItem and C1SimplifiedRibbonGroup.

6. Run the application and observe the ribbon toolbar which can be collapsed using the expand/collapse icon appearing on the top right of the ribbon.

XAML クイックリファレンス

このトピックでは、**C1Toolbar** コントロールの作成に使用される XAML の概要を提供します。

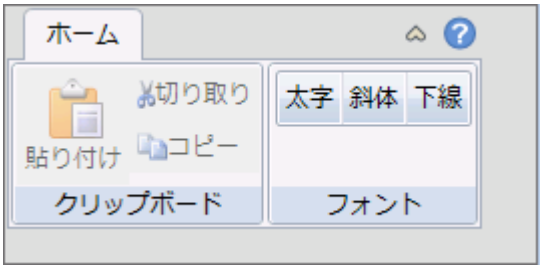
例:C1Toolbar に項目を追加する

次の XAML コードは、**C1ToolbarGroup**、**C1ToolbarStrip**、および **C1ToolbarButton** を **C1Toolbar** コントロールに追加する方法を示します。

XAML

```
<c1:C1Toolbar Name="c1Toolbar1">
 <c1:C1ToolbarTabControl>
  <c1:C1ToolbarTabItem Header="ホーム">
   <c1:C1ToolbarGroup Header="クリップボード">
     <c1:C1ToolbarButton LabelTitle="貼り付け" LargeImageSource="/Images/Paste.png" />
     <c1:C1ToolbarButton LabelTitle="切り取り" SmallImageSource="/Images/Cut.png" />
     <c1:C1ToolbarButton LabelTitle="コピー" SmallImageSource="/Images/Copy.png" />
   </c1:C1ToolbarGroup>
   <c1:C1ToolbarGroup Header="フォント">
    <c1:C1ToolbarStrip>
       <c1:C1ToolbarToggleButton LabelTitle="太字" />
       <c1:C1ToolbarToggleButton LabelTitle="斜体" />
       <c1:C1ToolbarToggleButton LabelTitle="下線" />
    </c1:C1ToolbarStrip>
   </c1:C1ToolbarGroup>
  </c1:C1ToolbarTabItem>
 </c1:C1ToolbarTabControl>
</c1:C1Toolbar>
```

上記の XAML コードを追加した後、**C1Toolbar** コントロールは次の図のように表示されます。



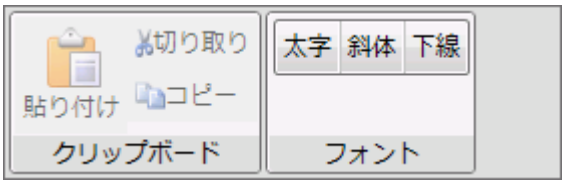
ツールバーの要素

C1Toolbar は、他のコントロール（ボタン、チェックボタン、テキストボックス、ドロップダウンリスト、分割ボタン、セパレータなど）を保持するためのコンテナとして使用されます。**C1ToolBar** は、**C1ToolBar** で使用されるさまざまなコントロールを7つのオブジェクト（**C1ToolbarButton**、**C1ToolbarDropDown**、**C1ToolbarGroup**, **C1ToolbarSplitButton**、**C1ToolbarStrip**、**C1ToolbarTabItem**、**C1ToolbarToggleButton**）によってサポートします。

ツールバーグループ

C1ToolBarGroup オブジェクトは、ツールバー要素のグループを定義します。**C1ToolbarGroup** は、**C1ToolbarButton**、**C1ToolbarToggleButton**、**C1ToolbarDropDown**、**C1ToolbarSplitButton** の各ツールバーコントロールを保持できます。

C1ToolbarGroup にツールバー項目を追加すると、次の図のように、四角形のボックス内に子のツールバー項目がまとめて表示されます。



C1ToolbarGroup は、通常、グループ内の機能が相互に排他的である場合に使用されます。つまり、このグループ内のボタンで表される機能のうち、一度に1つの機能のみをオンにすることができます。

C1ToolbarGroup 内のコントロール（**C1ToolbarButon** および **C1ToolbarToggleButton**）の画像は、**GroupSizedDefinitions** クラスを使用して、大、中、小のサイズを指定できます。

GroupSizeDefinition は、**C1ToolbarGroup** 内のコントロールのサイズ（大、中、小）とボタンテキストの位置を定義します。"大" を使用すると、グループ内のすべてのコントロール（**C1ToolbarButon** および **C1ToolbarToggleButton**）が、**LargeImageResource** プロパティから画像サイズを取得し、**LabelTitle** プロパティのテキスト値を画像の下に配置します。"中" を使用すると、グループ内のすべてのコントロール（**C1ToolbarButon** および **C1ToolbarToggleButton**）が、**SmallImageResource** プロパティから画像サイズを取得し、**LabelTitle** プロパティのテキスト値を画像の右側に配置します。"小" を使用すると、グループ内のすべてのコントロール（**C1ToolbarButon** および **C1ToolbarToggleButton**）が、SmallImageResource プロパティから画像サイズを取得し、テキストは表示しません。

例：いくつかの C1ToolbarButton を含む C1ToolbarStrip を使用した C1ToolbarGroup

XAML

```
</c1:C1ToolbarGroup>
  <c1:C1ToolbarGroup Header="フォント">
   <c1:C1ToolbarStrip>
      <c1:C1ToolbarToggleButton LabelTitle="太字" />
      <c1:C1ToolbarToggleButton LabelTitle="斜体" />
      <c1:C1ToolbarToggleButton LabelTitle="下線" />
   </c1:C1ToolbarStrip>
  </c1:C1ToolbarGroup>
```

C1ToolBarGroup には、以下のプロパティが含まれます。

プロパティ	定義
FocusBrush	コレクションの指定されたインデックスにある項目を取得または設定します。
GroupSizeDefinitions	コントロール内の項目の選択モードを取得または設定します。
Header	ツールバーグループのヘッダーを取得または設定します。

MouseOverBrush	マウスがあるコントロールを強調表示するために使用されるブラシを取得または設定します。
PressedBrush	クリックされたボタンを描画するために使用されるブラシを取得または設定します。
ShowDialogLauncher	ダイアログランチャーの表示/非表示を取得または設定します。

ツールバーボタン

C1ToolBarButton オブジェクトは、ツールバーボタンを定義します。ボタンには、テキスト、画像、またはテキストと画像の両方を含めることができます。**C1ToolBarButton** オブジェクトごとに、**LabelTitle** プロパティを使用してテキストを設定し、**LargeImageSource** または **SmallImageSource** プロパティを使用して画像を設定します。

ボタンは、設計時に **C1ToolbarItems** コレクションエディタを使用して追加するか、**C1ToolBarItemCollection** コレクションの **C1ToolBarItemCollection.Add** または **C1ToolBarItemCollection.Remove** メソッドを使用してプログラムで追加するか、XAML コードを使用して追加することができます。

C1ToolBarButton には、次の独自のプロパティが含まれます。

プロパティ	定義
LabelTitle	コントロールのラベルタイトルを取得または設定します。
LargeImageSource	コントロールの大きな画像ソースを取得または設定します。
MouseOverBrush	マウスがあるコントロールを強調表示するために使用されるブラシを取得または設定します。
PressedBrush	クリックされたボタンを描画するために使用されるブラシを取得または設定します。
SmallImageSource	コントロールの大きな画像ソースを取得または設定します。

ツールバードロップダウン

C1ToolbarDropDown コントロールは、**C1ToolbarStrip** 上のドロップダウンボタンを表します。これをクリックすると、**Content** プロパティを含むポップアップパネルまたは **Menu** プロパティによって設定されたコンテキストメニューが表示されます。ドロップダウンボタンは、オプションリストを提供します。**C1ToolBarDropDown** ボタンをクリックすると、**Click** イベントが常に発生し、ドロップダウンリストが表示されます。

例1: いくつかのボタンを含むポップアップスタックパネルを使用したドロップダウン

XAML

```
<c1:C1ToolbarDropDown Padding="2" Header="色">
   <c1:C1ToolbarDropDown.Content>
     <StackPanel Margin="2" Orientation="Horizontal">
       <Button Margin="2" Content="赤" Foreground="Red" />
       <Button Margin="2" Content="緑" Foreground="Green" />
       <Button Margin="2" Content="青" Foreground="Blue" />
     </StackPanel>
   </c1:C1ToolbarDropDown.Content>
</c1:C1ToolbarDropDown>
```

例2: ポップアップメニューを使用したドロップダウン

XAML

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

C1ToolBarDropDown には、次の独自のプロパティが含まれます。

プロパティ	定義
ContentBackground	コンテンツ背景を取得または設定します。
Menu	コントロールがクリックされたときに表示されるコンテキストメニューを取得または設定します。
MouseOverBrush	マウスがあるコントロールを強調表示するために使用されるブラシを取得または設定します。
PressedBrush	クリックされたボタンを描画するために使用されるブラシを取得または設定します。

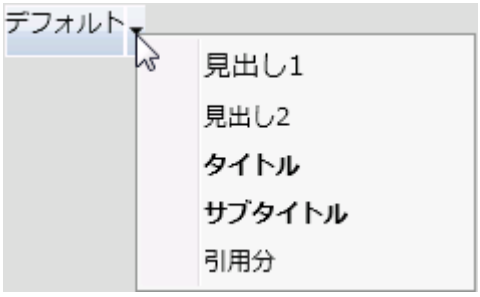
ツールバー分割ボタン

C1ToolbarSplitButton コントロールは、**C1ToobarStrip** 上のドロップダウン分割ボタンを表します。

このコントロールは、**C1ToolbarDropDown** に似ていますが、ボタン領域と下向き矢印という2つのクリック可能な領域が含まれます。

右端にある下向き矢印が付いた四角形の部分をクリックすると、**Content** プロパティを含むポップアップパネルまたは **ContextMenu** プロパティによって設定されたコンテキストメニューが表示されます。ボタンの左側をクリックすると、標準のボタンと同様に **Click** イベントが発生します。通常は、**Click** イベントを使用してデフォルトのアクションまたは最後のアクションを実行しますが、ポップアップを使用して別のオプションを選択することもできます。

次の図のように、ボタンの上にカーソルを合わせると、ボタン領域とドロップダウン矢印を分けるための縦線が表示されます。



例:ポップアップメニューを使用した分割ボタン

XAML

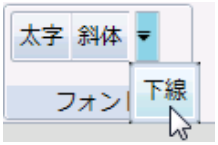
```
<c1:C1ToolbarSplitButton Padding="2" Header="デフォルト" Click="SetDefaultStyle">
   <c1:C1ToolbarDropDown.ContextMenu>
     <ContextMenu>
       <MenuItem Header="見出し 1" FontSize="14" />
       <MenuItem Header="見出し 2" FontSize="12" />
       <MenuItem Header="タイトル" FontWeight="Bold" />
       <MenuItem Header="サブタイトル" FontWeight="SemiBold" FontStyle="Italic" />
       <MenuItem Header="引用分" FontStyle="Italic" />
     </ContextMenu>
   </c1:C1ToolbarDropDown.ContextMenu>
</c1:C1ToolbarSplitButton>
```

ツールバーストリップ

ツールバーコントロールやその他の C1 WPF コントロールのコンテナになります。**C1ToolbarStrip** は、**C1ToolbarButton**、**C1ToolbarToggleButton**、**C1ToolbarDropDown**、**C1ToolbarSplitButton**、**C1Separator**、**C1ComboBox**、および **C1TextBox** コントロールをホストできます。

C1ToolbarStrip は、このストリップパネル内のスペースに応じて、オーバーフローをサポートします。項目は、ストリップパネルとドロップダウンオーバーフローパネルの間を移動します。デフォルトでは自動的に行われますが、無効にしたり、常時有効にしたり、必要時に有効にすることもできます。

次の図は、**C1ToolbarStrip** が提供するオーバーフローのサポートを示します。



例:トグルボタンを使用したツールバーストリップ

XAML

```
<c1:C1ToolbarGroup Header="フォント">
    <c1:C1ToolbarStrip>
        <c1:C1ToolbarToggleButton LabelTitle="太字" />
        <c1:C1ToolbarToggleButton LabelTitle="斜体" />
        <c1:C1ToolbarToggleButton LabelTitle="下線" />
    </c1:C1ToolbarStrip>
</c1:C1ToolbarGroup>
```

C1ToolBarStrip には、次の独自のプロパティが含まれます。

プロパティ	定義
ButtonBackground	コントロール内のボタンの背景に割り当てられる Brush を取得または設定します。
ButtonForeground	コントロール内のボタンの前景に割り当てられる Brush を取得または設定します。
FocusBrush	フォーカスがあるコントロールを強調表示するために使用されるブラシを取得または設定します。
MouseOverBrush	マウスがあるコントロールを強調表示するために使用されるブラシを取得または設定します。
Orientation	ツールバーストリップの方向を取得または設定します。
Overflow	使用可能なスペースに収まらない項目を処理する方法を示す値を取得または設定します。
OverflowMenuItems	オーバーフローメニューの要素が含まれているコレクションを取得します。

OverflowPanel	オーバーフローパネルのテンプレートを取得または設定します。
PressedBrush	クリックされたボタンを描画するために使用されるブラシを取得または設定します。

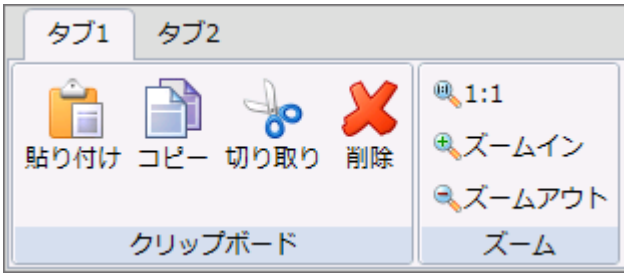
ツールバータブ項目

C1ToolbarTabItem は、**C1Toolbar** のタブ項目を表します。**C1ToolbarTabItem** は、小さなウィンドウスペースに大量の情報を表示する場合に使用すると効果的です。

C1ToolbarTabItem は、いくつかのコントロールを1つのページにまとめ、一度に1つのタブを表示します。

C1ToolbarTabItem は **C1ToolbarGroup** のコレクションを保持でき、各グループはそれぞれ **C1ToolbarButton**、**C1ToolbarToggleButton**、**C1ToolbarDropDown**、および **C1ToolbarSplitButton** を格納できます。

次の図では、2つの **C1ToolbarTabItem** が表示されています。



C1ToolbarTabItem は、**ToolbarItems** コレクションエディタ、XAML、またはプログラムで **C1Toolbar** に追加できます。

デザイナを使用して **C1ToolbarTabItem** を追加するには、次の手順に従います。

1. **C1Toolbar** コントロールをページに追加します。
2. **C1Toolbar** コントロールを選択し、**C1Toolbar** プロパティウィンドウで **ToolbarItems** プロパティの横にある省略符ボタンをクリックします。
   ToolbarItems コレクションエディタが表示されます。
3. 項目選択のドロップダウンリストから **C1ToolbarTabItem** を選択し、**[追加]**をクリックします。
   C1ToolbarTabItem が **C1Toolbar** コントロールに追加されます。
4. **Header** プロパティを **タブ1** に設定します。

XAML コードを使用して **C1ToolbarTabItem** を追加するには、次のコードを追加します。

XAML

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

C1ToolbarTabItem に **C1ToolBarGroup** を追加するには、**Groups** プロパティを使用して、**Groups** コレクションエディタ、XAML、またはプログラムで追加できます。

デザイナを使用して **C1ToolbarTabItem** に **C1ToolbarGroup** を追加するには、次の手順に従います。

1. **C1Toolbar** コントロールをページに追加します。

2. **C1ToolbarTabItem** を右クリックし、**[プロパティ]**を選択します。**C1ToolbarTabItem** プロパティウィンドウで、**Groups** プロパティの横にある**省略符**ボタンをクリックします。
   Groups コレクションエディタが表示されます。

3. **[追加]**をクリックして、**C1ToolbarGroup** を **C1ToolbarTabItem** コントロールに追加します。
   C1ToolbarTabItem に **C1ToolbarGroup** が追加されます。

4. **Header** プロパティを **タブ1** に設定します。

XAML コードを使用して **C1ToolbarTabItem** に **C1ToolbarGroup** を追加するには、次のコードを追加します。

XAML

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

C1ToolBarTabItem には、次の独自のプロパティが含まれます。

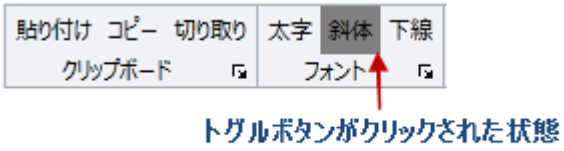
プロパティ	定義
Groups	ツールバーグループのコレクションを取得します。

ツールバートグルボタン

C1ToolbarToggleButton は、**C1ToolbarToolStrip** 上のトグルボタンを表します。これは状態を持つボタンで、ユーザーがオンとオフの状態を切り替えることができます。トグルボタンをクリックすると、もう一度クリックするまで、ボタンは有効状態、つまり押されたままになります。

トグルボタンにはコマンドボタンと同様にグラフィックまたはテキストが表示されますが、ボタンがクリックされて有効状態になると、それがオンの状態で表示されます。オンの状態を示すために、**PressedBrush** プロパティで色を設定できます。

次の図は、3つの **C1ToolbarToggleButton** のうち、**[斜体]**トグルボタンがクリックされた状態を示します。2番目の**[斜体]**トグルボタンは、**PressedBrush** が **Gray** に設定されています。



C1ToolbarToggleButtons を **C1ToolbarStrip** に追加するには、**Items** コレクションエディタ、プログラム、または XAML を使用できます。

例：1つがオンの状態を持つ3つの C1TolbarToggleButton

XAML

```
<c1:C1Toolbar Grid.Row="1" Name="c1Toolbar1">
  <c1:C1ToolbarGroup Header="クリップボード">
    <c1:C1ToolbarButton LabelTitle="貼り付け" />
    <c1:C1ToolbarButton LabelTitle="切り取り" />
```

```
    <c1:C1ToolbarButton LabelTitle="コピー" />
  </c1:C1ToolbarGroup>
  <c1:C1ToolbarGroup Header="フォント">
  <c1:C1ToolbarStrip>
     <c1:C1ToolbarToggleButton LabelTitle="太字" />
     <c1:C1ToolbarToggleButton PressedBrush ="Gray" LabelTitle="斜体" />
     <c1:C1ToolbarToggleButton LabelTitle="下線" />
     </c1:C1ToolbarStrip>
  </c1:C1ToolbarGroup>
</c1:C1Toolbar>
```

C1ToolBarToggleButton には、以下のプロパティが含まれます。

プロパティ	定義
GroupName	相互に排他的な C1ToolbarToggleButton コントロールを指定するための名前を取得または設定します。
LabelTitle	コントロールのラベルタイトルを取得または設定します。
LargeImageSource	コントロールの大きな画像ソースを取得または設定します。
MouseOverBrush	マウスがあるコントロールを強調表示するために使用されるブラシを取得または設定します。
PressedBrush	クリックされたボタンを描画するために使用されるブラシを取得または設定します。
SmallImageSource	コントロールの小さな画像ソースを取得または設定します。

ツールバー項目

Usually toolbar options are displayed along with the visual icons. Such options comprising text and icons can be displayed in the toolbar using the Toolbar Item element which is represented by the **C1ToolbarItem** class. This element lets you render icons using the C1Icon class. The icons can be rendered in three sizes, large, medium and small using the **LargeIcon**, **MediumIcon** and **SmallIcon** properties. The ribbon automatically determines size of the icon to be displayed depending on the collapsed state of the Toolbar or the space available to display the ToolBar.

The following code implements C1ToolbarItem to display large and medium icons for an item in Toolbar.

XAML

```
<c1:C1SimplifiedRibbon FontSize="14">
      <c1:C1SimplifiedTabItem Header="Home">
                <c1:C1SimplifiedRibbonGroup VerticalContentAlignment="Center"
Header="Scale">
                    <c1:C1SimplifiedRibbonGroup.GroupSizeDefinitions>
                        <c1:C1ToolbarGroupSizeDefinition>
                            <c1:C1ToolbarControlSize>Medium</c1:C1ToolbarControlSize>
                        </c1:C1ToolbarGroupSizeDefinition>
                    </c1:C1SimplifiedRibbonGroup.GroupSizeDefinitions>
                    <c1:C1ToolbarItem Command="{StaticResource cmdZoomOriginal}"
Content=" 1:1">
                        <c1:C1ToolbarItem.LargeIcon>
                            <c1:C1BitmapIcon
Source="/SimplifiedRibbon;component/Resources/zoom_original32.png" Width="32"
Height="32" />
                        </c1:C1ToolbarItem.LargeIcon>
```

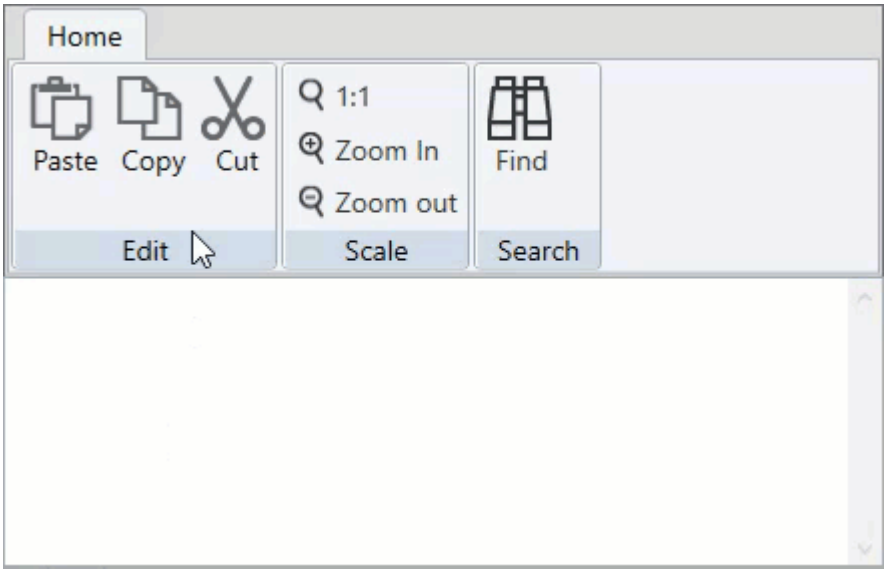
```
                        <c1:C1ToolbarItem.MediumIcon>
                            <c1:C1BitmapIcon
Source="/SimplifiedRibbon;component/Resources/zoom_original16.png" Width="16"
Height="16" />
                        </c1:C1ToolbarItem.MediumIcon>
                    </c1:C1ToolbarItem>
                </c1:C1SimplifiedRibbonGroup>
        </c1:C1SimplifiedTabItem>
</c1:C1SimplifiedRibbon>
```

ツールバー要素のスタイル

Toolbar for WPF allows you to style Toolbar elements to customize their appearance. You can use the following properties to change the foreground and background color of the elements:

- **MouseOverForegroundBrush** - It defines the foreground color to be applied to the toolbar element on mouse hover.
- **PressedForegroundBrush** - It defines the foreground color to be applied to the toolbar element on mouse left click.
- **MouseOverBrush** - It defines the background color to be applied to the toolbar element on mouse hover.
- **PressedBrush** - It defines the background color to be applied to the toolbar element on mouse left click.

The following GIF displays the styling applied on the toolbar elements to change their appearance.



The following code shows the implementation of the properties that can be used to change the appearance of the toolbar elements:

XAML

```
<Style TargetType="c1:C1ToolbarItem">
            <Setter Property="Foreground" Value="#333333"/>
            <Setter Property="MouseOverForegroundBrush" Value="DarkGreen"/>
            <Setter Property="PressedForegroundBrush" Value="Yellow"/>
            <Setter Property="MouseOverBrush" Value="LightGreen"/>
            <Setter Property="PressedBrush" Value="Green"/>
</Style>
```

レイアウトおよび外観

以下のトピックでは、**C1Toolbar** コントロールのレイアウトと外観をカスタマイズする方法について詳しく説明します。組み込みのレイアウトオプションを使用して、グリッドやキャンバスなどのコントロールをパネル内でレイアウトできます。

スタックパネルを使用して、ツールバー項目の外観をカスタマイズできます。スタックパネル内では、ツールバー項目の方向を指定できます。さらに、画像やテキストを追加できます。テキストの配置、フォントスタイル、および色を変更できます。

XAML

```
<c1tb:C1Toolbar>
 <c1tb:C1ToolbarGroup Header="クリップボード">
   <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="切り取り" 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="コピー" 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="貼り付け" VerticalAlignment="Center"
TextAlignment="Center"/>
        </StackPanel>
      </c1tb:C1ToolbarButton>
    </c1tb:C1ToolbarStrip>
  </c1tb:C1ToolbarGroup>
</c1tb:C1Toolbar>
```

ツールバーのレイアウト

C1ToolbarStrip を使用する際は、ツールバーの方向を水平（デフォルト）と垂直から選択します。

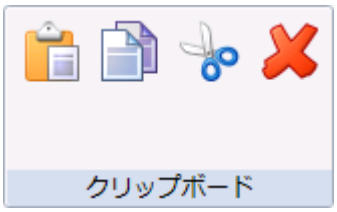
C1Toolbar コントロールのストリップパネルは、**Orientation** プロパティを設定することで、水平方向または垂直方向に表示できます。**C1Toolbar** の各子要素のレイアウトも **StackPanel.Orientation** プロパティを使用して制御できます。

ツールバーグループ内のボタンのサイズとテキストの位置

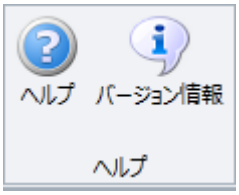
GroupSizeDefinition を使用して、1つの **C1ToolbarGroup** 内のすべての **C1ToolbarButton** および **C1ToolbarToggleButton** に対して同じサイズ（小、中、大）を定義できます。

GroupSizeDefinition は、**C1ToolbarGroup** 内のボタンのサイズ（大、中、小）とボタンテキストの位置を定義します。"大" を使用すると、グループ内のすべてのボタンが、LargeImageResource プロパティから画像サイズを取得し、**LabelTitle** プロパティのテキスト値を画像の下に配置します。"中" を使用すると、グループ内のすべてのボタンが、SmallImageResource プロパティから画像サイズを取得し、LabelTitle プロパティのテキスト値を画像の右側に配置します。"小" を使用すると、グループ内のすべてのボタンが、**SmallImageResource** プロパティから画像サイズを取得し、テキストは表示しません。

次の図は、**LargeImageResource** プロパティを使用してボタンを大サイズで表示し、画像の下に **LabelTitle** プロパティのテキストを表示しています。



次の図は、**SmallImageResource** プロパティを使用してボタンを中サイズで表示し、画像の右側に **LabelTitle** プロパティのテキストを表示しています。



次の図は、**SmallImageResource** プロパティを使用してボタンを小サイズで表示しています。

テンプレート

WPF/Silverlight コントロールを使用する主な利点の1つは、これが自由にカスタマイズできるユーザーインターフェイスを持つ「外観のない」コントロールであることです。Silverlightアプリケーションのユーザーインターフェイスであるルックアンドフィールを独自に設計するのと同様に、**Toolbar for WPF/Silverlight** で管理されるデータに関して独自の UI を提供できます。Extensible Application Markup Language（XAML。「ザムル」と発音する）は、コードを記述することなく独自の UI を設計するための簡単な方法を提供する XML ベースの宣言型言語です。

テンプレートへのアクセス

テンプレートにアクセスするには、Microsoft Expression Blend で、**C1Toolbar** コントロールを選択し、メニューから**[テンプレートの編集]**を選択します。**[コピーして編集]**を選択して現在のテンプレートのコピーを作成して編集するか、**[空アイテムの作成]**を選択して新しい空のテンプレートを作成します。

新しく作成されたテンプレートは、**[オブジェクトとタイムライン]**ウィンドウに表示されます。Template プロパティを使用してテンプレートをカスタマイズできます。

> **メモ**：メニューを使用して新しいテンプレートを作成する場合、テンプレートはそのテンプレートのプロパティに自動的にリンクされます。手作業でテンプレートの XAML を作成する場合は、作成したテンプレートに適切な Template プロパティをリンクする必要があります。

追加のテンプレート

デフォルトテンプレートのほかに、**C1Toolbar** コントロールには追加のテンプレートがいくつかあります。これらの追加テンプ

レートには、Microsoft Expression Blend からもアクセスできます。Blend で **C1Toolbar** コントロールを選択し、メニューから**[追加テンプレートの編集]**を選択します。テンプレートを選択し、**[空アイテムの作成]**を選択します。

ComponentOne ClearStyle 技術

ClearStyle は、WPF/Silverlight コントロールのスタイル設定をすばやく簡単に実行できる新技術です。ClearStyle を使用すると、面倒な XAML テンプレートやスタイルリソースを操作しなくても、コントロールのカスタムスタイルを作成できます。

現在のところ、すべての標準 Silverlightコントロールにテーマを追加するには、スタイルリソーステンプレートを作成する必要があります。Microsoft Visual Studio ではこの処理は困難であるため、Microsoft は、このタスクを簡単に実行できるように Expression Blend を導入しました。ただし、Blend に不慣れであったり、十分な学習時間を取れない開発者にとっては、この2つの環境を行き来することはかなり困難な作業です。デザイナに作業を任せることも考えられますが、デザイナと開発者が XAML ファイルを共有すると、かえって煩雑になる可能性があります。

このような場合に、ClearStyle を使用します。ClearStyle は、Visual Studio を使用して直感的な方法でスタイル設定を実行できるようにします。ほとんどの場合は、アプリケーション内のコントロールに対して単純なスタイル変更を行うだけなので、この処理は簡単に行えるべきです。たとえば、データグリッドの行の色を変更するだけであれば、1つのプロパティを設定するだけで簡単に行えるようにする必要があります。一部の色を変更するためだけに、完全で複雑なテンプレートを作成する必要はありません。

ClearStyle の仕組み

コントロールのスタイルの主な要素は、それぞれ単純な色プロパティとして表されます。これが集まって、コントロール固有のスタイルプロパティセットを形成します。たとえば、**Gauge** には **PointerFill** プロパティや **PointerStroke** プロパティがあり、**DataGrid** の行には **SelectedBrush** や **MouseOverBrush** があります。

たとえば、フォーム上に ClearStyle をサポートしていないコントロールがあるとします。その場合は、ClearStyle によって作成された XAML リソースを使用して、フォーム上の他のコントロールを調整して合わせることができます（正確な色合わせなど）。また、スタイルセットの一部を ClearStyle（カスタムスクロールバーなど）で上書きしたいとします。ClearStyle は拡張可能なのでこれも可能です。必要な場所でスタイルを上書きできます。

ClearStyle は、すばやく簡単にスタイルを変更することを意図したソリューションですが、ComponentOne のコントロールには引き続き従来の方法を使用して、必要なスタイルを細かく指定して作成できます。完全なカスタム設計が必要になる特別な状況で ClearStyle が邪魔になることはありません。

Toolbar の ClearStyle プロパティ

C1Toolbar は、コントロールのテンプレートを変更することなくコントロールの色を簡単に変更できる ComponentOne の新しい ClearStyle 技術をサポートします。色のプロパティをいくつか設定するだけで、**C1Toolbar** 要素のスタイルを簡単に設定できます。次の表に、**C1Toolbar** でサポートされているプロパティを一覧します。

プロパティ	説明
Background	**C1Toolbar** の塗りつぶしに使用される背景を取得または設定します。
MouseOverBrush	マウスポインタが置かれたコントロールを強調表示するために使用されるブラシを取得または設定します。
PressedBrush	クリックされたボタンを強調表示するために使用される System.Windows.Media.Brush を取得または設定します。
FocusBrush	フォーカスがあるコントロールのブラシを取得または設定します。
Header（C1ToolbarGroup のみ）	ヘッダーの描画に使用されるブラシを取得または設定します。

タスク別ヘルプ

タスク別ヘルプは、ユーザーの皆様が Visual Studio.NET のプログラミングに精通しており、C1Toolbar コントロールを使用する方法を理解していることを前提としています。**Toolbar for WPF/Silverlight** 製品を初めて使用される場合は、まず「クイックスタート」を参照してください。

このセクションの各トピックは、**Toolbar for WPF/Silverlight** 製品を使用して特定のタスクを実行するためのソリューションを提供します。

また、タスク別ヘルプトピックは、新しい WPF/Silverlight プロジェクトが既に作成されていることを前提としています。

ツールバーボタンの配置

ItemsPanel 内の表示位置を指定できます。

XAML

```
<c1:C1ToolbarGroup Header="グループ" >
      <c1:C1ToolbarGroup.ItemsPanel>
       <ItemsPanelTemplate>
          <c1:C1ToolbarGroupPanel HorizontalAlignment="Center" />
        </ItemsPanelTemplate>
      </c1:C1ToolbarGroup.ItemsPanel>
```

ツールバーボタンに画像を追加する

LargeImageSource または **SmallImageSource** プロパティを使用して、大きな画像または小さな画像を追加できます。

デザイナの使用

1. ページ内の **C1ToolbarButton** コントロールを右クリックし、**[プロパティ]**を選択します。
2. **LargeImageSource** プロパティを見つけ、そのプロパティの横にある**省略符**ボタンをクリックします。**[画像の選択]**ダイアログボックスが表示されます。
3. **[追加]**ボタンをクリックし、追加する画像を選択したら、**[OK]**をクリックします。

ToolbarButton クリックイベントにロジックを追加する

このトピックでは、XAML 表現を使用して **C1ToolbarButton** の **C1ToolbarButton Click** イベントにロジックを追加し、コードエディタでこのメソッドのコードを追加する方法を示します。

1. XAML エディタで、**C1Toolbar** 要素に **C1ToolbarButton** を追加し、**LabelTitle** プロパティを **Search** に設定します。

   XAML

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

2. XAML エディタで、**C1ToolbarButton** 要素に **Click** という名前の属性を追加し、その値を「**New Click**」に設定します。これは、コードでイベントハンドラに付ける名前になります。また、**C1ToolbarButton** 要素に一意の名前を指定し、その値を「Search」に設定します。

XAML

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

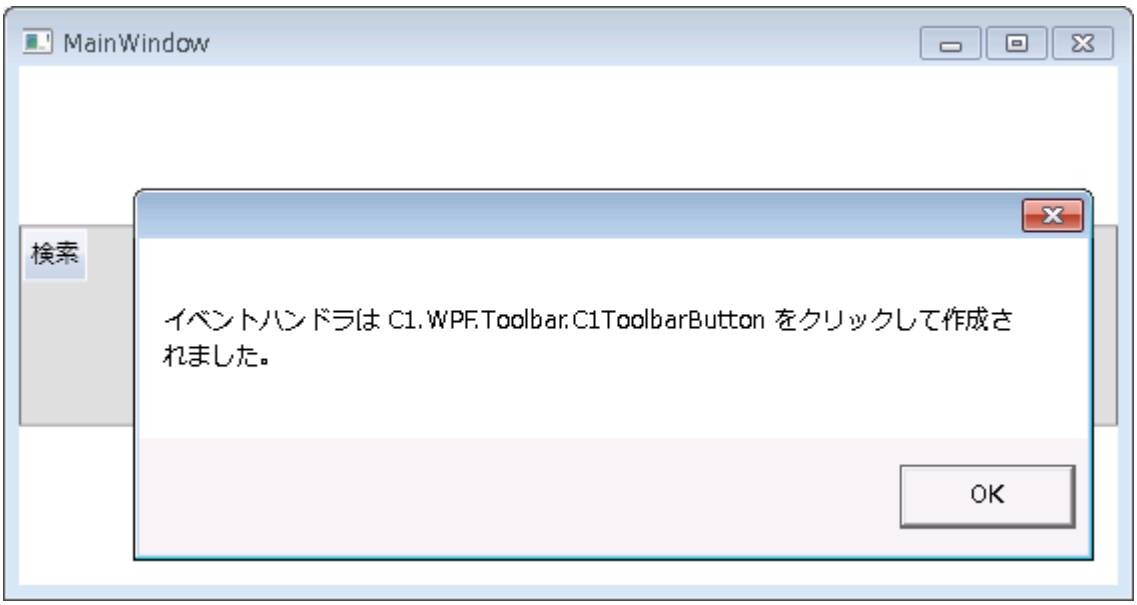
3. デザイナを右クリックし、**[コードの表示]**をクリックします。
4. Window1 クラスに次のイベントハンドラを追加します。このコードは、ボタンがクリックされるたびにメッセージを表示します。

C#

```
private void New_Click(object sender, RoutedEventArgs e)
{
  MessageBox.Show("イベントハンドラは " + Search + " をクリックして作成されました。");
}
```

アプリケーションの実行と動作の確認

メッセージボックスに、ツールバーボタンの名前が表示されます。



コマンドチュートリアル (WPF のみ)

Toolbar for WPF は、コマンドフレームワークをサポートします。WPF では、コマンドにいくつかの目的があります。主な目的は、コマンドを呼び出すオブジェクトを、そのコマンドを実行するロジックから分離することです。コマンドを使用すると、アクションの発生元と処理を緩く結合できるようになります。

次のチュートリアルは、WPF アプリケーションで **C1Toolbar** をコマンドと組み合わせて使用する方法を示します。

> **メモ**:このトピックの内容は、ComponentOne for WPF にのみ適用されます。

1部:コマンドライブラリの使用

WPF は、以下の定義済みコマンドのライブラリを提供します。

- アプリケーションコマンド(Cut、Copy、Paste)
- ナビゲーションコマンド(BrowseBack、BrowseForward)
- メディアコマンド(Play、Stop、Pause)
- 編集コマンド
- コンポーネントコマンド

多くの WPF コントロールには、これらのコマンドが組み込まれているため、コードを記述することなくコマンドを実装できます。

以下の手順は、**C1Toolbar** を使用して、一般的なクリップボードコマンドをいくつか実装する方法を示します。

1. 新しい WPF アプリケーションを開くか作成します。
2. デフォルトの **Grid** 要素に2つの **RowDefinition** を追加し、最初の行の高さを自動にします。次のようになります。

XAML

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

3. 最初の行に挿入する **C1Toolbar** を追加します。
4. 次の XAML を貼り付けて、**C1Toolbar** にいくつかのタブ、グループ、およびボタンを挿入します。

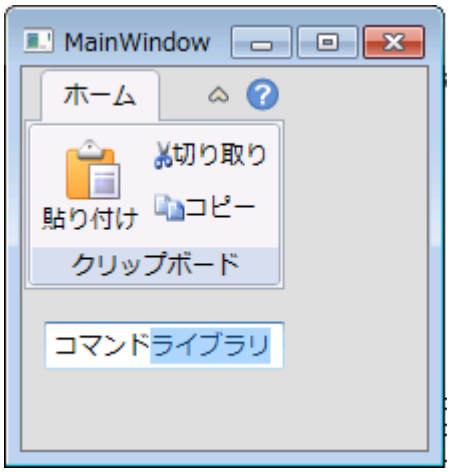
XAML

```
<c1:C1Toolbar Name="c1Toolbar1" FocusManager.IsFocusScope="True">
    <c1:C1ToolbarTabControl>
        <c1:C1ToolbarTabItem Header="Home">
            <c1:C1ToolbarGroup Header="Clipboard">
                <c1:C1ToolbarButton LabelTitle="貼り付け"
Command="ApplicationCommands.Paste"
                LargeImageSource="/Resources/paste.png" />
                <c1:C1ToolbarButton LabelTitle="切り取り"
Command="ApplicationCommands.Cut"
                SmallImageSource="/Resources/cut.png" />
                <c1:C1ToolbarButton LabelTitle="コピー"
```

```
Command="ApplicationCommands.Copy"
                SmallImageSource="/Resources/copy.png" />
            </c1:C1ToolbarGroup>
        </c1:C1ToolbarTabItem>
    </c1:C1ToolbarTabControl>
</c1:C1Toolbar>
```

このマークアップは、**C1Toolbar** 内に1つのグループを含む1つのタブを作成します。**Clipboard** グループには、3つの **C1ToolbarButton** が含まれます。各ボタンには、WPF アプリケーションコマンドライブラリにある一般的なクリップボードコマンドが割り当てられます。**C1Toolbar** の **IsFocusScope** 添付プロパティが **True** に設定されていることに注目してください。これにより、**C1Toolbar** は、その範囲内のフォーカスのある要素を追跡できます。これらの **C1ToolbarButton** には、小さな画像および大きな画像も指定されています。ただし、これらは必須ではありません。

5. **TextBox** コントロールを **C1Toolbar** の下に追加します。
6. アプリケーションを実行し、これらの **C1ToolbarButton** をクリックして、**TextBox** で一般的なクリップボードコマンド（Paste、Copy、Cut）が機能することを確認します。



2部：カスタムコマンドの作成

コマンドライブラリクラス内のコマンドが要件に合わない場合は、独自のコマンドを作成できます。カスタムコマンドを作成するには、ICommand インターフェイスを実装します。WPF は、**RoutedCommand**（および RoutedUICommand）という名前の固有の実装を提供しています。このクラスは、要素ツリーを通じてルーティングされるコマンドを定義します。

以下の手順は、**RoutedCommand** を使用して **C1Toolbar** にカスタムコマンドを追加する方法を示します。

1. 1部と同じサンプルを使用して、分離コードファイルを開きます。
2. **ClearCommand** という名前の RoutedCommand を定義します。このコマンドは、ページ内の **TextBox** からテキストをクリアするために使用されます。

C#

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

3. コマンドロジックを定義するイベントハンドラを作成します。

C#

```
// ClearCommand のロジックを実行します
private void ExecutedClearCommand(object sender, ExecutedRoutedEventArgs e)
{
```

```
    textBox1.Clear();
}
```

4. コマンドを実行できるかどうかを決定するもう1つのイベントハンドラを作成します。コマンドを実行できない場合は、このコマンドに関連付けられているツールバーボタンが淡色表示され、無効になります。

C#

```
// テキストボックスにテキストがある場合にのみ true を返します。
private void CanExecuteClearCommand(object sender, CanExecuteRoutedEventArgs e)
{
    if (textBox1.Text.Length > 0)
    {
        e.CanExecute = true;
    }
    else
    {
        e.CanExecute = false;
    }
}
```

5. 次に、このコマンドをイベントハンドラに関連付ける CommandBinding を作成します。このコマンドが呼び出されると、要素ツリー内で CommandBinding を含むオブジェクトが検索されます。ページの InitializeComponent 呼び出しの後に、次のコードを配置します。

C#

```
CommandBinding customCommandBinding = new CommandBinding(ClearCommand,
ExecutedClearCommand, CanExecuteClearCommand);
// CommandBinding をルート要素にアタッチします
this.CommandBindings.Add(customCommandBinding);
```

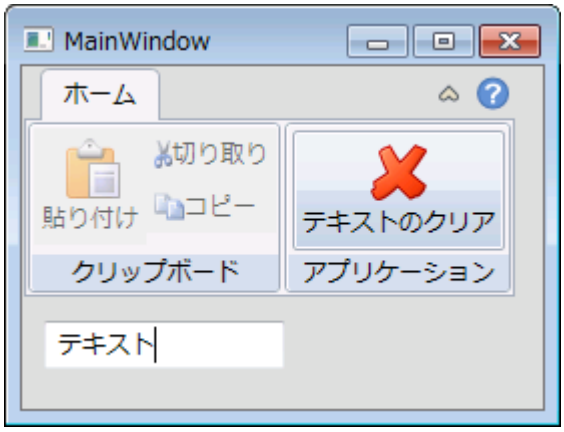
6. **C1Toolbar** を含む XAML で、**「アプリケーション」**という名前のグループに新しい **C1ToolbarButton** を追加します。

7. ページの分離コード内で **Command** プロパティを静的コマンドに設定します。

XAML

```
<c1:C1ToolbarGroup Header="アプリケーション">
    <c1:C1ToolbarButton LabelTitle="テキストのクリア"
    Command="{x:Static local:MainWindow.ClearCommand}"
    LargeImageSource="/Resources/clear.png"/>
</c1:C1ToolbarGroup>
```

8. アプリケーションを実行します。ツールバーボタンをクリックすると、**ClearCommand** が要素ツリーを通じてルーティングされ、関連付けられた **CommandBinding** が検索されます。これが見つかると、関連付けられたイベントハンドラが呼び出され、コマンドロジックが実行されます。



3部：MVVM でのコマンドの使用

UI とビジネスロジックの分離が鍵となる MVVM（Model-View-ViewModel）デザインパターンに、コマンドは必要不可欠です。**Toolbar for WPF** はコマンドフレームワークをサポートしており、一般的な MVVM パターンと組み合わせて使用できます。MVVM 設計のアプリケーションでは、コマンドターゲットが ViewModel で実装されることがよくありますが、これらは UI 要素ツリーに含まれません。したがって、MVVM で ICommand を使用する方法として RoutedCommand は最適な実装ではありません。**DelegateCommand**、**RelayCommand** などの特別な実装を使用する必要があります。これらは、View が UI 要素ツリーに含まれないオブジェクトに連結できるようにします。

以下の手順は、**RelayCommand** クラスと **C1Toolbar** を使用して、2部で実装したカスタムコマンドと同じカスタムコマンドを作成する方法を示します。

1. **INotifyPropertyChanged** インターフェイスを実装する新しいクラス **MainViewModel** を作成します。このクラスは、**C1Toolbar** を含む View の ViewModel として機能します。

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));
        }
    }

}
```

このクラスは、**DelegateCommand** の軽量バージョンである **RelayCommand** を使用します。どちらの ICommand 実装を使用しても、UI 要素ツリーには含まれないコマンドターゲットにパラメータとして渡されるメソッドに、コマンドロジックを委ねることができます。**RelayCommand** クラスと **DelegateCommand** クラスは、WPF フレームワークに含まれていません。これらは、オンラインまたはさまざまな MVVM ツールキットから入手できます。

2. プロジェクトに **RelayCommand** クラスを追加します。

C#

```
/// <summary>
///     このクラスは、パラメータとして渡されるメソッドにコマンドロジックを委ねます。
///     また、View が、要素ツリーに含まれないオブジェクトにコマンドを連結できるようにします。
/// </summary>
public class RelayCommand : ICommand
{
    #region Fields

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

    #endregion // フィールド

    #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 // コンストラクタ

    #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 のメンバ
}
```

3. ViewModel に連結するには、**C1Toolbar** を含む View の先頭に次の XAML を追加します。

XAML

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

これで、ViewModel がリソースとして追加され、Window または UserControl の **DataContext** に連結されます。

4. 続いて、1部で作成した **C1Toolbar** に次のツールバーグループを追加します。

XAML

```
<c1:C1ToolbarGroup Header="アプリケーション">
    <c1:C1ToolbarButton LabelTitle="テキストのクリア"
    Command="{Binding ClearCommand}" LargeImageSource="/Resources/delete.png"/>
</c1:C1ToolbarGroup>
```

このツールバーグループには1つの **C1ToolbarButton** が含まれ、その **Command** プロパティは、ViewModel で定義された **ClearCommand** に連結されます。

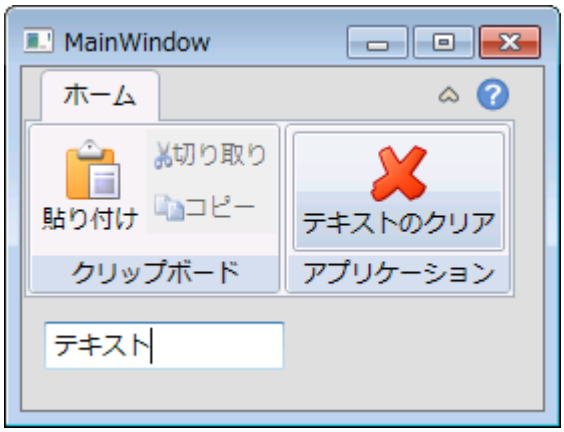
5. このコマンドは MVVM のベストプラクティスに準拠する必要があります。したがって、TextBox にビジネスロジックを適用する場合は、TextBox の **Text** プロパティも ViewModel のある値に連結する必要があります。**Text** プロパティを ViewModel で定義されている **TextValue** プロパティに連結します。

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" />
```

UpdateSourceTrigger を **PropertyChanged** に設定することで、ViewModel の TextValue プロパティは、**TextBox** がフォーカスを失うときだけでなく、**Text** 値が変更されたときにいつも更新されます。

6. アプリケーションを実行し、カスタムな **RelayCommand** の動作を確認します。



MVVM 設計パターンに準拠すると、UI とビジネスロジックを緩く結合できるため、さまざまなメリットを得ることができます。

コマンドチュートリアル (Silverlight のみ)

Silverlight 5 は、**ButtonBase** クラスから派生された任意のオブジェクトで **ICommand** インターフェイスをサポートしています。コントロールの **Command** プロパティを通して、**ICommand** を実装するオブジェクトに、**C1ToolbarButton** などの任意のボタンを関連付けることができます。ICommand インターフェイスはサポートされていますが、Silverlight は組み込みの実装を提供していません。**ComponentOne for Silverlight** には、**C1Command** と **C1ToolbarCommand** の2つの実装が含まれているため、自分で実装を記述する必要はありません。

C1Command クラスは **C1.Silverlight** アセンブリに含まれています。**C1ToolbarCommand** は **C1Command** を拡張し、**C1.Silverlight.Toolbar** アセンブリに含まれています。これには、ツールバーに関連して、ラベルとイメージのプロパティがいくつか追加されています。

以下の手順では、**C1Toolbar** を使用したコマンドの使用方法を示しています。

C1ToolbarCommand の使用

1. 新しい Silverlight アプリケーションを開くか、作成します。
2. デフォルトの **Grid** 要素に2つの **RowDefinition** を追加し、最初の行の高さを自動にします。次のようになります。

XAML

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

3. 最初の行に挿入する **C1Toolbar** を追加します。
4. 2行目に **TextBox** を追加します。
5. 次の **C1ToolbarCommand** をユーザーコントロールリソースに貼り付けます。次のようになります。

XAML

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

このコマンドは、**TextBox** からコンテンツをクリアするために使用されます。**C1ToolbarCommand** には **LargeImageSource** と **SmallImageSource** を指定できますが、必須ではありません。

6. 次の XAML を貼り付けて、**C1Toolbar** にタブ、グループ、ボタンを組み込みます。

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>
```

ここでは、**c1:CommandExtensions.Command** 添付プロパティを前述のコマンドに設定しています。**C1ToolbarButton** には、継承された **Command** プロパティもあることに注意してください。**C1Commands** の使用時には、添付プロパティ **CommandExtensions.Command** を使用してコマンドを設定することをお勧めします。

7. ページ内の “textBox1”という名前のツールバーの下に1つの **TextBox** を追加します。
8. 次に、次のコードを使用して、ページの初期化後にコマンドを登録する必要があります。

C#

```
// コマンドのメソッドを登録します
CommandManager.RegisterClassCommandBinding(GetType(),
new CommandBinding((C1ToolbarCommand)Resources["cmdClear"], Clear, CanClear));
// 登録されたコマンドを実行できるかどうかをチェックします
CommandManager.InvalidateRequerySuggested();
```

C1.Silverlight.CommandManager は、コマンドの登録に関連するユーティリティメソッドを提供します。ここでは、Clear と CanClear のイベントハンドラを渡します。

9. 次の **Clear** イベントハンドラと **CanClear** イベントハンドラを貼り付けます。これらは、コマンドのロジックを実行します。

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;
}
```

10. Silverlight では、登録されたコマンドが機能するかどうかをコードで明示的にチェックする必要があります。**TextBox TextChanged** イベントに次のコードを追加します。

C#

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

11. サンプルを実行します。これで、**C1Toolbar** のコマンドと **C1Command** を使用して、テキストボックスのコンテンツをクリアできるようになりました。