Extended Library for WPF/Silverlight

2018.04.25 更新

グレープシティ株式会社

目次

製品の概要

ComponentOne for WPF/Silverlight Extended Library は、ComponentOne for WPF/Silverlight の一部です。これには、**C1.WPF.Extended.4.dll/C1.Silverlight.Extended.dll** アセンブリ内のすべてのコントロールが含まれます。このアセンブリには、C1.WPF.4.dll/C1.Silverlight.dll アセンブリ内のコントロールほど頻繁には使用されない特殊なコントロールが入っています。

C1.WPF.Extended.4.dll 内のコントロールの多くは、WinForms にはない視覚的表現力に優れたコントロールです。
C1.Silverlight.Extended.dll 内のコントロールの多くは、WinForms または WPF にはない視覚的表現力に優れたコントロールです。

WPF の主要なクラス

C1.WPF.Extended.4.dll アセンブリには、次の主要なクラスが含まれます。

- **C1Accordion**:各項目が **C1Expander** コントロール内に表示され、一度に1つしか展開できない ItemsControl です。これは、Microsoft Outlook のナビゲーションバーに似ています。
- **C1Book**:本の中のページのような UIElement オブジェクトを提供します。一度に2つの要素を表示し、本物の本のページをめくるようにマウスを使用してページをめくることができます。このコントロールは、視覚的な表現力に優れしかもなじみのある、ページめくり効果と影を提供します。
- **C1ColorPicker**:パレットを使用して色を参照したり、RGB カラーモデルや HSB カラーモデルを使用して色を作成できます。
- **C1Expander**:ヘッダーをクリックすることでコンテンツを折りたたんだり展開することができる C1HeaderedContentControl です。
- **C1PropertyGrid**:任意のオブジェクトに使用できる編集可能フォームを提供します。10 以上の組み込みエディタがあり、カスタムエディタをサポートします。また、メソッドもサポートします。

Silverlight の主要なクラス

C1.Silverlight.Extended.dll アセンブリには、次の主要なクラスが含まれます。

- **C1Accordion**:各項目が **C1Expander** コントロール内に表示され、一度に1つしか展開できない ItemsControl です。これは、Microsoft Outlook のナビゲーションバーに似ています。
- **C1Book**:本の中のページのような UIElement オブジェクトを提供します。一度に2つの要素を表示し、本物の本のページをめくるようにマウスを使用してページをめくることができます。このコントロールは、視覚的な表現力に優れしかもなじみのある、ページめくり効果と影を提供します。
- **C1ColorPicker**:パレットを使用して色を参照したり、RGB カラーモデルや HSB カラーモデルに基づいて色を作成できます。
- **C1Expander**:ヘッダーをクリックすることでコンテンツを折りたたんだり展開することができる C1HeaderedContentControl です。
- **C1HtmlHost**:任意の HTML コンテンツをホストできるフレームを提供します。このコントロールは、任意の URI(SourceUri プロパティ)にあるコンテンツを表示したり、HTML ドキュメント(SourceHtml プロパティ)を表示することができます。

  > **メモ**:**C1HtmlHost** コントロールでは、ホストの Silverlight プラグインの **Windowless** プロパティが **True** に設定されている必要があります。

- **C1PropertyGrid**:任意のオブジェクトに使用できる編集可能フォームを提供します。10 以上の組み込みエディタがあり、カスタムエディタをサポートします。また、メソッドもサポートします。さらに、添付プロパティもサポートします。

- **C1CoverFlow**:選択可能な項目の 3D アニメーション表示を提供します(任意の UIElement をサポート)。Apple iTunes® アプリケーションに似ています。
- **C1Reflector**:コンテンツにリアルな 3D リフレクションを加える ContentControl です(任意の UIElement をサポート)。

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。

ComponentOne for WPF/Silverlight のヘルプ

はじめに

ComponentOne for WPF/Silverlight のすべてのコンポーネントで共通の使用方法については、「ComponentOne for WPF/Silverlight ユーザーガイド」を参照してください。

Accordion

Accordion for WPF/Silverlight を使用して、展開可能な項目のリストを表示します。項目を選択すると、その項目が展開され、他の項目が折りたたまれます。したがって、UI を自動的に整理し、画面を有効に使用できます。

操作

C1Accordion コントロールは、テキスト、画像、およびコントロールを格納する一連の展開/折りたたみ可能なペインを保持できるコンテナです。**C1Accordion** コントロールは **ItemsControl** の1つです。つまり、このコントロールは、一連のオブジェクトをホストするように設計されています。**C1AccordionItem** クラスは、**C1Accordion** コントロールによってホストされる項目（アコーディオンペイン）を表します。

プロジェクトに追加された **C1Accordion** コントロールは、それだけでは単なるコンテナです。ただし、このコントロールをプロジェクトに追加したら、Blend、XAML、またはコードを使用して、このコントロールに複数のアコーディオンペインを簡単に追加できます。次の画像に示す **C1Accordion** コントロールには、3つのアコーディオンペイン項目を含まれ、最初のペインが展開されています。

上の画像では、アコーディオンペインにヘッダーテキストやコンテンツが含まれていませんが、ヘッダーをカスタマイズしたり、コンテンツをペインに追加する作業は、いくつかのプロパティを設定するように簡単に実行できます。また、コントロールの展開可能性や方向などの動作を変更することもできます。

以下のトピックでは、**C1Accordion** コントロールの要素および機能について説明します。

主な特長

Accordion for WPF/Silverlight を使用して、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。以下の主要な機能をうまく利用して、**Accordion for WPF/Silverlight** を最大限に活用してください。

- **展開方向**
  C1Accordion コントロールは、4方向に展開できます。**ExpandDirection** プロパティでコントロールの展開方向を指定します。**Top**、**Right**、**Bottom**、または Left を設定できます。詳細については、**「展開方向」**トピックを参照してください。

- **カスタムヘッダー**
  アコーディオンペインのヘッダーは、テキストとコントロールの両方を使用してカスタマイズできます。

- **項目の整理方法の設定**
  Accordion を使用すると、スペースを最大限に活用できます。**C1Accordion** のサイズと位置を設定し、必要になるまで項目を非表示にできます。

- **任意のデータ型のオブジェクトの追加**
  C1Accordion コントロールは **ItemsControl** を継承するため、このコントロールの Items コレクションに任意のデータ型のオブジェクトを追加できます。また、**DataTemplate** を使用して、項目のビジュアル表現を作成できます。

XAML クイックリファレンス

このトピックでは、**C1Accordion** 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。詳細については、「タスク別ヘルプ」セクションを参照してください。

C1AccordionItem から成る C1Accordion

次の XAML マークアップは、3つの **C1AccordionItem** を含む **C1Accordion** コントロールを作成します。

XAML

```
<c1:C1Accordion Height="100" HorizontalAlignment="Left" Name="c1Accordion1"
VerticalAlignment="Top" Width="200">
     <c1:C1AccordionItem IsTabStop="False" />
     <c1:C1AccordionItem IsTabStop="False" />
     <c1:C1AccordionItem IsTabStop="False" />
</c1:C1Accordion>
```

C1AccordionItem のコンテンツ

次の XAML マークアップは、3つの **C1AccordionItem** を含む **C1Accordion** コントロールを作成します。これは、**C1AccordionItem** から成る **C1Accordion** のマークアップと似ていますが、今回は各 C1AccordionItem にコンテンツがあります。最初の項目には **TextBlock**、2番目の項目には **Button** コントロール、3番目の項目には **Calendar** コントロールが含まれます。

XAML

```
<c1:C1Accordion Height="276" HorizontalAlignment="Left" Name="c1Accordion1"
VerticalAlignment="Top" Width="355">
    <c1:C1AccordionItem IsTabStop="False">
       <TextBlock Height="23" HorizontalAlignment="Left" Margin="10,10,0,0"
Name="textBlock1" Text="TextBlock" VerticalAlignment="Top">Hello world!</TextBlock>
    </c1:C1AccordionItem>
    <c1:C1AccordionItem IsTabStop="False">
        <Button Content="ボタン" Height="23" HorizontalAlignment="Left"
Margin="10,10,0,0" Name="button1" VerticalAlignment="Top" Width="75" />
     </c1:C1AccordionItem>
     <c1:C1AccordionItem IsTabStop="False">
         <sdk:Calendar Height="169" HorizontalAlignment="Left" Margin="15,15,0,0"
Name="calendar1" VerticalAlignment="Top" Width="230" />
     </c1:C1AccordionItem>
</c1:C1Accordion>
```

Accordion for WPF クイックスタート

このクイックスタートは、**Accordion for WPF** 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、最初に Visual Studio で **C1Accordion** コントロールを含む新しいプロジェクトを作成します。また、アコーディオンをカスタマイズし、アコーディオンペインを追加し、そこにコンテンツを追加してから、コントロールの実行時機能をいくつか使用してみます。

手順 1：アプリケーションの作成

この手順では、最初に **Visual Studio** で **Accordion for WPF** を使用する WPF アプリケーションを作成します。

次の手順に従います。

1. Visual Studio で新しい WPF プロジェクトを作成します。
2. ツールボックスに移動し、[**C1Accordion**]アイコンをダブルクリックして、コントロールをプロジェクトに追加します。
3. C1Accordion コントロールを右クリックしてコンテキストメニューを開き、[プロパティ]を選択します。[プロパティ]ウィンドウが開き、**C1Accordion** コントロールのプロパティが表示されます。
4. 次の**プロパティ**を設定します。
    - **Height** プロパティを「250」に設定して、コントロールの高さを設定します。
    - **Width** プロパティを「400」に設定して、コントロールの幅を設定します。
    - ExpandDirection プロパティを Left に設定します。これにより、**C1Accordion** コントロールは、デフォルトの上からではなく下から展開されます。
    - [Fill]チェックボックスをオンにし、**Fill** プロパティを **True** に設定します。これにより、各ペインが **C1Accordion** コントロールの指定された幅に合わせて展開されます。
    - **[AllowCollapseAll]**チェックボックスをオフにし、**AllowCollapseAll** を **False** に設定します。これにより、すべてのペインを同時に折りたたむことができなくなります。

これで、**C1Accordion** コントロールを含む WPF アプリケーションを作成できました。次の手順では、**C1Accordion** コントロールの外観と動作をカスタマイズします。

手順 2:アコーディオンペインの追加

前の手順では、カスタマイズされた **C1Accordion** コントロールを含むプロジェクトを作成しました。この手順では、いくつかのアコーディオンペインを追加します。これらのペインは、後でカスタマイズし、コンテンツを追加します。

次の手順に従います。

1. **C1Accordion** コントロールをクリックして選択します。
2. [プロパティ]ウィンドウで、[項目]の省略符ボタンをクリックします。
[コレクション エディター:Items]ダイアログボックスが開きます。
3. [追加]ボタンを3回クリックして、3つの **C1AccordionItem** 項目を **C1Accordion** コントロールに追加します。
4. 最初の[C1AccordionItem]を選択し、次のプロパティを設定します。
    - **Background** プロパティを **Aqua** に設定し、アコーディオンペインの背景色を設定します。
    - **Content** プロパティを「これはコンテンツ領域です。」に設定し、アコーディオンペインにテキストコンテンツを追加します。
5. 2番目の**[C1AccordionItem]**を選択し、次のプロパティを設定します。
    - **Background** プロパティを **AliceBlue** に設定し、アコーディオンパネルの背景色を設定します。
    - **IsExpanded** プロパティを **True** に設定します。これにより、このペインは実行時に展開されます。
6. 3番目の**[C1AccordionItem]**を選択し、**Background** プロパティを **LawnGreen** に設定して、アコーディオンペインの背景色を設定します。
7. [OK]をクリックして、[コレクション エディター:Items]ダイアログボックスを閉じます。
8. XAML ビューに切り替えて、次の手順に従います。
    - **Header**="ペイン1" を最初の <c1:C1AccordionItem> タグに追加します。

- **Header**="ペイン2" を2番目の <c1:C1AccordionItem> タグに追加します。
- **Header**="ペイン3" を3番目の <c1:C1AccordionItem> タグに追加します。

9. デザインビューに切り替え、次の手順に従って、2番目のアコーディオンペインにコントロールを追加します。
   a. デザイナで、2番目のアコーディオンペインをクリックして選択します。
   b. ツールボックスに移動し、**[カレンダ]**アイコンをダブルクリックして、アコーディオンペインにコントロールを追加します。

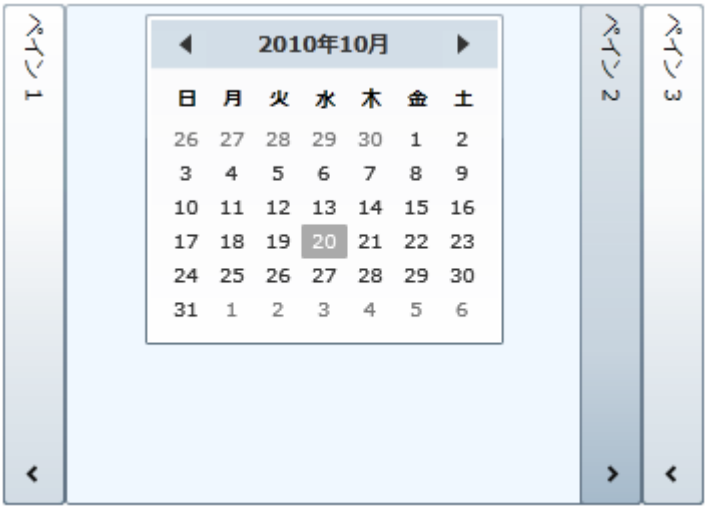
この手順では、**C1Accordion** コントロールに3つのアコーディオンペインを追加し、2つのアコーディオンペインにコンテンツを追加しました。次の手順では、プロジェクトを実行し、コントロールの実行時機能を確認します。

手順 3:プロジェクトの実行

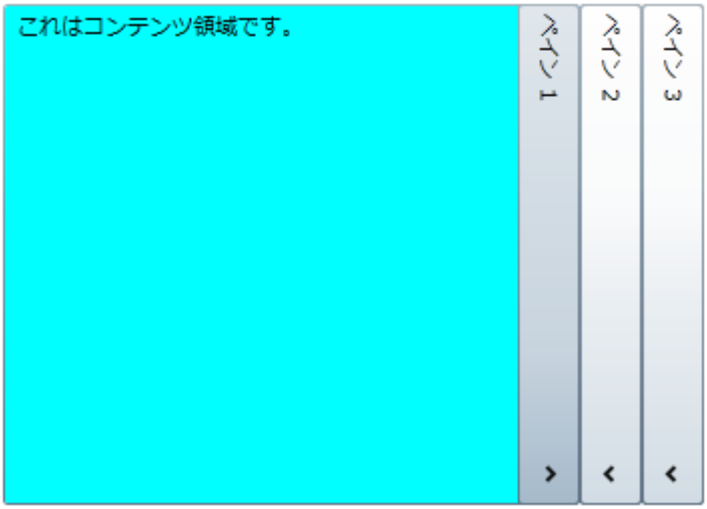
前の手順では、**C1Accordion** コントロールにアコーディオンペインとコンテンツを追加しました。この手順では、プロジェクトを実行し、**C1Accordion** コントロールの実行時機能をいくつか確認します。

次の手順に従います。

1. [デバッグ]メニューから[デバッグ開始]を選択し、実行時にアプリケーションがどのように表示されるかを確認します。**Calendar** コントロールを保持する2番目のペインが展開されていることを確認します。



2. [ペイン1]をクリックし、最初のペインが展開され、コンテンツが表示されることを確認します。



3. [ペイン3]をクリックし、最後のペインが展開されることを確認します。3番目のペインにはコンテンツが含まれていませんが、他のペインと同じ幅で展開されることがわかります。

4. [ペイン3]をクリックし、ペインを閉じることができないことを確認します。これは、**AllowCollapseAll** プロパティが False に設定されているためです。このため、1つのアコーディオンペインが常に展開されている状態になります。

おめでとうございます!

これで、**Accordion for WPF** クイックスタートは終了です。このチュートリアルでは、**C1Accordion** コントロールを含む WPF プロジェクトを作成し、コントロールの外観と動作を変更しました。さらに、コントロールにアコーディオンペインとそのコンテンツを追加し、コントロールの実行時機能をいくつか確認しました。

Accordion for Silverlight クイックスタート

このクイックスタートは、**Accordion for 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、最初に Visual Studio で **C1Accordion** コントロールを含む新しいプロジェクトを作成します。また、アコーディオンをカスタマイズし、アコーディオンペインを追加し、そこにコンテンツを追加してから、コントロールの実行時機能をいくつか使用してみます。

手順 1:アプリケーションの作成

この手順では、最初に **Visual Studio** で **Accordion for Silverlight** を使用する Silverlight アプリケーションを作成します。

次の手順に従います。

1. Visual Studio で、**[ファイル]**→**[新しいプロジェクト]**を選択します。
2. **[新しいプロジェクト]**ダイアログボックスの左ペインから言語を選択し、テンプレートリストから[Silverlight アプリケーション]を選択します。プロジェクトの**名前**を入力し、**[OK]**をクリックします。[新しい Silverlight アプリケーション]ダイアログボックスが表示されます。
3. **[OK]**をクリックすると、**[新しい Silverlight アプリケーション]**ダイアログボックスが閉じ、プロジェクトが作成されます。
4. プロジェクトの XAML ウィンドウで、<UserControl> タグの **DesignWidth="400" DesignHeight="300"** を **DesignWidth="Auto" DesignHeight="Auto"** に変更して、UserControl をサイズ変更します。次のようになります。

XAML

```
<UserControl x:Class="SilverlightApplication1.MainPage"
  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
mc:Ignorable="d" d:DesignWidth="Auto" d:DesignHeight="Auto">
```

これで、**UserControl** は、中に置かれた内容に合わせてサイズ変更されるようになります。

5. プロジェクトの XAML ウィンドウで、カーソルを <Grid> タグと </Grid> タグの間に置き、1回クリックします。
6. ツールボックスに移動し、**C1Accordion** アイコンをダブルクリックして、コントロールをグリッドに追加します。XAML マークアップは次のようになります。

XAML

```
<Grid x:Name="LayoutRoot">
    <c1:C1Accordion></c1:C1Accordion>
</Grid>
```

これで、**C1Accordion** コントロールを含む Silverlight アプリケーションを作成できました。次の手順では、**C1Accordion クラス**C1Accordion コントロールの外観と動作をカスタマイズします。

手順 2:コントロールの設定

前の手順では、Silverlight プロジェクトを作成し、**C1Accordion** コントロールを追加しました。この手順では、**C1Accordion** コントロールの動作と外観をカスタマイズします。

次の手順に従います。

1. <c1:C1Accordion> タグに **Height="250"** を追加して、コントロールの高さを設定します。XAML マークアップは次のようになります。

XAML

```
<c1:C1Accordion Height="250">
```

2. <c1:C1Accordion> タグに **Width="400"** を追加して、コントロールの幅を設定します。XAML マークアップは次のようになります。

XAML

```
<c1:C1Accordion Height="250" Width="400">
```

3. <c1:C1Accordion> タグに **ExpandDirection="Left"** を追加します。これにより、**C1Accordion** コントロールが上からではなく下から展開されます。これはデフォルトです。XAML マークアップは次のようになります。

XAML

```
<c1:C1Accordion Height="250" Width="400" ExpandDirection="Left">
```

4. <c1:C1Accordion> タグに **Fill="True"** を追加します。これにより、各ペインの幅は、**C1Accordion** コントロールに指定された幅と同じになります。XAML マークアップは次のようになります。

XAML

```
<c1:C1Accordion Height="250" Width="400" ExpandDirection="Left" Fill="True">
```

5. <c1:C1Accordion> タグに **AllowCollapseAll="False"** を追加します。これにより、すべてのペインを同時に折りたた

むことができなくなります。XAML マークアップは次のようになります。

```
XAML
<c1:C1Accordion Height="250" Width="400" ExpandDirection="Left" Fill="True"
AllowCollapseAll="False">
```

この手順では、**C1Accordion** コントロールの外観と動作をカスタマイズしました。次の手順では、このコントロールにカスタマイズしたアコーディオンペインをいくつか追加します。

手順 3：アコーディオンペインの追加

前の手順では、**C1Accordion** コントロールの外観と動作をカスタマイズしました。この手順では、いくつかのアコーディオンペインを追加します。これらのペインは、後でカスタマイズし、コンテンツを追加します。

次の手順に従います。

1. カーソルを <c1:C1Accordion> タグと </c1:C1Accordion> タグの間に置き、[Enter]キーを押します。
2. ツールバーに移動し、**C1AccordionItem** アイコンをクリックして、コントロールにアコーディオンペインを追加します。**C1Accordion** コントロールに合計3つの **C1AccordionItem** が追加されるように、この手順をもう2回繰り返します。XAML マークアップは次のようになります。

```
XAML
<c1:C1AccordionItem></c1:C1AccordionItem>
<c1:C1AccordionItem></c1:C1AccordionItem>
<c1:C1AccordionItem></c1:C1AccordionItem>
```

3. **Header**="**Pane 1**" を最初の <c1:C1AccordionItem> タグに、**Header**="**Pane 2**" を2番目の <c1:C1AccordionItem> タグに、**Header**="**Pane 3**" を3番目の <c1:C1AccordionItem> タグに追加します。XAML マークアップは次のようになります。

```
XAML
<c1:C1AccordionItem Header="ペイン 1"></c1:C1AccordionItem>
<c1:C1AccordionItem Header="ペイン 2"></c1:C1AccordionItem>
<c1:C1AccordionItem Header="ペイン 3"></c1:C1AccordionItem>
```

4. **Background**="**Aqua**" を最初の <c1:C1AccordionItem> タグに、**Background**="**AliceBlue**" を2番目の <c1:C1AccordionItem> タグに、**Background**="**LawnGreen**" を3番目の <c1:C1AccordionItem> タグに追加します。XAML マークアップは次のようになります。

```
XAML
<c1:C1AccordionItem Header="ペイン 1" Background="Aqua"></c1:C1AccordionItem>
<c1:C1AccordionItem Header="ペイン 2" Background="AliceBlue"></c1:C1AccordionItem>
<c1:C1AccordionItem Header="ペイン 3" Background="LawnGreen"></c1:C1AccordionItem>
```

5. 次の手順に従って、最初の2つのアコーディオンペインにコンテンツを追加します。
   a. **Content**="**テキスト**" を最初の <c1:C1AccordionItem> タグに追加します。XAML は次のようになります。
   <c1:C1AccordionItem Header="ペイン 1" Background="Aqua" **Content**="**テキスト**">
   b. カーソルを2組目の <c1:C1AccordionItem> タグと </c1:C1AccordionItem> タグの間に置き、[Enter]キーを押します。

c. ツールボックスに移動し、**[Calendar]**アイコンをダブルクリックして、**Calendar** コントロールを2番目のペインのコンテンツとして追加します。XAML は次のようになります。

XAML

```
<c1:C1AccordionItem Header="ペイン 2" Background="AliceBlue">
    <sdk:Calendar></sdk:Calender>
</c1:C1AccordionItem>
```

6. **IsExpanded**="**True**" を2番目の <c1:C1AccordionItem> タグに追加します。XAML は次のようになります。

XAML

```
<c1:C1AccordionItem Header="ペイン 2" Background="AliceBlue" IsExpanded="True">
```

これにより、**Calendar** コントロールを保持する2番目のアコーディオンペインが実行時に展開されます。

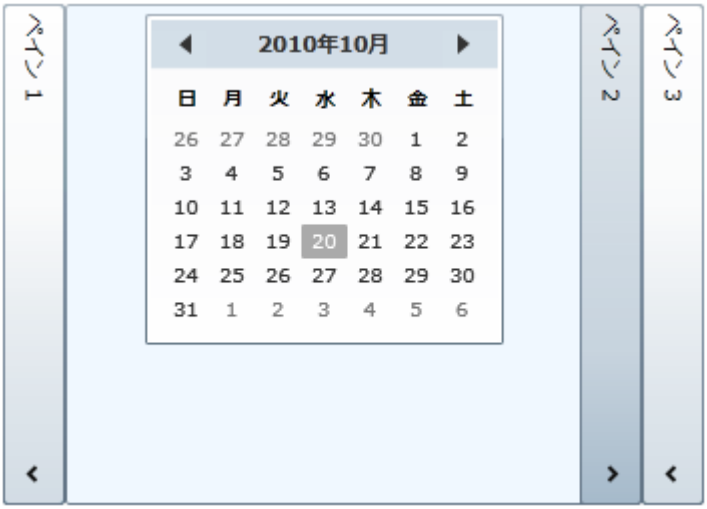
この手順では、**C1Accordion** コントロールに3つのアコーディオンペインを追加し、2つのアコーディオンペインにコンテンツを追加しました。次の手順では、プロジェクトを実行し、コントロールの実行時の機能を確認します。

手順 4:アプリケーションの実行

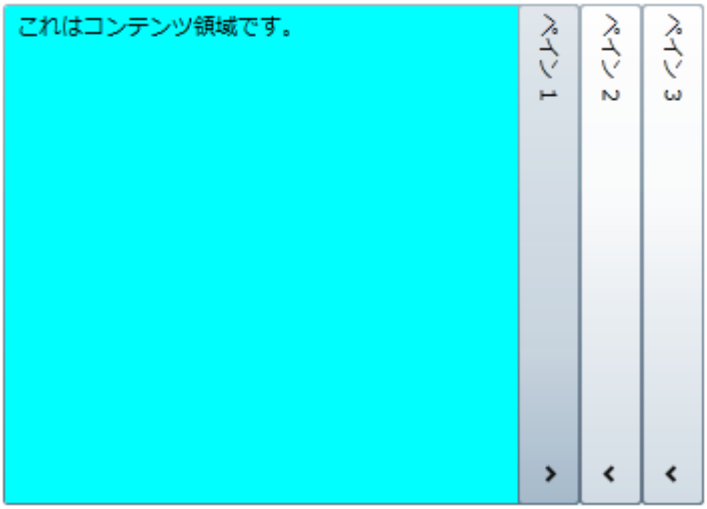
前の手順では、**C1Accordion** コントロールにアコーディオンペインとコンテンツを追加しました。この手順では、プロジェクトを実行し、**C1Accordion** コントロールの実行時機能をいくつか確認します。

次の手順に従います。

1. [デバッグ]メニューから**[デバッグ開始]**を選択し、実行時にアプリケーションがどのように表示されるかを確認します。**Calendar** コントロールを保持する2番目のペインが展開されていることを確認します。



2. **[ペイン 1]**をクリックし、最初のペインが展開され、コンテンツが表示されることを確認します。



3. **[ペイン 3]**をクリックし、最後のペインが展開されることを確認します。3番目のペインにはコンテンツが含まれていませんが、他のペインと同じ幅で展開されることがわかります。

4. **[ペイン 3]**をクリックし、ペインを閉じることができないことを確認します。これは、**AllowCollapseAll** プロパティが False に設定されているためです。このため、1つのアコーディオンペインが常に展開されている状態になります。

おめでとうございます!

これで、**Accordion for Silverlight** クイックスタートチュートリアルは終了です。このチュートリアルでは、**C1Accordion** コントロールを含む Silverlight プロジェクトを作成し、コントロールの外観と動作を変更しました。さらに、コントロールにアコーディオンペインとそのコンテンツを追加し、コントロールの実行時機能をいくつか確認しました。

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 のプログラミングに精通しており、**C1Accordion** 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。Accordion for WPF/Silverlight 製品を初めて使用される場合は、まず「Accordion for WPF クイックスタート/ Accordion for Silverlight 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、**Accordion for WPF/Silverlight** 製品を使用して特定のタスクを行うための方法を提供します。

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。

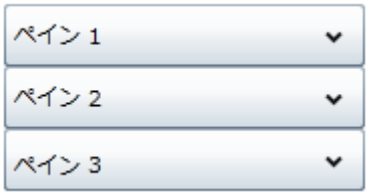
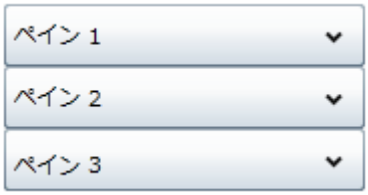
展開と折りたたみ

このセクションでは、アコーディオンペインの展開/折りたたみ方法をカスタマイズするオプションについて説明します。

初期展開状態

デフォルトでは、各アコーディオンペインの **IsExpanded** プロパティは **False** に設定されます。これにより、ページがロードされると、ペインは折りたたまれた状態で表示されます。ページのロード時にペインを展開した状態にする場合は、**IsExpanded** プロパティを **True** に設定します。

次の表に、この2つの展開状態の違いを示します。

IsExpanded	結果
IsExpanded=False	
IsExpanded=True	

展開状態は、デザインビュー、XAML、またはコードで設定できます。

展開方向

C1Accordion コントロールには、**ExpandDirection** プロパティを使用して展開方向を指定するオプションがあります。コントロールの展開方向を設定するほかに、**ExpandDirection** を変更すると、コントロールのコンテンツ領域に対するヘッダーの方向も変更されます。デフォルトでは、**ExpandDirection** プロパティは **Down** に設定され、コントロールは上から下に展開されます。

次の表に、**ExpandDirection** のそれぞれの設定状態を示します。

ExpandDirection	結果
Down	

Up	
Right	
Left	

折りたたみ/展開方向は、Blend、XAML、またはコードで設定できます。

折りたたみ

デフォルトでは、**C1Accordion** コントロールの **AllowCollapseAll** プロパティは True に設定されます。これにより、アコーディオンのすべてのペインを折りたたむことができます。これは、ユーザーの目から余分な項目を隠し、画面を整理して有効に利用するために便利です。楽器のアコーディオンを思い浮かべてみてください。保管時には蛇腹を閉じておきますが、演奏時には蛇腹を開いて使用します。

ただし、これが目的のユーザーインターフェイスの設計には適さない場合もあります。たとえば、アコーディオンをメニュー要素として使用する場合です。この場合は、**C1Accordion** コントロールを常に一定の高さまたは幅にしておきたいですが、すべてのペインを折りたたむことができると、この要件に合いません。したがって、1つのペインが常に展開された状態になるように、**AllowCollapseAll** プロパティを **False** に設定します。次に、目的の高さまたは幅のプロパティを設定してから、**Fill** プロパティを **True** に設定します。これで、**C1Accordion** コントロールとペインは、常に指定された高さまたは幅に合わせて展開されます。

メモ:**C1Accordion** コントロールは、一度に1つのペインしか展開することはできません。

展開のサイズ

アコーディオンペイン(**C1AccordionItem**)のデフォルトの動作では、コンテンツの高さ(上下に展開する場合)または幅(左右に展開する場合)に合わせてペインが展開されます。このため、それぞれのペインのコンテンツの高さまたは幅に応じて、ア

コーディオンの高さまたは幅が異なってしまいます。次の図に、この例を示します。

これを回避するには、**C1Accordion** コントロールの高さまたは幅を設定してから、**Fill** プロパティを **True** に設定します。これで、各パネルが指定された幅または高さに合わせて展開され、アコーディオンペインの幅または高さが一定になります。次の図で、この違いを確認できます。

アコーディオンペインを追加する

このトピックでは、デザインビュー、XAML、およびコードで、**C1Accordion** コントロールにアコーディオンペインを追加します。

デザインビューでの設計時

C1Accordion コントロールにペインを追加するには、次の手順に従います。

1. **C1Accordion** コントロールをクリックして選択します。
2. **[プロパティ]**ウィンドウで、**[項目]**の省略符ボタンをクリックします。
3. **[コレクション エディター:Items]**ダイアログボックスが開きます。
4. **[追加]**ボタンを1回クリックして、1つの **C1AccordionItem** 項目を **C1Accordion** コントロールに追加します。
5. [プロパティ]グリッドで、**Width** プロパティを**「150」**に設定します。

XAML の場合

C1Accordion コントロールにペインを追加するには、<c1ext:C1Accordion> タグと </c1ext:C1Accordion> タグの間に次のマークアップを追加します。

XAML

```
<c1ext:C1AccordionItem Name="C1AccordionItem" Width="150">
</c1ext:C1AccordionItem>
```

コードの場合

コードでアコーディオンペインを追加するには、次の手順に従います。

1. コードビューに切り替えて、次の名前空間をインポートします。

VisualBasic

```
Imports C1.WPF.Extended
```

C#

```
using C1.WPF.Extended;
```

2. **InitializeComponent()** メソッドの下に次のコードを追加します。

VisualBasic

```
'アコーディオンペインを作成し、コンテンツを追加します
Dim C1AccordionItem1 As New C1AccordionItem()
C1AccordionItem1.Content = "C1AccordionItem1"
'C1Accordion コントロールにアコーディオンペインを追加します
C1Accordion1.Items.Add(C1AccordionItem1)
```

C#

```
//アコーディオンペインを作成し、コンテンツを追加します
C1AccordionItem C1AccordionItem1 = new C1AccordionItem();
C1AccordionItem1.Content = "c1AccordionItem1";
//C1Accordion コントロールにアコーディオンペインを追加します
c1Accordion1.Items.Add(c1AccordionItem1);
```

3. プログラムを実行します。

ヘッダー要素へコンテンツを追加する

アコーディオンペインのヘッダーには、単純なテキストと WPF/Silverlight コントロールの両方を簡単に追加できます。このセクションの各トピックは、ヘッダーにテキストコンテンツおよびコントロールを追加するための手順を提供します。

ヘッダーへテキストを追加する

デフォルトでは、アコーディオンペインのヘッダーは空になります。コントロールのヘッダーにテキストを追加するには、デザインビュー、XAML、またはコードで **Header** プロパティに文字列を設定します。

このトピックは、少なくとも1つの C1AccordionItem 項目を含む C1Accordion コントロールがプロジェクトに追加されていることを前提としています。

XAML の場合

XAML で **Header** プロパティを設定するには、Header="Hello World" を <c1ext:C1AccordionItem> タグに追加します。次のようになります。

<c1ext:C1AccordionItem Name="C1AccordionItem1" **Header**=**"Hello World"** Width="150" Height="55">

コードの場合

コードで **Header** プロパティを設定するには、次の手順に従います。

1. コードビューに切り替えて、InitializeComponent() メソッドの下に次のコードを追加します。

Visual Basic

```
C1AccordionItem1.Content = "Hello World"
```

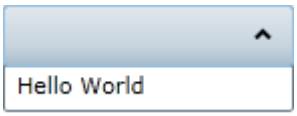
C#

```
c1AccordionItem1.Content = "Hello World";
```

2. プログラムを実行します。

このトピックの作業結果

アコーディオンペインのコンテンツに「Hello World」が表示されます。このトピックの結果は、次のようになります。



ヘッダーへコントロールを追加する

アコーディオンペインのヘッダー要素は、WPF/Silverlight コントロールを受け入れることができます。このトピックでは、XAML およびコードを使用して、ヘッダーに **Button** コントロールを追加します。このトピックは、少なくとも1つの **C1AccordionItem** 項目を含む **C1Accordion** コントロールがプロジェクトに追加されていることを前提としています。

XAML の場合

XAML で **Button** をヘッダーに追加するには、<c1:C1AccordionItem> タグと </c1:C1AccordionItem> タグの間に、次の XAML マークアップを配置します。

XAML

```
<c1:C1AccordionItem.Header>
    <Button Content="ボタン" Height="Auto" Width="50"/>
</c1:C1AccordionItem.Header>
```

コードの場合

コードで **Button** コントロールをヘッダーに追加するには、次の手順に従います。

1. x:Name="C1AccordionItem1" を **<c1:C1Accordion>** タグに追加します。これにより、このオブジェクトをコード内で呼び出すための一意の識別子が指定されます。
2. コードビューに切り替えて、**InitializeComponent()** メソッドの下に次のコードを追加します。

VisualBasic

```
' Button コントロールを作成します
Dim NewButton As New Button()
NewButton.Content = "ボタン"
' Button コントロールの Width および Height プロパティを設定します
NewButton.Width = 50
NewButton.Height = Double.NaN
' ボタンをヘッダーに追加します
C1AccordionItem1.Header = (NewButton)
```

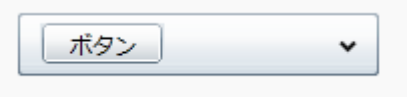
C#

```
InitializeComponent();
// Button コントロールを作成します
Button NewButton = new Button();
NewButton.Content = "ボタン";
// Button コントロールの Width および Height プロパティを設定します
NewButton.Width = 50;
NewButton.Height = Double.NaN;
// ボタンをヘッダーに追加します
C1AccordionItem1.Header = (NewButton);
```

3. プログラムを実行します。

このトピックの作業結果

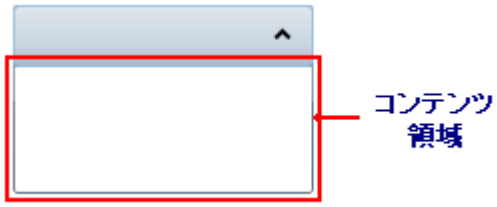
ヘッダー内に **Button** コントロールが表示されます。最終的な結果は、次の画像のようになります。



コンテンツ領域

最初、アコーディオンペインのコンテンツ領域は空です。コンテンツ領域には、グリッド、テキスト、画像、および任意のコントロールを追加できます。Blend または Visual Studio のデザインビューでは、簡単なドラッグアンドドロップ操作を使用して、コントロールのコンテンツ領域に要素を追加したり、コントロール内の要素を移動することができます。

次の画像は、アコーディオンペインのコンテンツ領域を示します。



コンテンツ領域にテキストを追加するには、項目の **Content** プロパティを設定するか、**TextBox** 要素をコンテンツ領域に追加します。実行時にコンテンツ領域に WPF 要素を追加することは簡単です。Visual Studio または Blend では、簡単なドラッグアンドドロップ操作または XAML のいずれかを使用できます。実行時にコントロールを追加する場合は、C# または Visual Basic コードを使用できます。

C1AccordionItem 項目は、子要素を一度に1つだけ受け入れることができます。ただし、この問題は、パネルベースのコントロールを子要素として追加することで回避できます。**StackPanel** コントロールなどのパネルベースのコントロールは、複数の要素を保持できます。パネルベースのコントロール自身が複数の要素を保持できるため、これを使用すると、C1AccordionItem 項目はコントロールを1つだけ保持できるという制限を満たしつつ、アコーディオンペインのコンテンツ領域に複数のコントロールを表示できます。

属性構文とプロパティ要素構文

コンテンツ領域に単純な要素（書式設定されていない文字列、1つのコントロールなど）を追加する場合は、次に示すように、XAML マークアップの一般的な XML 属性を使用できます。

<c1ext:C1AccordionItem **Content="Hello World"/**>

ただし、コンテンツ領域に、グリッドやパネルなどの複雑な要素を追加することもできます。このような場合は、次に示すように、プロパティ要素構文を使用できます。

XAML

```
<c1ext:C1AccordionItem Width="150" Height="55" Name="C1AccordionItem1">
            <c1ext:C1AccordionItem.Content>
                <StackPanel>
                    <TextBlock Text="Hello"/>
                    <TextBlock Text="World"/>
                </StackPanel>
            </c1ext:C1AccordionItem.Content>
</c1ext:C1AccordionItem>
```

コンテンツ領域へコンテンツを追加する

アコーディオンペインのヘッダーには、単純なテキストと WPF/Silverlight コントロールの両方を簡単に追加できます。このセクションの各トピックは、ヘッダーにテキストコンテンツおよびコントロールを追加するための手順を提供します。

コンテンツ領域へテキストを追加する

アコーディオンペインのコンテンツ領域に単純なテキスト行を簡単に追加するには、デザインビュー、XAML、またはコードで Content プロパティに文字列を設定します。

このトピックは、少なくとも1つの **C1AccordionItem** 項目を含む **C1Accordion** コントロールがプロジェクトに追加されていることを前提としています。

メモ:コンテンツ領域に **TextBox** コントロールを追加してから、**TextBox** コントロールの **Text** プロパティを設定することで、コンテンツ領域にテキストを追加することもできます。コンテンツ領域にコントロールを追加する方法については、「コンテンツ領域へコンテンツを追加する」を参照してください。

デザインビューでの設計時

デザインビューで Content プロパティを設定するには、次の手順に従います。

1. **C1AccordionItem** 項目をクリックして選択します。
2. [プロパティ]ウィンドウで、**Content** プロパティを文字列("Hello World" など)に設定します。
3. プログラムを実行し、アコーディオンペインを展開します。

XAML の場合

XAML で Content プロパティを設定するには、次の手順に従います。

1. <c1ext:C1AccordionItem> タグに **Content="Hello World"** を追加します。次のようになります。

```
XAML
<c1ext:C1AccordionItem Name="C1AccordionItem1" Content="Hello World" Width="150"
Height="55">
```

2. プログラムを実行し、アコーディオンペインを展開します。

コードの場合

コードで **Content** プロパティを設定するには、次の手順に従います。

1. コードビューに切り替えて、**InitializeComponent()** メソッドの下に次のコードを追加します。

Visual Basic

```
C1AccordionItem1.Content = "Hello World"
```

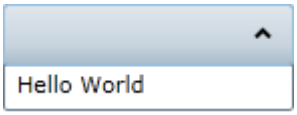
C#

```
c1AccordionItem1.Content = "Hello World";
```

2. プログラムを実行し、アコーディオンペインを展開します。

このトピックの作業結果

アコーディオンペインのコンテンツに「Hello World」が表示されます。このトピックの結果は、次のようになります。



コンテンツ領域へコントロールを追加する

各アコーディオンペイン(**C1AccordionItem**)は、コンテンツ領域に1つの子コントロールを受け入れます。このトピックでは、デザインビュー、XAML、およびコードで、1つの WPF/Silverlight ボタンコントロールを追加します。

このトピックは、少なくとも1つの C1AccordionItem 項目を含む **C1Accordion** コントロールがプロジェクトに追加されていることを前提としています。

デザインビューでの設計時

Button コントロールをコンテンツ領域に追加するには、次の手順に従います。

1. コントロールを追加するアコーディオンペインを選択します。
2. **[ボタン]**アイコンをダブルクリックして、アコーディオンペインのコンテンツ領域に追加します。
3. デザイナで、Button コントロールを選択します。[プロパティ]ウィンドウにボタンのプロパティが表示されます。
4. **Width** プロパティを**「Auto」**に設定します。
5. **Height** プロパティを**「Auto」**に設定します。
6. プログラムを実行し、アコーディオンペインを展開すると、ボタンコントロールが表示されます。

XAML の場合

次の手順に従います。

1. XAML で **Button** コントロールをコンテンツ領域に追加するには、次の手順に従います。
2. 次のマークアップを <c1ext:C1AccordionItem> タグと </c1ext:C1AccordionItem> タグの間に配置します。

XAML

```
<Button Content="ボタン" Height="Auto" Width="Auto"/>
```

3. プログラムを実行し、アコーディオンペインを展開すると、ボタンコントロールが表示されます。

コードの場合

コードで **Button** コントロールをコンテンツ領域に追加するには、次の手順に従います。

1. コードビューに切り替えて、**InitializeComponent()** メソッドの下に次のコードを追加します。

Visual Basic

```
'Button コントロールを作成します
Dim NewButton As New Button()
NewButton.Content = "ボタン"
'Button コントロールの Width および Height プロパティを設定します
NewButton.Width = Double.NaN
NewButton.Height = Double.NaN
'ボタンをコンテンツ領域に追加します
C1AccordionItem1.Content = (NewButton)
```

C#

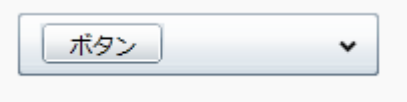
```
//Button コントロールを作成します
Button NewButton = new Button();
NewButton.Content = "ボタン";
//Button コントロールの Width および Height プロパティを設定します
NewButton.Width = double.NaN;
```

```
NewButton.Height = double.NaN;
//ボタンをコンテンツ領域に追加します
c1AccordionItem1.Content = (NewButton);
```

2. プログラムを実行し、アコーディオンペインを展開すると、ボタンコントロールが表示されます。

このトピックの作業結果

アコーディオンペインを展開すると、コンテンツ領域にボタンコントロールが表示されます。次の画像のようになります。



コンテンツ領域へ複数のコントロールを追加する

アコーディオンペイン(**C1AccordionItem**)の Content プロパティに一度に複数のコントロールを設定することはできません。ただし、複数のコントロールを受け入れることができるパネルベースのコントロール(**StackPanel** コントロールなど)をアコーディオンペインのコンテンツ領域に追加することで、この問題を回避できます。パネルベースのコントロールに複数のコントロールを追加すると、アコーディオンペインのコンテンツ領域に各コントロールが表示されます。

このトピックは、少なくとも1つの **C1AccordionItem** 項目を含む **C1Accordion** コントロールがプロジェクトに追加されていることを前提としています。

コンテンツ領域に複数のコントロールを追加するには、次の手順に従います。

1. 次の XAML マークアップを タグと タグの間に配置します。

XAML

```
<c1ext:C1AccordionItem.Content>
        <StackPanel>
            <TextBlock Text="1番目のテキストブロック"/>
            <TextBlock Text="2番目のテキストブロック"/>
            <TextBlock Text="3番目のテキストブロック"/>
        </StackPanel>
</c1ext:C1AccordionItem.Content>
```

2. プログラムを実行します。
3. アコーディオンペインを展開し、3つの **TextBlock** コントロールがコンテンツ領域に表示されていることを確認します。結果は次のようになります。

展開方向を変更する

デフォルトでは、**ExpandDirection** プロパティは **Down** に設定されるため、**C1Accordion** コントロールのアコーディオンペインは上から下に展開されます。デザインビュー、XAML、またはコードで **ExpandDirection** プロパティを **Up**、**Right**、または **Left** に設定することで、展開方向を簡単に変更できます。

このトピックは、少なくとも1つの **C1AccordionItem** を含む **C1Accordion** コントロールがプロジェクトに追加されていることを前提としています。

デザインビューでの設計時

デザインビューで **ExpandDirection** プロパティを設定するには、次の手順に従います。

1. **C1Accordion** コントロールをクリックして選択します。
2. **[プロパティ]**ウィンドウで、**ExpandDirection** ドロップダウン矢印をクリックし、リストからいずれかのオプションを選択します。この例では、**[Right]**を選択します。

XAML の場合

XAML で **ExpandDirection** プロパティを **Right** に設定するには、**ExpandDirection**="**Right**" を <c1ext:C1Accordion> タグに追加します。次のようになります。

XAML

```
<c1ext:C1Accordion Name="C1Accordion1" ExpandDirection=Right Width="150" Height="55">
```

コードの場合

コードで **ExpandDirection** プロパティを設定するには、次の手順に従います。

1. コードビューに切り替えて、InitializeComponent() メソッドの下に次のコードを追加します。

Visual Basic

```
C1Accordion1.ExpandDirection = C1.WPF.Extended.ExpandDirection.Right
```

C#

```
c1Accordion1.ExpandDirection = C1.WPF.Extended.ExpandDirection.Right;
```

2. プログラムを実行します。

このトピックの作業結果

このトピックの手順に従うことで、**ExpandDirection** プロパティの設定方法について理解できました。このトピックでは、**ExpandDirection** プロパティを **Right** に設定しました。C1Accordion コントロールは次の画像のようになります。

アコーディオンの高さに合わせる

アコーディオンペイン(**C1AccordionItem**)のデフォルトの動作では、コンテンツの高さ(上下に展開する場合)または幅(左右に展開する場合)に合わせてペインが展開されます。**C1Accordion** コントロールを特定の高さまたは幅に合わせる場合は、コントロールの高さまたは幅を設定してから、コントロールの **Fill** プロパティを True に設定する必要があります。

デザインビューでの設計時

デザインビューで **Fill** プロパティを設定するには、次の手順に従います。

1. 3つのアコーディオンペインを **C1Accordion** コントロールに追加します(「アコーディオンペインを追加する」を参照)。
2. **C1Accordion** コントロールをクリックして選択します。
3. [プロパティ]ウィンドウで、次の手順に従います。
    - [幅/高さに合わせる]チェックボックスをオンにします。
    - Height プロパティを「200」に設定します。

XAML の場合

XAML で **Fill** プロパティを **True** に設定するには、次の手順に従います。

1. <c1ext:C1Accordion> タグに **Fill="True"** を追加します。次のようになります。

```
XAML
<c1ext:C1Accordion Name="C1Accordion1" Fill="True">
```

2. <c1ext:C1Accordion> タグに Height="200" を追加します。次のようになります。

```
XAML
<c1ext:C1Accordion Name="C1Accordion1" Fill="True" Height="200">
```

コードの場合

コードで Fill プロパティを設定するには、次の手順に従います。

1. <c1ext:C1Accordion> タグに **Height="200"** を追加します。これにより、各アコーディオンペインは 200 ピクセルの高さに合わせて展開されます。
2. コードビューに切り替えて、**InitializeComponent()** メソッドの下に次のコードを追加します。

Visual Basic

```
C1Accordion1.Fill = True
```

C#

```
c1Accordion1.Fill = true;
```

3. プログラムを実行します。

レイアウトおよび外観

以下のトピックでは、**C1Accordion** 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。テーマを使用することで、グリッドの外観をカスタマイズしたり、WPF/Silverlight の XAML ベースのスタイル設定を活用することができます。また、テンプレートを使用して、コントロールを書式設定およびレイアウトしたり、コントロールの操作をカスタマイズすること

もできます。

ComponentOne ClearStyle 技術

ComponentOne 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 WPF/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。

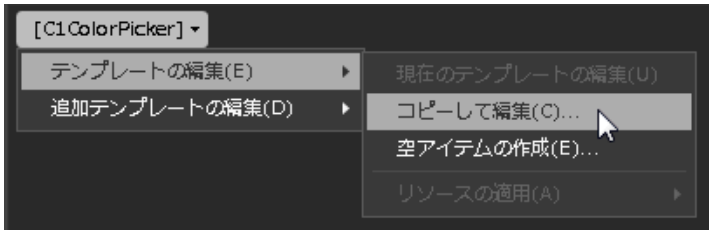
ClearStyle は、すばやく簡単にスタイルを変更することを意図したソリューションですが、ComponentOne のコントロールには引き続き従来の方法を使用して、必要なスタイルを細かく指定して作成できます。完全なカスタム設計が必要になる特別な状況で ClearStyle が邪魔になることはありません。

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。WPF/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、**Accordion for WPF/Sliverlight** 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、C1Accordion 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。

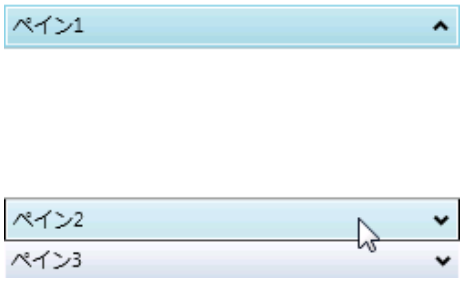
メモ:メニューを使用して新しいテンプレートを作成する場合、テンプレートはそのテンプレートのプロパティに自動的にリンクされます。手作業でテンプレートの `XAML` を作成する場合は、作成したテンプレートに適切な `Template` プロパティをリンクする必要があります。

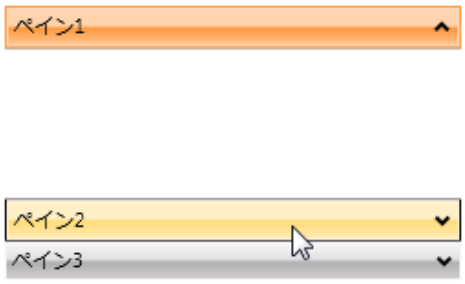
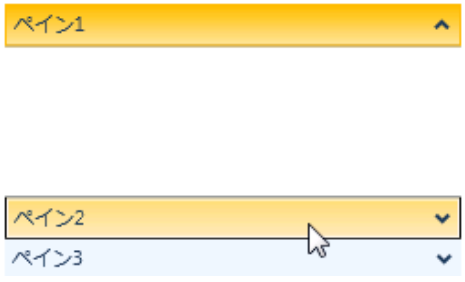
Template プロパティを使用して、テンプレートをカスタマイズできます。

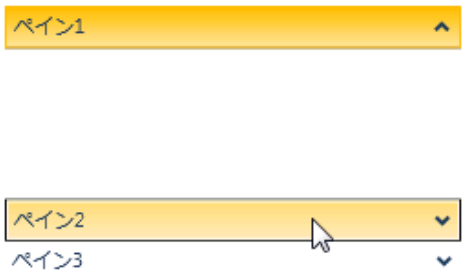
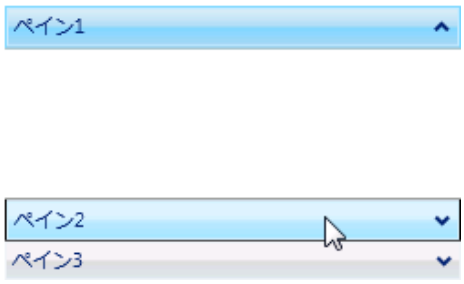
Accordion for WPF テーマ

Accordion for WPF には、グリッドの外観をカスタマイズできるいくつかのテーマが組み込まれています。**C1Accordion** コントロールを初めてページに追加すると、次の図のように表示されます。

これは、このコントロールのデフォルトの外観です。この外観は、組み込みテーマの1つを使用したり、独自のカスタムテーマを作成することで変更できます。すべての組み込みテーマは、WPF Toolkit テーマに基づいています。以下に、組み込みテーマの説明と図を示します。以下の図では、選択状態のスタイルを示すために1つの行が選択されています。

テーマ名	テーマのプレビュー
C1Blue	
C1ThemeBureauBlack	
C1ThemeExpressionDark	

C1ThemeExpressionLight	
C1ThemeOffice2007Blue	
C1ThemeOffice2007Black	
C1ThemeOffice2007Silver	
C1ThemeOffice2010Blue	
C1ThemeOffice2010Black	

C1ThemeOffice2010Silver	
C1ThemeShinyBlue	
C1ThemeWhistlerBlue	

要素のテーマを設定するには、**ApplyTheme** メソッドを使用します。最初に、テーマアセンブリへの参照をプロジェクトに追加し、次のようにコードでテーマを設定します。

VisualBasic

```
Private Sub Window_Loaded(sender As System.Object, e As System.Windows.RoutedEventArgs)
Handles MyBase.Loaded
   Dim theme As New C1ThemeExpressionDark
   ' ApplyTheme の使用
   C1Theme.ApplyTheme(LayoutRoot, theme)
```

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
   {
      C1ThemeExpressionDark theme = new C1ThemeExpressionDark();
      //ApplyTheme の使用
      C1Theme.ApplyTheme(LayoutRoot, theme);
   }
```

アプリケーション全体にテーマを適用するには、**System.Windows.ResourceDictionary.MergedDictionaries** プロパティを使用します。最初に、テーマアセンブリへの参照をプロジェクトに追加し、次のようにコードでテーマを設定します。

VisualBasic

```
 Private Sub Window_Loaded(sender As System.Object, e As System.Windows.RoutedEventArgs)
```

```
Handles MyBase.Loaded
   Dim theme As New C1ThemeExpressionDark
   ' MergedDictionaries の使用
Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThemeResources(theme))
End Sub
```

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
   {
      C1ThemeExpressionDark theme = new C1ThemeExpressionDark();
      //MergedDictionaries の使用

Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThemeResources(theme));
   }
```

この方法は、初めてテーマを適用する場合にのみ使用できることに注意してください。別の ComponentOne テーマに切り替える場合は、最初に、**Application.Current.Resources.MergedDictionaries** から前のテーマを削除します。

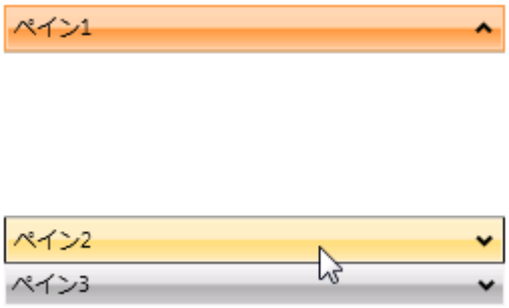
Accordion for Silverlight テーマ

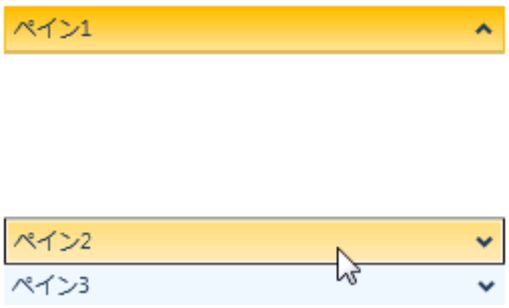
Silverlight テーマは、いくつかのコントロールの外観を定義するイメージ設定のコレクションです。テーマはアプリケーション内の複数のコントロールに適用できるため、テーマを使用すると、スタイル設定作業を繰り返さなくても、一貫性のあるコントロールを作成できます。

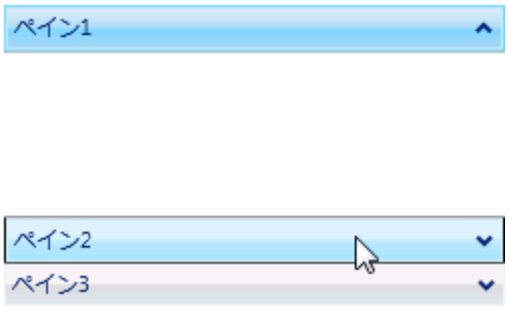
C1Accordion コントロールをプロジェクトに追加すると、コントロールはデフォルトの青色のテーマで表示されます。

C1Accordion コントロールには、Silverlight の 複数 のテーマ（BureauBlack、ExpressionDark、ExpressionLight、RainierOrange、ShinyBlue、WhistlerBlue）の中から1つテーマを設定できます。次の表に、テーマごとのサンプルを示します。

完全なテーマ名	外観
C1ThemeBureauBlack	

C1ThemeCosmopolitan	
C1ThemeExpressionDark	
C1ThemeExpressionLight	
C1ThemeOffice2007Black	
C1ThemeOffice2007Blue	

C1ThemeOffice2007Silver	
C1ThemeOffice2010Black	
C1ThemeOffice2010Blue	
C1ThemeOffice2010Silver	
C1ThemeRainierOrange	

C1ThemeShinyBlue	
C1ThemeWhistlerBlue	

C1Accordion コントロールにいずれかのテーマを追加するには、マークアップでこのコントロールに対してテーマを宣言してから、Theme.Apply モードを Auto に設定します。

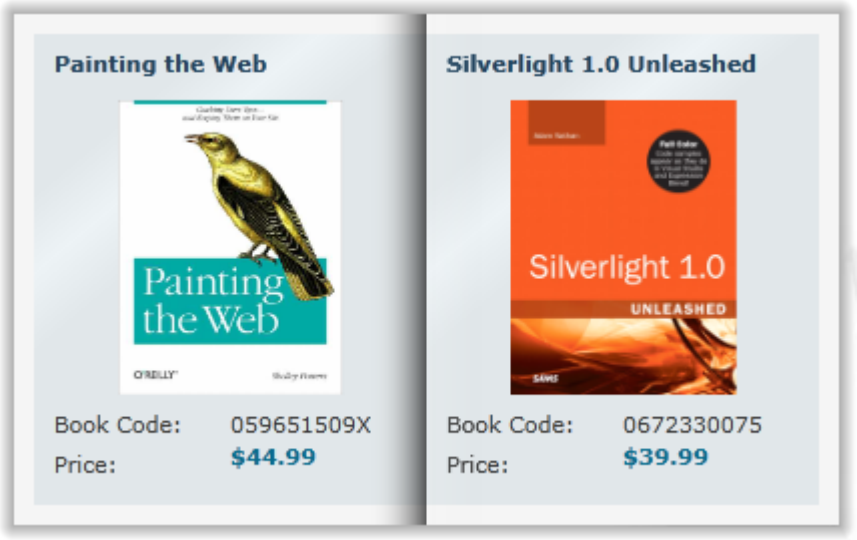
Book

C1Book は、本のようにページをめくることができる革新的な WPF/Silverlight のナビゲーションコントロールです。この見慣れた本の体裁を使用して、情報を表示します。**C1Book** コントロールを使用して、**UIElement** オブジェクトを普通の本のページのように表現できます。**Book for WPF/Silverlight** を使用して、一度に2つの要素を表示、影を追加、マウスでページをめくるなどの操作を行うことができます。

C1Book for WPF/Silverlight の概要

Book for WPF/Silverlight には C1Book コントロールが入っています。これは、コンテナとして動作する簡単なブックコントロールです。コントロールや画像などを見慣れた本の形式で追加できます。XAML ウィンドウに追加された C1Book コントロールは、パネルに似たコンテナになります。これをカスタマイズしたり、これにコンテンツを追加することができます。

コントロールのインターフェイスは、次の図のように表示されます。



Book for WPF クイックスタート

このクイックスタートは、**Book for WPF** 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、最初に 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 **Book for WPF** コントロールを追加して、コントロールの外観と動作をカスタマイズします。

さまざまなコンテンツを含む C1Book コントロールを使用する WPF アプリケーションを作成します。アプリケーションに C1Book コントロールを追加し、Book にコンテンツを追加してカスタマイズし、**Book for WPF** で実行可能ないくつかの操作を確認します。

手順 1:アプリケーションの作成

この手順では、**Book for WPF** を使用して WPF アプリケーションを作成します。C1Book コントロールをアプリケーションに追加すると、完全に機能する本のようなインターフェイスになり、そこに画像、コントロールなどの要素を追加することができます。プロジェクトをセットアップし、C1Book 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 プロジェクトを作成します。詳細について、「ComponentOne for WPF/Silverlight ユーザーガイド」を参照してください。
2. ソリューションエクスプローラで、References 項目を右クリックし、[参照の追加]を選択します。C1.WPF、C1.WPF.Extended、および WPFToolkit アセンブリを選択し、[OK]をクリックしてプロジェクトに参照を追加します。

3. Visual Studio のツールボックスに移動し、[C1Book]アイコンをダブルクリックして、ウィンドウにコントロールを追加します。
4. ウィンドウのサイズを変更し、ウィンドウに C1Book コントロールを再配置します。
5. デザインビューで C1Book コントロールをクリックし、[プロパティ]ウィンドウに移動して、次のプロパティを設定します。
   - [Width]を「450」に、[Height]を「300」に設定します。
   - HorizontalAlignment および VerticalAlignment を Center に設定して、コントロールをパネルの中央に配置します。
   - [IsFirstPageOnTheRight]チェックボックスをオンにして、最初のページが右側に表示されるように設定します。
   - TurnInterval プロパティを 600 に設定して、ページめくりアニメーションにかかる時間を長くします。

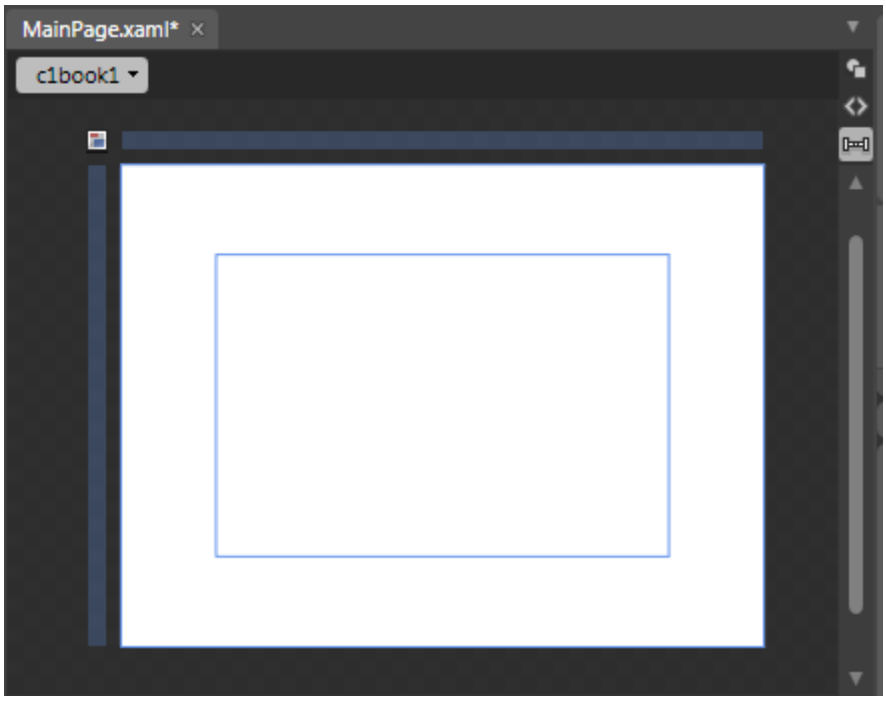
   XAML は次のようになります。

```
XAML
<c1:C1Book Name="C1Book1" Width="450" Height="300" VerticalAlignment="Center"

HorizontalAlignment="Center" IsFirstPageOnTheRight="True" TurnInterval="600" />
```

6. XAML ビューで C1Book コントロールのマークアップを更新して、次のように終了タグを入れます。

```
XAML
<c1:C1Book Name="C1Book1" Width="450" Height="300" VerticalAlignment="Center"

HorizontalAlignment="Center" IsFirstPageOnTheRight="True" TurnInterval="600">
</c1:C1Book>
```

   ページのデザインビューは次の図のようになります（フォームで C1Book コントロールを選択している場合）。



これで、アプリケーションのユーザーインターフェイスのセットアップは終了しましたが、C1Book コントロールにはまだコンテンツがありません。次の手順では、C1Book コントロールにコンテンツを追加し、さらにアプリケーションにコードを追加して、コン

トロールに機能を追加します。

手順 2:コントロールへのコンテンツの追加

この手順では、設計時、XAML マークアップ、およびコードで C1Book コントロールにコンテンツを追加します。標準の Microsoft コントロールおよびコンテンツを追加して、ページめくり可能な複数のページから成る仮想の本を作成します。プロジェクトをカスタマイズしてアプリケーションの C1Book コントロールにコンテンツを追加するには、次の手順に従います。

1. C1Book コントロールをクリックして選択します。
2. ツールボックスに移動し、TextBlock コントロールをダブルクリックしてプロジェクトに追加します。
3. XAML ビューで、TextBlock のタグを **C1Book** コントロールのタグ内に移動します。
4. デザインビューで TextBlock を選択し、[プロパティ]ウィンドウに移動して、次のプロパティを設定します。
   - **[Text]**を「Hello World!」に
   - **[HorizontalAlignment]**を **Center** に
   - **[VerticalAlignment]**を **Center** に
5. XAML ビューに切り替え、マークアップで TextBlock の直後に2つのボタンコントロールを追加します。マークアップは、次のようになります。

XAML

```
<c1:C1Book Name="C1Book1" Width="450" Height="300" VerticalAlignment="Center"
HorizontalAlignment="Center" IsFirstPageOnTheRight="True" TurnInterval="600">
  <TextBox Height="23" HorizontalAlignment="Center" Margin="10,0,0,102"
Name="TextBox1"
   VerticalAlignment="Center" Width="120">Hello World!</TextBox>
   <Button x:Name="Button1" Content="前へ" Height="100" Width="100"
Click="Button1_Click"/>
   <Button x:Name="Button2" Content="次へ" Width="150" Height="150"
Click="Button2_Click"/>
</c1:C1Book>
```

これは、コードでボタンにアクセスできるように名前を付け、コントロールをサイズ設定し、イベントハンドラを追加します。イベントハンドラには、次の手順でコードを追加します。

6. デザインビューで、ウィンドウをダブルクリックしてコードビューに切り替えます。
7. コードビューで、次の import 文をページの先頭に追加します。

Visual Basic

```
Imports C1.WPF
Imports C1.WPF.Extended
```

C#

```
using C1.WPF;
```

```
using C1.WPF.Extended;
```

8. 次のコードをページのコンストラクタの直後に追加することで、**Click** イベントにハンドラを追加します。

Visual Basic

```
Private Sub Button1_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)
     Me.C1Book1.CurrentPage = Me.c1book1.CurrentPage - 1
End Sub
Private Sub Button2_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)
    Me.C1Book1.CurrentPage = Me.c1book1.CurrentPage + 2
End Sub
```

C#

```
private void Button1_Click(object sender, System.Windows.RoutedEventArgs e)
{
    this.c1Book1.CurrentPage = this.c1Book1.CurrentPage - 1;
}
private void Button2_Click(object sender, System.Windows.RoutedEventArgs e)
{
    this.c1Book1.CurrentPage = this.c1Book1.CurrentPage + 1;
}
```

これで、ユーザーは、実行時にボタンを使用して最終ページや次ページに移動できるようになります。

9. コードを Window_Loaded イベントに追加します。次のようになります。

Visual Basic

```
Private Sub Window1_Loaded(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles MyBase.Loaded
  Dim txt1 As New TextBlock
  txt1.VerticalAlignment = VerticalAlignment.Center
   txt1.HorizontalAlignment = HorizontalAlignment.Center
    txt1.Text = "終わりです。"
    C1Book1.Items.Add(txt1)
End Sub
```

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    TextBlock txt1 = new TextBlock();
    txt1.VerticalAlignment = VerticalAlignment.Center;
    txt1.HorizontalAlignment = HorizontalAlignment.Center;
```

```
    txt1.Text = "終わりです、E;
    c1Book1.Items.Add(txt1);
}
```

これにより、コードで C1Book コントロールに **TextBlock** が追加されます。

10. プロジェクトを保存し、XAML ビューに戻ります。
11. XAML ビューで、<Button x:Name="Button2"/> タグの直後に次のマークアップを追加します。

XAML

```
<Grid x:Name="checkers" Background="White" ShowGridLines="True">
    <Grid.RowDefinitions>
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
    </Grid.ColumnDefinitions>
    <Rectangle Fill="Red" Grid.Row="0"  Grid.Column="0" Margin="5" />
    <Rectangle Fill="Black" Grid.Row="0" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="0" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="0" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="1" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="1" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="1" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="1" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="2" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="2" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="3" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="3" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="3" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="3" Grid.Column="3" Margin="5"  />
</Grid>
<Grid x:Name="checkers2" Background="White" ShowGridLines="True">
    <Grid.RowDefinitions>
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />

        <RowDefinition Height=".25*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
```

```
        <ColumnDefinition Width=".25*" />
    </Grid.ColumnDefinitions>
    <Rectangle Fill="Red" Grid.Row="0" Grid.Column="0" Margin="5" />
    <Rectangle Fill="Black" Grid.Row="0" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="0" Grid.Column="2" Margin="5"  />
   <Rectangle Fill="Black" Grid.Row="0" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="1" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="1" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="1" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="1" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="2" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="2" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="3" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="3" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="3" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="3" Grid.Column="3" Margin="5"  />
</Grid>
```

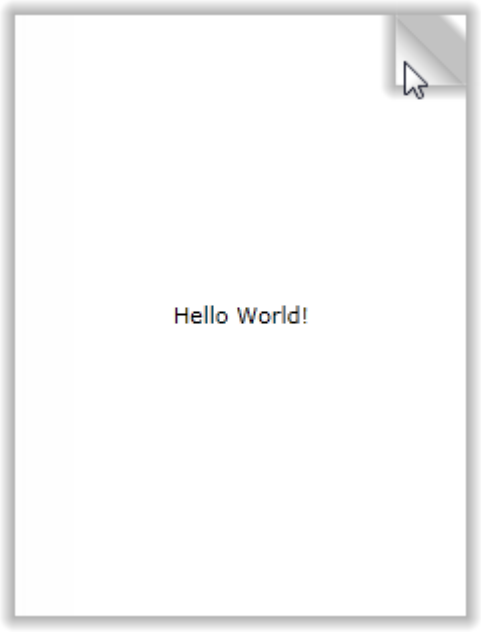
このマークアップにより、複数の **Rectangle** 要素から成る2つのグリッドが追加されます。このマークアップは、**C1Book** コントロールの1つのページに複数のコントロールを追加する方法を示しています。ただし、Grid、StackPanel などの1つのパネルにすべてのコントロールを置く必要があります。

この手順では、**C1Book** コントロールにコンテンツを追加しました。次の手順では、アプリケーションを実行し、実行時の操作を確認します。

手順 3：アプリケーションの実行

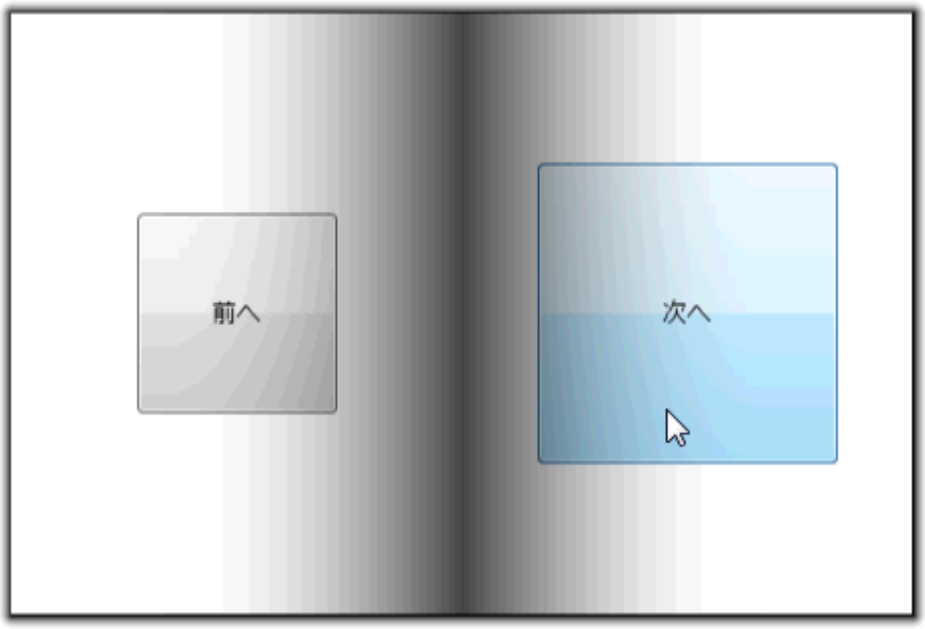
これまでに WPF アプリケーションを作成し、外観と動作をカスタマイズしたので、次にアプリケーションを実行します。アプリケーションを実行し、**Book for WPF** の実行時の動作を確認するには、次の手順に従います。

1. [デバッグ]メニューから[デバッグ開始]を選択し、実行時にアプリケーションがどのように表示されるかを確認します。アプリケーションは次の図のように表示されます。



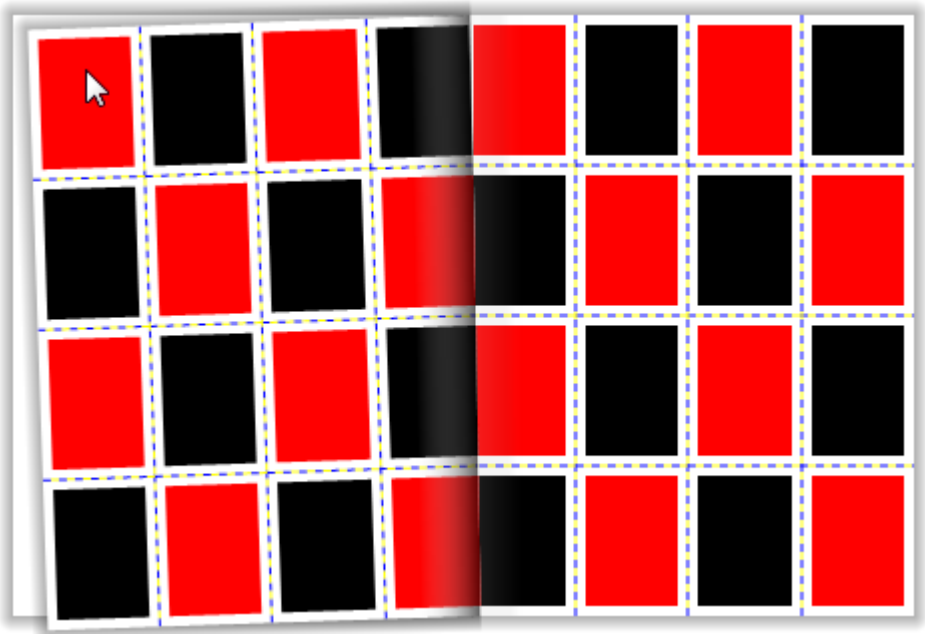
最初に1ページしか表示されないように **IsFirstPageOnTheRight** プロパティを設定しました。**C1Book** コントロールの右下または右上にカーソルを置くと、ページが少し折れて、ページをめくることができることがわかります。詳細については、「ブックのゾーン」を参照してください。

2. ページの右上をクリックするとページがめくれ、2ページ目と3ページ目が表示されます。



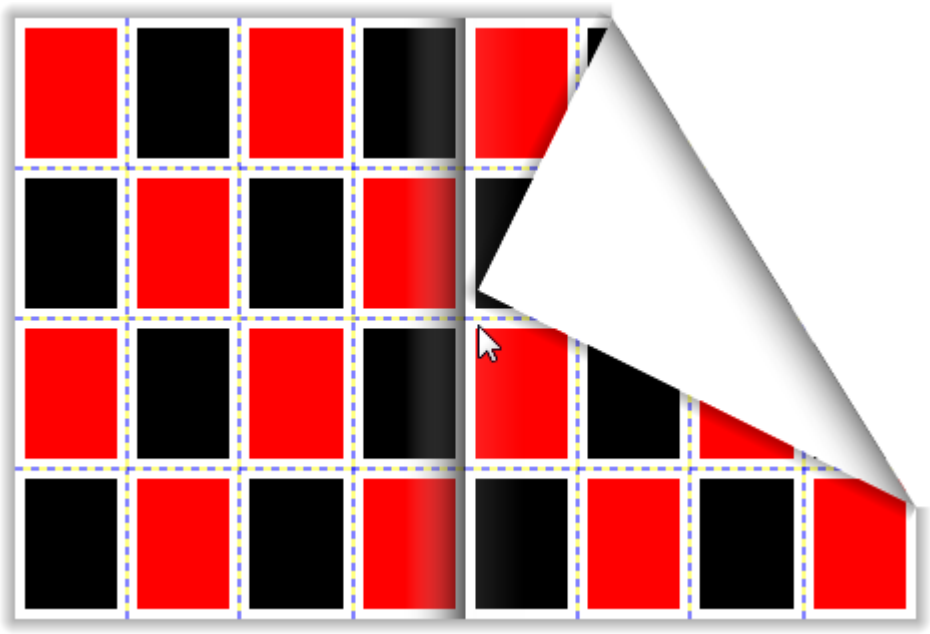
ページをめくる時間は、**TurnInterval** プロパティを設定してカスタマイズしました。

3. [次へ]ボタンをクリックします。次のページが表示されます。



左上または左下をクリックすると、前のページに戻ることができます。

4. ページの右上をクリックして左にドラッグし、次のページを開きます。



最後のページには、コードで追加した **TextBlock** コンテンツが入っています。

おめでとうございます!

これで **Book for WPF** クイックスタートは完了です。簡単な WPF アプリケーションを作成し、**Book for WPF** コントロールを1つ追加してカスタマイズしました。その後、コントロールの実行時機能をいくつか確認しました。

Book for Silverlight クイックスタート

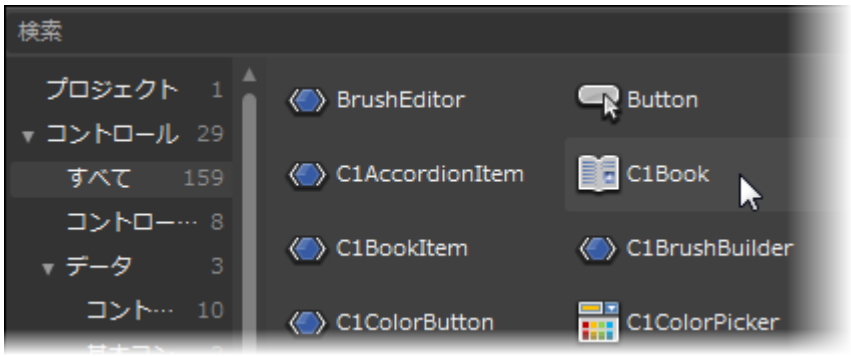
このクイックスタートは、**Book for Silverlight** を初めて使用するユーザーのために用意されています。このクイックスタートでは、新しいプロジェクトを作成し、アプリケーションに **C1Book** コントロールを追加して、コントロールの外観と動作をカスタマイズします。

この例では、Expression Blend を使って Silverlight アプリケーションを作成およびカスタマイズしていますが、下記の手順を Visual Studio で行うこともできます。**C1Book** コントロールを使用する簡単なプロジェクトを作成します。**C1Book** コントロールは、さまざまなコンテンツを含むブックの作成に使用します。

手順 1:アプリケーションの作成

この手順では、Expression Blend で **Book for Silverlight** を使って Silverlight アプリケーションを作成します。**C1Book** コントロールをアプリケーションに追加すると、完全に機能する本のようなインターフェイスになり、そこに画像、コントロールなどの要素を追加することができます。プロジェクトをセットアップし、**C1Book** 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. Expression Blend で、**[ファイル]**→**[新しいプロジェクト]**を選択します。
2. **[新しいプロジェクト]**ダイアログボックスで、左ペインからプロジェクトの種類として**[Silverlight]**を選択し、右ペインから**[Silverlight アプリケーション + Web サイト]**を選択します。プロジェクトの**[名前]**と**[場所]**を入力し、ドロップダウンボックスで **[言語]**を選択し、**[OK]**をクリックします。 新しいアプリケーションが作成され、**MainPage.xaml** ファイルがデザインビューで開きます。
3. **[プロジェクト]**ウィンドウに移動し、プロジェクトファイルリストで**[参照]** フォルダを右クリックします。コンテキストメニューから**[参照の追加]**を選択し、**C1.Silverlight.dll** および **C1.Silverlight.Extended.dll**アセンブリを見つけて選択し、**[開く]**をクリックします。ダイアログボックスが閉じ、プロジェクトに参照が追加されます。
4. ツールボックスで、**[アセット]**ボタン(二重山かっこアイコン)をクリックして、**[アセット]**ダイアログボックスを開きます。
5. **[アセット ライブラリ]**ダイアログボックスで、左ペインから**[コントロール]**項目を選択し、右ペインで**[C1Book]**アイコンをクリックします。



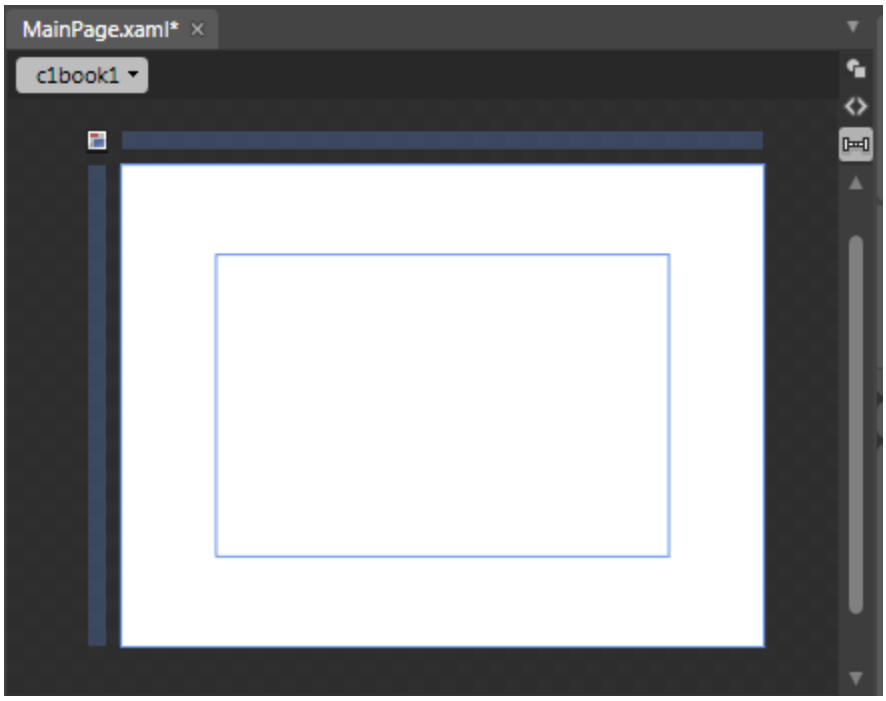
[C1Book]アイコンが、ツールボックスの**[アセット]**ボタンの下に表示されます。

6. **UserControl** のデザイン領域をクリックして選択します。Visual Studio とは異なり Blend では、次の手順に示すように、Silverlight コントロールを直接デザインサーフェスに追加できます。
7. ツールボックスの**[C1Book]**アイコンをダブルクリックして、コントロールをパネルに追加します。
8. デザインビューで **C1Book** コントロールをクリックし、[プロパティ]ウィンドウに移動して、次のプロパティを設定します。
   - コード内でコントロールにアクセスできるように、**[Name]**を「c1book1」に設定して名前を付けます。
   - **[Width]**を「450」に、**[Height]**を「300」に設定します。
   - **[HorizontalAlignment]**および**[VerticalAlignment]**を **Center** に設定して、コントロールをパネルの中央に配置します。
   - **[IsFirstPageOnTheRight]**チェックボックスをオンにして、最初のページが右側に表示されるように設定します。
   - **TurnInterval** プロパティを 600 に設定して、ページめくりアニメーションにかかる時間を長くします。

XAML は次のようになります。

```
<c1:C1Book x:Name="c1book1" Height="300" HorizontalAlignment="Center" VerticalAlignment="Center"
Width="450" IsFirstPageOnTheRight="True" TurnInterval="600"/>
```

ページのデザインビューは次の図のようになります（フォームで **C1Book** コントロールを選択している場合）。



これで、アプリケーションのユーザーインターフェイスのセットアップは終了しましたが、**C1Book** コントロールにはまだ何もコンテンツが入っていません。次の手順では、**C1Book** コントロールにコンテンツを追加し、さらにアプリケーションにコードを追加して、コントロールに機能を追加します。

手順 2：コンテンツの追加

この手順では、設計時、XAML マークアップ、およびコードで **C1Book** コントロールにコンテンツを追加します。標準の Microsoft コントロールおよびコンテンツを追加して、ページめくり可能な複数のページから成る仮想の本を作成します。プロジェクトをカスタマイズしてアプリケーションの **C1Book** コントロールにコンテンツを追加するには、次の手順に従います。

1. **C1Book** コントロールをクリックして選択します。
2. ツールボックスに移動し、**TextBlock** コントロールをダブルクリックしてプロジェクトに追加します。XAML ビューで、**C1Book** コントロールのマークアップ内に **TextBlock** コントロールが追加されたことがわかります。
3. **[オブジェクトとタイムライン]**ウィンドウで **TextBlock** を選択し、次のプロパティを設定します。
   - **[Text]**を「Hello World!」に設定します。
   - **[HorizontalAlignment]**を Center に設定します。
   - **[VerticalAlignment]**を Center に設定します。
4. **[オブジェクトとタイムライン]**ウィンドウで **C1Book** を選択し、ツールボックスに移動します。**[ボタン]**項目を2回ダブルクリックして、ページに2つのボタンを追加します。
5. XAML ビューで、これらの Button コントロールのマークアップを更新します。次のようになります。

XAML

```
<Button x:Name="btnLast" Content="前へ" Height="100" Width="100"
Click="btnLast_Click"/>
<Button x:Name="btnNext" Content="次へ" Width="150" Height="150"
Click="btnNext_Click"/>
```

これは、コントロールにコード内でアクセスできるように名前を付け、コントロールをサイズ変更し、イベントハンドラを追加します。イベントハンドラには、次の手順でコードを追加します。

6. [プロジェクト]ウィンドウで**[MainPage.xaml]**項目を展開し、コードファイル(**MainPage.xaml.cs** または **MainPage.xaml.vb**)をダブルクリックします。
7. コードビューで、次の Imports 文または using 文をページの先頭に追加します。

VisualBasic

```
Imports C1.Silverlight
Imports C1.Silverlight.Extended
```

C#

```
using C1.Silverlight;
using C1.Silverlight.Extended;
```

8. 次のコードをページのコンストラクタの直後に追加することで、**Click** イベントにハンドラを追加します。

VisualBasic

```
Private Sub btnLast_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)
    Me.c1book1.CurrentPage = Me.c1book1.CurrentPage - 2
End Sub
Private Sub btnNext_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)
    Me.c1book1.CurrentPage = Me.c1book1.CurrentPage + 2
End Sub
```

C#

```
private void btnLast_Click(object sender, System.Windows.RoutedEventArgs e)
{
    this.c1book1.CurrentPage = this.c1book1.CurrentPage - 2;
}
private void btnNext_Click(object sender, System.Windows.RoutedEventArgs e)
{
    this.c1book1.CurrentPage = this.c1book1.CurrentPage + 2;
}
```

これは、コントロールにコード内でアクセスできるように名前を付け、コントロールをサイズ変更し、イベントハンドラを追加します。イベントハンドラには、次の手順でコードを追加します。

9. [プロジェクト]ウィンドウで[MainPage.xaml]項目を展開し、コードファイル(MainPage.xaml.cs または MainPage.xaml.vb)をダブルクリックします。

Visual Basic

```
Public Sub New()
    InitializeComponent()
    Dim txt1 as New TextBlock
    txt1.VerticalAlignment = VerticalAlignment.Center
    txt1.HorizontalAlignment = HorizontalAlignment.Center
    txt1.Text = "終わりです。"
    c1book1.Items.Add(txt1)
End Sub
```

C#

```
public MainPage()
{
    InitializeComponent();
    TextBlock txt1 = new TextBlock();
    txt1.VerticalAlignment = VerticalAlignment.Center;
    txt1.HorizontalAlignment = HorizontalAlignment.Center;
    txt1.Text = "終わりです。";
    c1book1.Items.Add(txt1);
}
```

これで、ユーザーは、実行時にボタンを使って前のページや次ページに移動できるようになります。

1. 次のコードを Page コンストラクタに追加します。

Visual Basic

```
Public Sub New()
    InitializeComponent()
    Dim txt1 as New TextBlock
    txt1.VerticalAlignment = VerticalAlignment.Center
    txt1.HorizontalAlignment = HorizontalAlignment.Center
    txt1.Text = "終わりです。"
    c1book1.Items.Add(txt1)
End Sub
```

C#

```
public MainPage()
{
    InitializeComponent();
    TextBlock txt1 = new TextBlock();
    txt1.VerticalAlignment = VerticalAlignment.Center;
    txt1.HorizontalAlignment = HorizontalAlignment.Center;
    txt1.Text = "終わりです、E;
```

```
    c1book1.Items.Add(txt1);
}
```

2. プロジェクトを保存し、XAML ビューに戻ります。
3. XAML ビューで、<Button x:Name="btnNext"/> タグの直後に次のマークアップを追加します。

XAML

```
<Grid x:Name="checkers" Background="White" ShowGridLines="True">
    <Grid.RowDefinitions>
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
    </Grid.ColumnDefinitions>
    <Rectangle Fill="Red" Grid.Row="0"  Grid.Column="0" Margin="5" />
    <Rectangle Fill="Black" Grid.Row="0" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="0" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="0" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="1" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="1" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="1" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="1" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="2" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="2" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="3" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="3" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="3" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="3" Grid.Column="3" Margin="5"  />
</Grid>
<Grid x:Name="checkers2" Background="White" ShowGridLines="True">
    <Grid.RowDefinitions>
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
       <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
    </Grid.ColumnDefinitions>
    <Rectangle Fill="Red" Grid.Row="0" Grid.Column="0" Margin="5" />
    <Rectangle Fill="Black" Grid.Row="0" Grid.Column="1" Margin="5"  />
```

```
    <Rectangle Fill="Red" Grid.Row="0" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="0" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="1" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="1" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="1" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="1" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="2" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="2" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="3" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="3" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="3" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="3" Grid.Column="3" Margin="5"  />
</Grid>
```

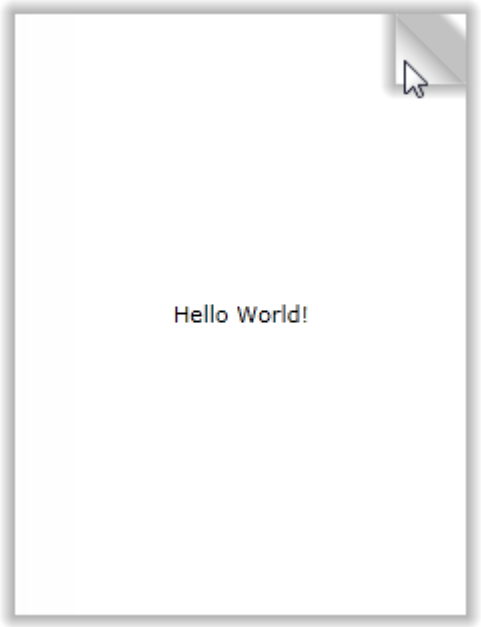
このマークアップにより、複数の Rectangle 要素から成る2つのグリッドが追加されます。このマークアップは、**C1Book** コントロールの1つのページに複数のコントロールを追加する方法を示しています。ただし、Grid、StackPanel などの1つのパネルにすべてのコントロールを置く必要があります。

この手順では、**C1Book** コントロールにコンテンツを追加しました。次の手順では、アプリケーションを実行し、実行時の操作を確認します。

手順 3：アプリケーションの実行

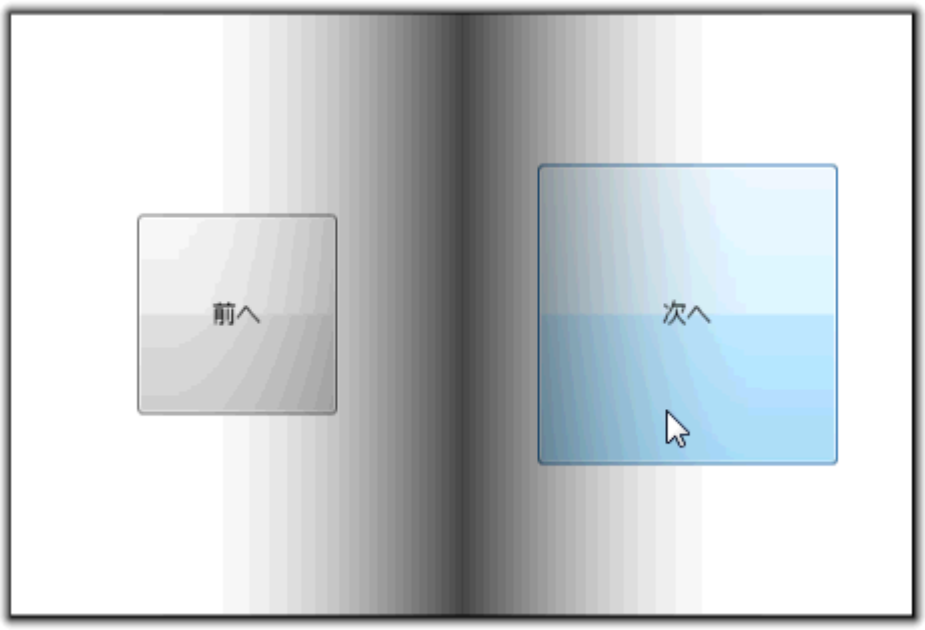
これまでに Silverlight アプリケーションを作成し、外観と動作をカスタマイズしたので、次にアプリケーションを実行します。アプリケーションを実行し、**Book for Silverlight** の実行時の動作を確認するには、次の手順に従います。

1. **[デバッグ]**メニューから**[デバッグ開始]**を選択して、実行時にアプリケーションがどのように表示されるかを確認します。アプリケーションは次の図のように表示されます。



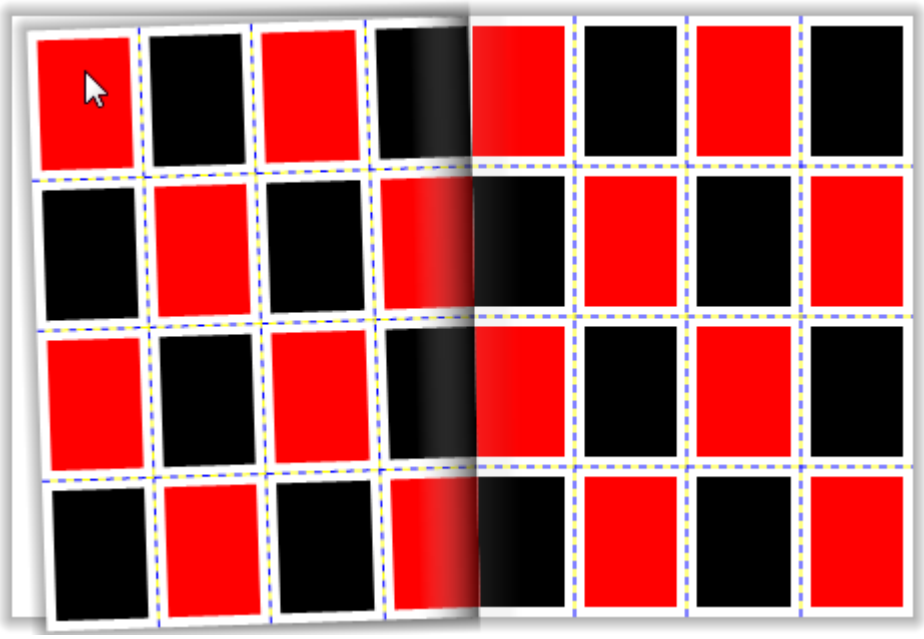
最初に1ページしか表示されないように **IsFirstPageOnTheRight** プロパティを設定しました。**C1Book** コントロールの右下または右上にカーソルを置くと、ページが少し折れて、ページをめくることができることがわかります。詳細については、「ブックのゾーン」を参照してください。

2. ページの右上をクリックするとページがめくれ、2ページ目と3ページ目が表示されます。



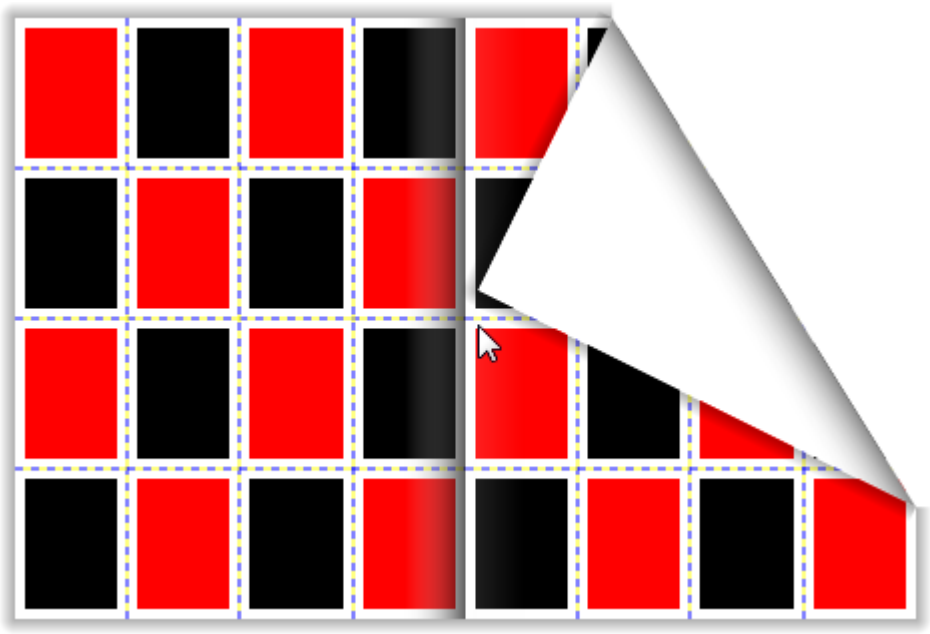
ページをめくる時間は、**TurnInterval** プロパティを設定してカスタマイズしました。

3. **[次へ]**ボタンをクリックします。次のページが表示されます。



左上または左下をクリックすると、前のページに戻ることができます。

4. ページの右上をクリックして左にドラッグし、次のページを開きます。



最後のページには、コードで追加した **TextBlock** コンテンツが入っています。

おめでとうございます!

これで **Book for Silverlight** クイックスタートは完了です。簡単な Silverlight アプリケーションを作成し、**Book for Silverlight** コントロールを1つ追加およびカスタマイズし、コントロールの実行時機能をいくつか確認しました。

主な特長

Book for WPF/Silverlight を使用して、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。以下の主要な機能をうまく利用して、**Book for WPF/Silverlight** を最大限に活用してください。

- **見慣れた本の体裁**
  C1Book は、本という見慣れたモデルを使用して、情報を革新的に表現することができます。しかも、Book for WPF/Silverlight は普通の静的な本ではなく、動的かつ対話式であり、単に見慣れた体裁という以上に WPF を使いこなすための機能を持ちます。

- **本物の本のような画像**
  C1Book では、ページの外観をカスタマイズできます。たとえば、ページの折り返しを表示したり、内側と外側に影を作ることができます。本のように見えるだけでなく、本のように扱うことができます。
- **柔軟なデータ連結**
  C1Book は **ItemsControl** なので、どのデータソースにも連結可能です。データソースの各項目は、**UIElement** にするか、テンプレートを使用して **UIElement** に変換される汎用オブジェクトにすることができます。
- **カスタムスタイルのページと表紙**
  C1Book は、奇数ページと偶数ページで異なるテンプレートをサポートしています。また、表紙などのカスタムページを定義することができます。

ブックのゾーン

C1Book コントロールにはいくつかのゾーンがあります。これらのゾーンを使用して、ユーザーがコントロールの各部を操作したときに行う動作をカスタマイズできます。CurrentZone プロパティを使用して、現在のゾーンを取得できます。**CurrentZoneChanged** イベントを使用すると、ユーザーが別のゾーンに移動したときに行う動作をカスタマイズできます。

C1Book コントロールには6つのゾーンがあります。各ゾーンの具体的な場所については、次の表内の画像でマウスの位置を確認してください。

ゾーン	説明	例
Out	ブックの境界線の外側のゾーンを指定します。	
BottomLeft	左下の折り返しゾーンを指定します。	

TopLeft	左上の折り返しゾーンを指定します。	
Center	ブックの中央を指定します（折り返しゾーンではない）。	
TopRight	右上の折り返しゾーンを指定します。	
BottomRight	左下の折り返しゾーンを指定します。	

ページの折り返しのサイズ

ブックのページをめくる際の外観をカスタマイズする方法の1つは、**FoldSize** プロパティを使用して、ページの折り返しのサイズを設定することです。ページの折り返しは、ユーザーが所定のブックのゾーンにマウスを置くと表示され、ページをめくることができることを示します。

FoldSize プロパティを設定すると、すべてのページ折り返し(右上、右下、左上、左下)のサイズが設定されます。したがって、たとえば **FoldSize** が **40** の場合、左下と右下の折り返しは次の図のように表示されます。

大きい数字を設定するほど、折り返しも大きくなります。**FoldSize** が **80** の場合、左下と右下の折り返しは次の図のように表示されます。

ページの折り返しの表示/非表示

デフォルトでは、ユーザーが所定のブックのゾーンにマウスを置くと、ページの折り返しが表示されます。ただし、必要に応じて、ページの折り返しの表示/非表示を変更できます。**ShowPageFold** プロパティを次の表のいずれかの値に設定して、C1Book コントロールの操作方法を決定できます。

値	説明
OnMouseOver	折り返しは、ユーザーがマウスでページの端をドラッグすると表示されます。これはデフォルト設定です。
Never	折り返しは表示されません。
Always	折り返しは常に表示されます。

ページめくりオプション

デフォルトでは、ユーザーがページの折り返しを1回クリックすると、ブックに前のページまたは次のページが表示されます。**PageFoldAction** プロパティを使用すると、ページをクリックしてめくる方法をカスタマイズできます。たとえば、ユーザーが

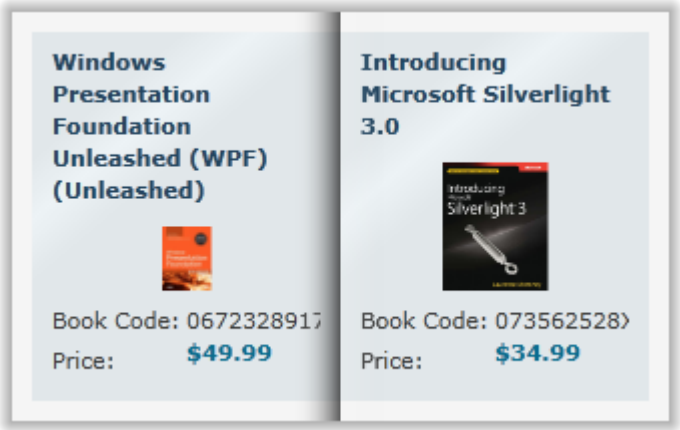
ダブルクリックするとページがめくられるように **PageFoldAction** を設定できます。または、マウスのクリックではページをめくらず、ページの折り返し上でドラッグアンドドロップ操作をしないとページめくりが行われないようにすることもできます。

PageFoldAction プロパティを次の表のいずれかの値に設定して、C1Book コントロールの操作方法を決定できます。

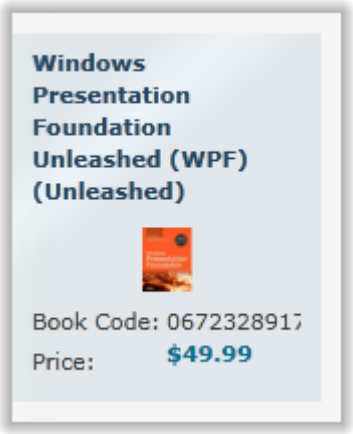
値	説明
TurnPageOnClick	ユーザーがページの折り返しをクリックするとページがめくられます。
TurnPageOnDoubleClick	ユーザーがページの折り返しをダブルクリックするとページがめくられます。
None	ユーザーがページの折り返しをブック上で左右にドラッグするとページがめくられます。

最初のページの表示

デフォルトでは、C1Book コントロールの最初のページは左側に表示されます。これは、開いた本のような見え方になります。



ただし、必要に応じて、**IsFirstPageOnTheRight** プロパティを **True** に設定すると、最初のページが右側に表示されるように変更できます。最初のページが右側に表示されるように設定すると、閉じた本の表紙のように見えます。



ページの影

C1Book コントロールには、立体感が出るように影が付いています。コントロールには内側の影と外側の影があります。内側の影は、ブックの中央部と、めくった右ページの背後に表示されます。外側の影は、コントロールの外側と、めくった左ページの背後に表示されます。たとえば、次の画像に内側の影と外側の影を示します。

影を表示しない場合は、**ShowInnerShadows** と **ShowOuterShadows** プロパティを **False** に設定することで、表示/非表示を変更できます。

ブックナビゲーション

実行時は、マウスを使用して **C1Book** コントロール内を移動できます。ブックのゾーン内でクリックしたり、ドラッグアンドドロップ操作を行ってページをめくります。**C1Book** コントロールには、ナビゲーション関連のメソッド、プロパティ、およびイベントが含まれており、現在どのページが表示されているかを簡単に判断したり、ユーザーがブック内を移動する際のアプリケーションのアクションを簡単に設定することができます。

CurrentPage プロパティは、実行時に現在表示中のページを取得または設定します。ページをめくると、見開きで左側に表示されるページが **CurrentPage** になることに注意してください。ページ番号は **0** から始まり、ページ 0 は常に左側に表示されます。したがって、**IsFirstPageOnTheRight** プロパティを **True** に設定すると、ブックの最初のページは最初に右側に表示されてページ1になり、左側には非表示のページ 0 があります。

表示されるページは **CurrentPage** プロパティを使用して設定できますが、**TurnPage** メソッドを使用して、実行時に現在のページを変更することもできます。**TurnPage** メソッドは、ブックを1ページずつ進めたり戻したりします。

CurrentPageChanged イベントを使用すると、現在のページが変更されたときに発生するアクションを指定できます。また、**DragPageStarted** イベントと **DragPageFinished** イベントを使用すると、ユーザーがドラッグアンドドロップ操作によってページをめくったときに発生するアクションを指定できます。

レイアウトおよび外観

以下のトピックでは、**C1Book** 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。テーマを使用することで、グリッドの外観をカスタマイズしたり、WPF/Silverlight の XAML ベースのスタイル設定を活用することができます。また、テンプレートを使用して、コントロールを書式設定およびレイアウトしたり、コントロールの操作をカスタマイズすることもできます。

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。WPF 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、**Book 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、C1Book 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。

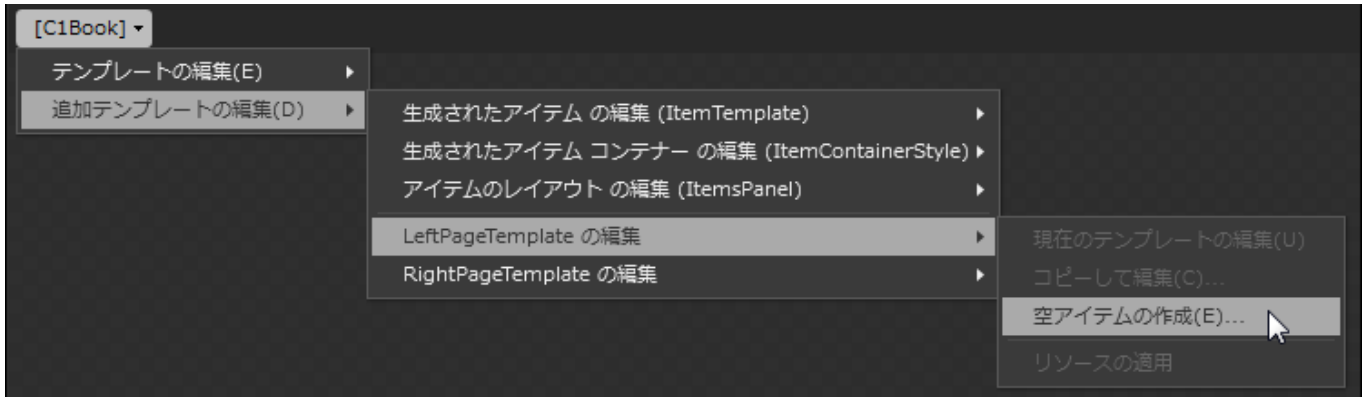
> **メモ**:メニューを使用して新しいテンプレートを作成する場合、テンプレートはそのテンプレートのプロパティに自動的にリンクされます。手作業でテンプレートの `XAML` を作成する場合は、作成したテンプレートに適切な `Template` プロパティをリンクする必要があります。

ページテンプレート

Book for WPF/Silverlight のブックコントロールには、LeftPageTemplate および RightPageTemplate の2つのページテンプレートが含まれます。これらのテンプレートは、それぞれブックの左ページと右ページの外観とレイアウトを制御します。これらのテンプレートはマスターページとして機能します。つまり、これらのページテンプレートに追加した項目は、そのテンプレートが適用されたすべてのページに表示されます。これは、たとえば、透かし模様や会社のロゴをすべてのページに追加する場合に便利です。ブック内の各ページに1つずつ追加していく必要はありません。

ページテンプレートへのアクセス

ページテンプレートにアクセスするには、Microsoft Expression Blend で、**C1Book** 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。**[LeftPageTemplate の編集]** または **[RightPageTemplate の編集]**を選択し、**[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。



[ControlTemplate リソースの作成]ダイアログボックスが表示されるので、ここでテンプレートに名前を付け、テンプレートを定義する場所を決定します。デフォルトでは、空の Grid コントロールが入った空のテンプレートが表示されます。

XAML

```
<ControlTemplate x:Key="NewLeftPageTemplate">
<Grid/>
</ControlTemplate>
```

テンプレートは、他の **ControlTemplate** と同様にカスタマイズできます。

スタイル

Book for WPF/Silverlight の **C1Book** コントロールは、コントロールの外観を変更するために使用できるスタイルのプロパティを提供します。これらのスタイルの一部について、次の表で説明します。

スタイル	説明
FontStyle	フォントスタイルを取得または設定します。これは依存プロパティです。
Style	この要素のレンダリング時に使用されるスタイルを取得または設定します。これは依存プロパティです。

テンプレートパーツ

Microsoft Expression Blend では、新しいテンプレートを作成して、テンプレートパーツを表示および編集できます。たとえば、C1Book コントロールをクリックして選択し、**[オブジェクト]**→**[テンプレートの編集]**→**[コピーして編集]**を選択します。新しいテンプレートを作成すると、テンプレートのパーツが**[パーツ]**ウィンドウに表示されます。

[パーツ]ウィンドウにパーツを表示するには、**ControlTemplate** を選択する必要があります。

[パーツ]ウィンドウで、任意の要素をダブルクリックすると、そのパーツをテンプレートに作成できます。これで、そのパーツがテンプレートに表示されます。また、**[パーツ]**ペイン内の要素アイコンが選択状態に変更されます。

C1Book コントロールでは、次のテンプレートパーツを使用きます。

名前	タイプ	説明
Root	FrameworkElement	WPF/Silverlight レイアウトに含まれるオブジェクトに共通 API フレームワークを提供します。また、WPF/Silverlight のデータ連結、オブジェクトツリー、およびオブジェクトライフタイムの機能に関連する API を定義します。

表示状態

Microsoft Expression Blend で、カスタム状態や状態グループを追加して、ユーザーコントロールの状態ごとに異なる外観を定義できます。たとえば、マウスが置かれたときのコントロールの表示状態を変更できます。 新しいテンプレートを作成し、「テンプレートパーツ」を追加することで、表示状態を表示および編集できます。これで、そのパーツで利用可能な表示状態が**[表示状態]**ウィンドウに表示されます。

よく使用される状態としては、項目の通常の外観を示す **Normal**、マウスが置かれている項目を示す **MouseOver**、有効でない項目を示す **Disabled** などがあります。フォーカスの状態には、項目にフォーカスがないときの **Unfocused**、項目にフォーカスがあるときの **Focused** などがあります。

タスク別ヘルプ

次のタスク別ヘルプトピックは、ユーザーの皆様が Visual Studio および Expression Blend に精通しており、**C1Book** コントロールの一般的な使用方法を理解していることを前提としています。**Book 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、**Book for WPF/Silverlight** 製品を使用して特定のタスクを行うための方法を提供します。タスク別ヘルプの多くのトピックは、新しい WPF/Silverlight プロジェクトが作成されており、プロジェクトに **C1Book** コントロールが追加されていることを前提としています。コントロールの作成の詳細については、「ブックを作成する」を参照してください。

ブックを作成する

C1Book コントロールは、設計時に、XAML およびコードで簡単に作成できます。次の手順で作成した **C1Book** コントロールは、空のコンテナとして表示されます。このコントロール

設計時

C1Book コントロールを作成するには、次の手順に従います。

1. ウィンドウのデザインサーフェスをクリックして選択します。
2. ツールボックスの **[C1Book]**アイコンをダブルクリックして、コントロールをパネルに追加します。これで、**C1Book** コントロールがアプリケーションに追加されました。
3. 必要に応じて、コントロールを選択し、[プロパティ]ウィンドウでプロパティを設定することにより、コントロールをカスタマイズすることもできます。

XAML の場合

C1Book コントロールを XAML マークアップを使用して作成するには、次の手順に従います。

1. Visual Studio ソリューションエクスプローラーで、プロジェクトファイルリスト内の[参照]フォルダを右クリックします。コンテキストメニューから[参照の追加]を選択し、**C1.WPF.4.dll/C1.Silverlight.5.dll** および **C1.WPF.Extended.4.dll/C1.Silverlight.Extended.5.dll** アセンブリを選択して、**[OK]**をクリックします。
2. xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml" を初期状態の タグに追加することで、プロジェクトに XAML 名前空間を追加します。次のようになります。

```
XAML
<Window xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:c1=http://schemas.componentone.com/winfx/2006/xaml
        x:Class="Window1" Title="Window1" Height="230" Width="348">
```

3. <c1:C1Book> タグをプロジェクトの <Grid> タグ内に追加して、C1Book コントロールを作成します。マークアップは次のようになります。

```
XAML
<Grid>
    <c1:C1Book x:Name="C1Book1" Height="300" Width="450"/>
    </c1:C1Book>
</Grid>
```

このマークアップは、「C1Book1」という名前の空の C1Book コントロールを作成し、コントロールのサイズを設定します。

コードの場合

C1Book コントロールをコードで作成するには、次の手順に従います。

1. Visual Studio ソリューションエクスプローラーで、プロジェクトファイルリスト内の[参照]フォルダを右クリックします。コンテキストメニューから[参照の追加]を選択し、**C1.WPF.4.dll/C1.Silverlight.5.dll** および **C1.WPF.Extended.4.dll/C1.Silverlight.Extended.5.dll** アセンブリを選択して、[OK]をクリックします。
2. XAML ビューで、タグを次のように更新し、ウィンドウ内の初期状態のグリッドに名前を付けます。

```
<Grid x:Name="LayoutRoot">
```

3. [Window1.xaml]ウィンドウ内で右クリックし、[コードの表示]を選択してコードビューに切り替えます。
4. 次の import 文をページの先頭に追加します。

Visual Basic

```
Imports C1.WPF
Imports C1.WPF.Extended
```

C#

```
using C1.WPF;
using C1.WPF.Extended;
```

5. ページのコンストラクタに、C1Book コントロールを作成するコードを追加します。次のようになります。

Visual Basic

```
Public Sub New()
    InitializeComponent()
    Dim c1book1 as New C1Book
    c1book1.Height = 300
    c1book1.Width = 450
     LayoutRoot.Children.Add(c1book1)
End Sub
```

C#

```
public MainPage()
{
    InitializeComponent();
    C1Book c1book1 = new C1Book();
    c1book1.Height = 300;
    c1book1.Width = 450;
    LayoutRoot.Children.Add(c1book1);
}
```

このコードは、「c1book1」という名前の空の C1Book コントロールを作成し、コントロールのサイズを設定し、コントロールをページに追加します。

ここまでの成果

C1Book コントロールを作成しました。上記の手順で作成した **C1Book** コントロールは、空のコンテナとして表示されます。このコントロールを実行時にブックとして表示するには、項目を追加する必要があります。例については、「ブックへ項目を追加する」を参照してください。

ブックへ項目を追加する

C1Book コントロールには、任意の種類のコンテンツを追加できます。これには、テキスト、画像、レイアウトパネルなどの標準コントロールやサードパーティのコントロールがあります。この例では、**C1Book** コントロールに **TextBlock** コントロールを追加しますが、手順をカスタマイズして他の種類のコンテンツを追加することもできます。

設計時

TextBlock コントロールをブックに追加するには、次の手順に従います。

1. C1Book コントロールをクリックして選択します。
2. ツールボックスに移動し、**TextBlock** 項目をダブルクリックして、コントロールを **C1Book** コントロールに追加します。
3. 必要に応じて、**C1Book** コントロールと TextBlock コントロールをカスタマイズできます。それには、各コントロールを選択し、[プロパティ]ウィンドウでプロパティを設定します。たとえば、TextBlock の Text プロパティを「Hello World!」に設定します。

XAML の場合

たとえば、**TextBlock** コントロールをブックに追加するには、<TextBlock Text="Hello World!"/> を <c1:C1Book> タグ内に追加します。次のようになります。

XAML

```
<c1:C1Book x:Name="C1Book1" Height="300" Width="450">
    <TextBlock Text="Hello World!"/>
</c1:C1Book>
```

コードの場合

たとえば、**TextBlock** コントロールをブックに追加するには、ページのコンストラクタに次のようにコードを追加します。

Visual Basic

```
Public Sub New()
    InitializeComponent()
    Dim txt1 as New TextBlock
    txt1.Text = "Hello World!"
    C1Book1.Items.Add(txt1)
End Sub
```

C#

```
public MainPage()
{
    InitializeComponent();
    TextBlock txt1 = new TextBlock();
    txt1.Text = "Hello World!";
    c1Book1.Items.Add(txt1);
}
```

ここまでの成果

C1Book コントロールにコントロールを追加しました。アプリケーションを実行し、**TextBlock** コントロールが **C1Book** コントロールに追加されていることを確認してください。他のコンテンツやコントロールも同様に追加できます。

ブック内の項目をクリアする

たとえば、実行時にユーザーがすべての項目を **C1Book** コントロールからクリアできるようにすることができます。また、コントロールを連結してから別のデータソースに再連結する際に、項目コレクションをクリアすることもできます。

たとえば、ブックのコンテンツをクリアするには、プロジェクトに次のコードを追加します。

Visual Basic

```
Me.C1Book1.Items.Clear()
```

C#

```
this.c1Book1.Items.Clear();
```

ここまでの成果

コントロールのコンテンツがクリアされます。アプリケーションを実行すると、ブックが空になっています。

最初のページを右側に表示する

IsFirstPageOnTheRight プロパティは、ブックの最初のページが右側または左側のどちらに表示されるかを取得または設定します。デフォルトでは、**C1Book** コントロールは、起動時に最初のページを左側にして2ページを見開きで表示します。これは、設計時に、XAML およびコードで **IsFirstPageOnTheRight** プロパティを設定してカスタマイズできます。

設計時

設計時に **IsFirstPageOnTheRight** プロパティを設定するには、次の手順に従います。

1. C1Book コントロールをクリックして選択します。
2. [プロパティ]ウィンドウに移動し、IsFirstPageOnTheRight 項目の横にあるチェックボックスをオンにします。

XAML の場合

たとえば、**IsFirstPageOnTheRight** プロパティを設定するには、**IsFirstPageOnTheRight**="**True**" を <c1:C1Book> タグに追加します。次のようになります。

XAML

```
<c1:C1Book x:Name="C1Book1" Height="300" Width="450" IsFirstPageOnTheRight="True">
```

コードの場合

たとえば、**IsFirstPageOnTheRight** プロパティを設定するには、ページのコンストラクタで次のコードをプロジェクトに追加します。

Visual Basic

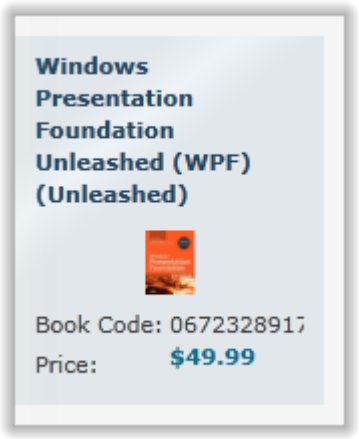
```
Me.C1Book1.IsFirstPageOnTheRight = True
```

C#

```
this.c1Book1.IsFirstPageOnTheRight = true;
```

ここまでの成果

最初のページが右側に表示されるように設定しました。アプリケーションを実行すると、最初のページが本の表紙のように1ページだけ表示されます。



初期ページを設定する

CurrentPage プロパティは、**C1Book** コントロールの現在のページの値を取得または設定します。デフォルトでは、**C1Book** コントロールは、起動時に最初のページを表示します。これは、Expression Blend での設計時に、XAML、およびコードで **CurrentPage** プロパティを設定してカスタマイズできます。

Blend での設計時

Blend での設計時に CurrentPage プロパティを **3** に設定するには、次の手順に従います。

1. **C1Book** コントロールをクリックして選択します。
2. [プロパティ]ウィンドウに移動し、**CurrentPage** 項目の横にあるテキストボックス内をクリックします。
3. 「3」など、表示する初期ページの番号を入力します。

XAML の場合

たとえば、CurrentPage プロパティを **3** に設定するには、CurrentPage="3" を タグに追加します。次のようになります。

XAML

```
<c1:C1Book x:Name="c1book1" Height="300" Width="450" CurrentPage="3">
```

コードの場合

たとえば、**CurrentPage** プロパティを **3** に設定するには、プロジェクトに次のコードを追加します。

Visual Basic

```
Me.c1book1.CurrentPage = 3
```

C#

```
this.c1book1.CurrentPage = 3;
```

ここまでの成果

ブックの初期開始ページを変更しました。アプリケーションを実行すると、最初にページ3が表示されます。

コードによりブック内のナビゲーションを行う

表示されるページは CurrentPage プロパティを使用して設定できますが、**C1Book.TurnPage** メソッドを使用して、実行時に現在のページを変更することもできます。詳細については、「ブックナビゲーション」を参照してください。このトピックでは、アプリケーションに2つのボタンを追加します。一方はページをめくって前のページに戻り、もう一方はページをめくって次のページに進みます。

ブックにナビゲーションを追加するには、次の手順に従います。

ここまでの成果

ブック内のナビゲーションをカスタマイズしました。ブック内のナビゲーションを確認するには、アプリケーションを実行し、右ボタンをクリックします。ページめくりアニメーションによって次のページがめくられることがわかります。

1. ツールボックスに移動し、[ボタン]項目をダブルクリックして、アプリケーションに2つの **Button** コントロールを追加します。
2. [Button1]を選択し、[プロパティ]ウィンドウに移動したら、**Content** プロパティを「<」に設定します。
3. [Button2]を選択し、[プロパティ]ウィンドウに移動したら、**Content** プロパティを「>」に設定します。
4. ウィンドウ内でボタンのサイズと位置を変更します。[Button1]ボタンをブックの左側に、[Button2]ボタンをブックの右側に置きます。
5. [Button1]をダブルクリックして **Button_Click** イベントハンドラを作成し、コードビューに切り替えます。
6. デザインビューに戻り、[Button2]で同じ手順を繰り返します。つまり、各ボタンに Click イベントを指定します。
   XAML マークアップは次のようになります。

   XAML
   ```
   <Button HorizontalAlignment="Right" Margin="0,43,12,0" Name="Button1" Width="28"
   Height="23" VerticalAlignment="Top">&gt;</Button>
   <Button Height="23" HorizontalAlignment="Left" Margin="12,43,0,0" Name="Button2"
   VerticalAlignment="Top" Width="28">&lt;</Button>
   ```

7. コードビューに切り替え、次の import 文をページの先頭に追加します。

   ## Visual Basic

   ```
   Imports C1.WPFImports C1.WPF.Extended
   ```

   ## C#

   ```
   using C1.WPF;using C1.WPF.Extended;
   ```

8. Click イベントハンドラにコードを追加します。次のようになります。

   ## Visual Basic

   ```
   Private Sub Button1_Click(ByVal sender as Object, ByVal e as
   System.Windows.RoutedEventArgs)
   ```

```
    Me.C1Book1.TurnPage(True)
End Sub
Private Sub Button2_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)
    Me.C1Book1.TurnPage(False)
End Sub
```

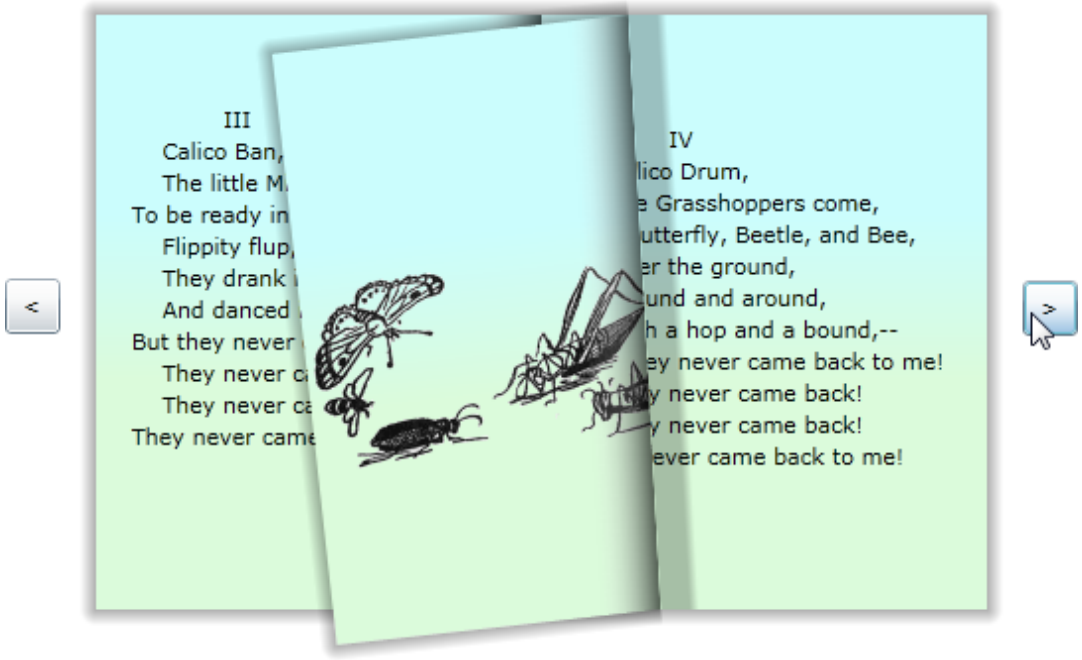
C#

```
public MainPage()
{
  private void button1_Click(object sender, System.Windows.RoutedEventArgs e)
   {
     this.c1Book1.TurnPage(true);
   }
  private void button2_Click(object sender, System.Windows.RoutedEventArgs e)
   {
     this.c1Book1.TurnPage(false);
   }
}
```

このコードは、ボタンのクリックによってブックを1ページ前または後にめくります。

ここまでの成果

ブック内のナビゲーションをカスタマイズしました。ブック内のナビゲーションを確認するには、アプリケーションを実行し、右ボタンをクリックします。ページめくりアニメーションによって次のページがめくられることがわかります。



左ボタンをクリックし、ブックが前のページに戻ることを確認します。

ColorPicker

ColorPicker for WPF/Silverlight は、機能豊富な対話式の色選択インターフェイスを提供する色入力エディタです。ユーザーは、高度な設計のパレットまたは独自に指定したカスタム色から色を選択できます。

事前選択済の色と標準の色を使った基本カラーパレット、ユーザーが色の選択を完全にカスタマイズできる高度なパレット、またはその両方を含めることができます。**ColorPicker for WPF/Silverlight** は、透過性、16 進数色表記、RGB と HLS のカラーモデルのサポートを含む豊富なビジュアル色入力インターフェイスも備えています。

ColorPicker for WPF クイックスタート

このクイックスタートは、**ColorPicker for WPF** 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 **ColorPicker for WPF** コントロールを追加して、コントロールの外観と動作をカスタマイズします。

2つの C1ColorPicker コントロールと標準の **Rectangle** コントロールを使用する簡単なプロジェクトを作成します。2つの **C1ColorPicker** コントロールは、**Rectangle** に適用されるグラデーションを制御し、実行時に選択した色によってグラデーションの色が変更されます。これにより、**ColorPicker for WPF** を使用したさまざまな可能性が広がります。

手順 1:アプリケーションの設定

この手順では、最初に Visual Studio で **ColorPicker for WPF** を使用する **WPF** アプリケーションを作成します。**C1ColorPicker** コントロールをアプリケーションに追加すると、完全な機能を備えた色入力セレクタとして使用できます。

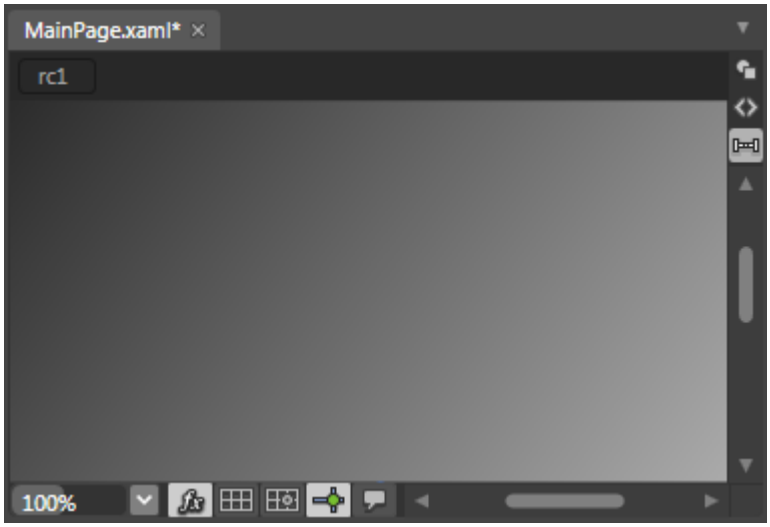
プロジェクトをセットアップし、**C1ColorPicker** コントロールをアプリケーションに追加するには、次の手順に従います。

1. Visual Studio で新しい WPF プロジェクトを作成します。
2. ツールボックスに移動し、**[Rectangle]**アイコンをダブルクリックして、Grid に標準コントロールを追加します。
3. ウィンドウのサイズを変更し、四角形をサイズ変更してウィンドウ全体に拡大します。
4. XAML ビューに切り替えて、Fill を <Rectangle> タグに追加します。次のようになります。

XAML

```
<Rectangle Name="Rectangle1" Stroke="Black">
    <Rectangle.Fill>
        <LinearGradientBrush x:Name="colors">
            <GradientStop x:Name="col1" Color="Black" Offset="0" />
            <GradientStop x:Name="col2" Color="White" Offset="1" />
        </LinearGradientBrush>
    </Rectangle.Fill>
</Rectangle>
```

これで、白黒の直線グラデーションが四角形に追加されます。ページのデザインビューは、次の図のようになります。



これで、**WPF** アプリケーションとカスタマイズした **Rectangle** コントロールが作成されました。次の手順では、**C1ColorPicker** のコントロールを追加してカスタマイズします。

手順 2:コントロールの追加

前の手順では、新しいプロジェクトを作成し、アプリケーションにグラデーション付きの **Rectangle** コントロールを1つ追加しました。この手順では、引き続き、**Rectangle** のグラデーション塗りつぶしを制御する **C1ColorPicker** のコントロールを追加します。

次の手順に従います。

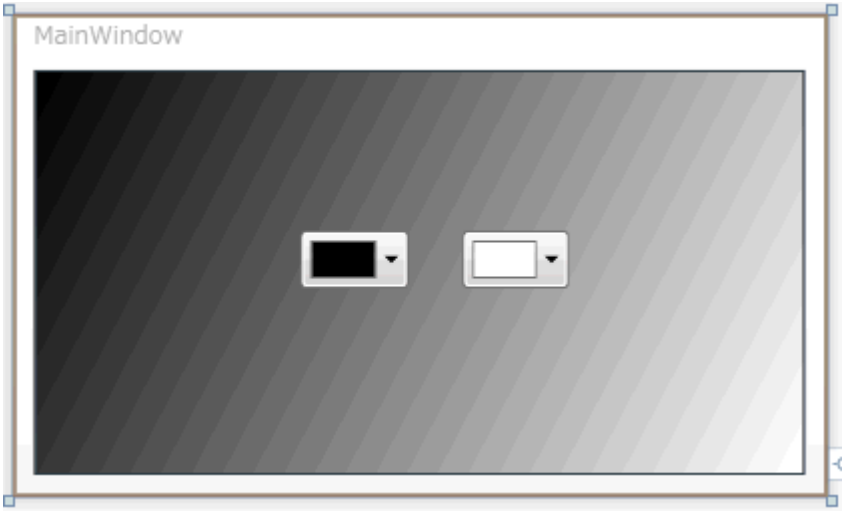
1. デザインビューで、Rectangle をクリックして選択し、Visual Studio のツールボックスに移動します。
2. ツールボックスで、**[C1ColorPicker]**アイコンを見つけて2回ダブルクリックし、フォームに2つのコントロールを追加します。
3. 2つの **C1ColorPicker** コントロールが四角形の中央に表示されるように、サイズを変更して配置します。
4. デザインビューで、最初の **C1ColorPicker** コントロール(**C1ColorPicker1**)をクリックし、[プロパティ]ウィンドウに移動して、次のプロパティを設定します。
    - **DropDownDirection** を **AboveOrBelow** に設定して、コントロールの開き方を制御します。
    - **Mode** を **Advanced** に設定して、詳細カラーピッカーのみを表示します。
    - **SelectedColor** を **Black**(または "#FF000000")に設定します。

      XAML は次のようになります。

      XAML

      ```
      <c1ext:C1ColorPicker Margin="125,70,185,107.5" Name="C1ColorPicker1"
      xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
      DropDownDirection="AboveOrBelow" Mode="Advanced" SelectedBrush="Black" />
      ```

      2番目の C1ColorPicker コントロールはデフォルト値のままにします。ページのデザインビューは次の図のようになります。



これで、アプリケーションのユーザーインターフェイスが正しくセットアップされましたが、ここでアプリケーションを起動して色を選択しても、カラーピッカーは何も実行しません。次の手順では、コントロールに機能を追加するコードをアプリケーションに追加します。

手順 3：アプリケーションへのコードの追加

これまでの手順では、アプリケーションのユーザーインターフェイスを設定し、いくつかのコントロールをアプリケーションに追加しました。この手順では、アプリケーションにコードを追加して完成させます。

次の手順に従います。

1. 最初の **C1ColorPicker** コントロール（C1ColorPicker1）をクリックして選択します。
2. [プロパティ]ウィンドウで、稲妻の[イベント]アイコンをクリックしてコントロールのイベントを表示します。
3. **SelectedColorChanged** イベントの横にあるテキストボックスをダブルクリックし、コードビューに切り替えてイベントハンドラを作成します。
4. コードビューで、次の import 文をページの先頭に追加します。

Visual Basic

```
Imports C1.WPF
Imports C1.WPF.Extended
```

C#

```
using C1.WPF;
using C1.WPF.Extended;
```

5. Page コンストラクタの直後に次のコードを追加して、グラデーションの値を更新します。

Visual Basic

```
Private Sub UpdateGradient()
    If C1ColorPicker1 IsNot Nothing And C1ColorPicker2 IsNot Nothing Then
        Me.col1.Color = Me.C1ColorPicker1.SelectedColor
        Me.col2.Color = Me.C1ColorPicker2.SelectedColor
    End If
End Sub
```

C#

```
void UpdateGradient()
{
    if (c1ColorPicker1 != null & c1ColorPicker2 != null)
    {
        this.col1.Color = this.c1ColorPicker1.SelectedColor;
        this.col2.Color = this.c1ColorPicker2.SelectedColor;
    }
}
```

6. C1ColorPicker1_SelectedColorChanged イベントハンドラにコードを追加します。次のようになります。

Visual Basic

```
Private Sub C1ColorPicker1_SelectedColorChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles
C1ColorPicker1.SelectedColorChanged
    UpdateGradient()
End Sub
```

C#

```
private void c1ColorPicker2_SelectedColorChanged(object sender,
C1.WPF.PropertyChangedEventArgs e)
{
    UpdateGradient();
}
```

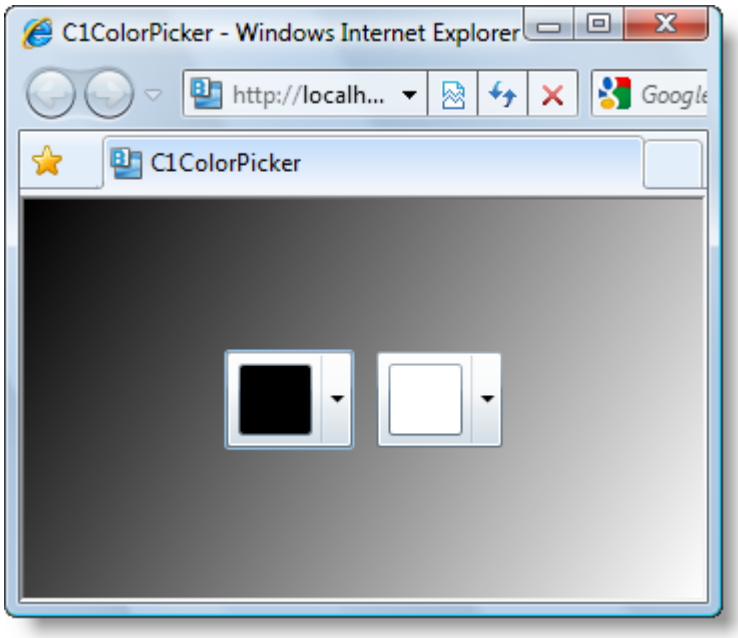
7. デザインビューに戻ります。
8. 2番目の **C1ColorPicker** コントロール(**C1ColorPicker2**)をクリックして選択します。
9. [プロパティ]ウィンドウで、**SelectedColorChanged** イベントの横にあるテキストボックスをダブルクリックしてコードビューに切り替え、イベントハンドラを作成します(イベントが表示されていない場合は、稲妻の[イベント]アイコンをクリックしてコントロールのイベントを表示する必要があります)。

この手順では、アプリケーションにコードを追加しました。次の手順では、アプリケーションを実行し、実行時の操作を確認します。

手順 4:アプリケーションの実行

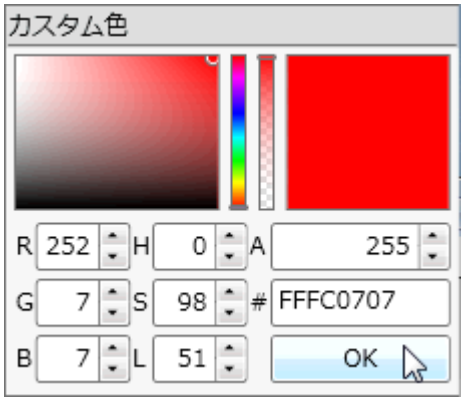
これまでに WPF アプリケーションを作成し、外観と動作をカスタマイズしたので、次にアプリケーションを実行します。アプリケーションを実行し、**ColorPicker for WPF** の実行時の動作を確認するには、次の手順に従います。

1. [デバッグ]メニューから[デバッグ開始]を選択し、実行時にアプリケーションがどのように表示されるかを確認します。アプリケーションは次の図のように表示されます。



2. 左のカラーピッカーのドロップダウン矢印をクリックします。ドロップダウンボックスの上にウィンドウが開き、拡張モードのみが表示されることを確認してください。コントロールに対して行った変更が反映されていることがわかります。拡張モードでは、ユーザーは任意の色を指定でき、複数の方法を使用して色を選択できます。

3. [赤]などの色を選択し、[OK]をクリックします。

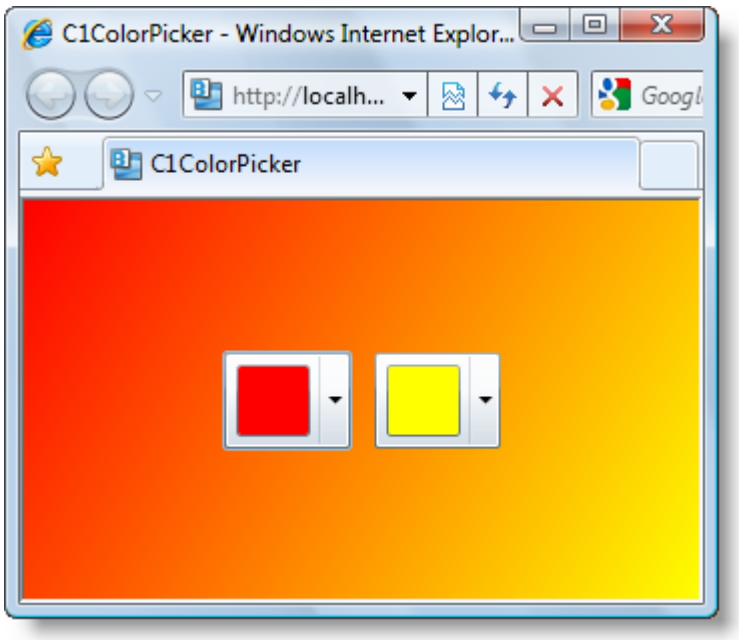


コントロールに対して選択した色と四角形に対して選択したグラデーションが選択内容を反映して変更されていることを確認してください。

4. 右のカラーピッカーのドロップダウン矢印をクリックします。

[基本]タブが表示されていることを確認します（デフォルト設定）。このタブには、**[パレットの色]**、**[標準の色]**、および**[最近使用した色]**が表示されます。任意の色を選択できますが、**[拡張]**タブに切り替えてカスタム色を選択することもできます。選択された色は、赤色の境界線で強調表示されます。

5. [黄色]などの色を選択します。選択された色が変化し、四角形の背景グラデーションも選択内容に合わせて変化します。



おめでとうございます!

これで **ColorPicker for WPF** クイックスタートは完了です。簡単な WPF アプリケーションを作成し、**ColorPicker for WPF** コントロールを追加およびカスタマイズし、コントロールの実行時機能をいくつか確認しました。

ColorPicker for Silverlight クイックスタート

このクイックスタートは、**ColorPicker for 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、Expression Blend で作業を開始し、新しいプロジェクトを作成し、アプリケーションに **C1ColorPicker** コントロールを追加し、コントロールの外観と動作をカスタマイズし、コントロールで実行できるいくつかの実行時の操作を確認します。

この例では、Expression Blend を使って Silverlight アプリケーションを作成およびカスタマイズしていますが、下記の手順を Visual Studio で行うこともできます。2つの **C1ColorPicker** コントロールと標準の **Rectangle** コントロールを使用する簡単なプロジェクトを作成します。2つの **C1ColorPicker** コントロールは、**Rectangle** に適用されるグラデーションを制御し、実行時

に選択した色によってグラデーションの色が変更されます。これにより、**ColorPicker for Silverlight** を使用したさまざまな可能性が広がります。

手順 1:アプリケーションの作成

この手順では、Expression Blend で ColorPicker for Silverlight を使って Silverlight アプリケーションを作成します。**C1ColorPicker** コントロールをアプリケーションに追加すると、完全な機能を備えた色入力セレクタとして使用できます。プロジェクトをセットアップし、**C1ColorPicker** 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。

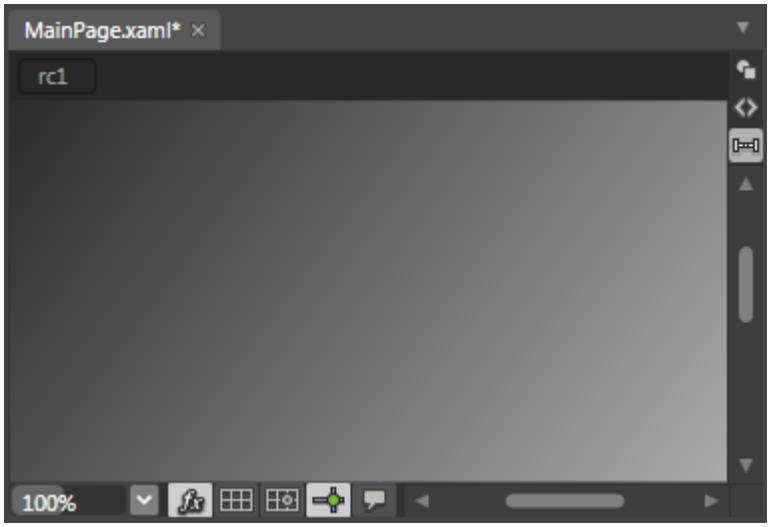
1. Expression Blend で、**[ファイル]**→**[新しいプロジェクト]**を選択します。
2. **[新しいプロジェクト]**ダイアログボックスで、左ペインからプロジェクトの種類として[Silverlight]を選択し、右ペインから **[Silverlight アプリケーション + Web サイト]**を選択します。プロジェクトの**[名前]**と**[場所]**を入力し、ドロップダウンボックスで**[言語]**を選択し、**[OK]**をクリックします。 新しいアプリケーションが作成され、MainPage.xaml ファイルがデザインビューで開きます。
3. [プロジェクト]ウィンドウに移動し、プロジェクトファイルリストで**[参照]** フォルダを右クリックします。**[参照の追加]** ダイアログボックスで **[参照の追加]**を選択し、**C1.Silverlight.dll** アセンブリと **C1.Silverlight.Extended.dll** アセンブリを見つけて選択し、**[開く]**をクリックします。ダイアログボックスが閉じ、プロジェクトに参照が追加されます。
4. **UserControl** のデザイン領域をクリックして選択します。Visual Studio とは異なり Blend では、次の手順に示すように、Silverlight コントロールを直接デザインサーフェスに追加できます。
5. ツールボックスに移動し、**[Rectangle]**アイコンをダブルクリックして、**Grid** に標準コントロールを追加します。
6. <Rectangle> タグに x:Name="rc1" Width="Auto" Height="Auto" を追加します。次のようになります。
   <Rectangle x:Name="rc1" Width="Auto" Height="Auto"></Rectangle>

   これで、四角形が **Grid** 全体に拡大されます。
7. XAML ビューに切り替えて、Fill を タグに追加します。次のようになります。

   XAML

   ```
   <Rectangle x:Name="rc1" Height="Auto" Width="Auto">
       <Rectangle.Fill>
           <LinearGradientBrush x:Name="colors">
               <GradientStop x:Name="col1" Color="Black" Offset="0" />
               <GradientStop x:Name="col2" Color="White" Offset="1" />
           </LinearGradientBrush>
       </Rectangle.Fill>
   </Rectangle>
   ```

   これで、白黒の直線グラデーションが四角形に追加されます。ページのデザインビューは、次の図のようになります。



これで、Silverlight アプリケーションとカスタマイズした **Rectangle** コントロールが作成されました。次の手順では、**C1ColorPicker** のコントロールを追加してカスタマイズします。

手順 2:コントロールの追加

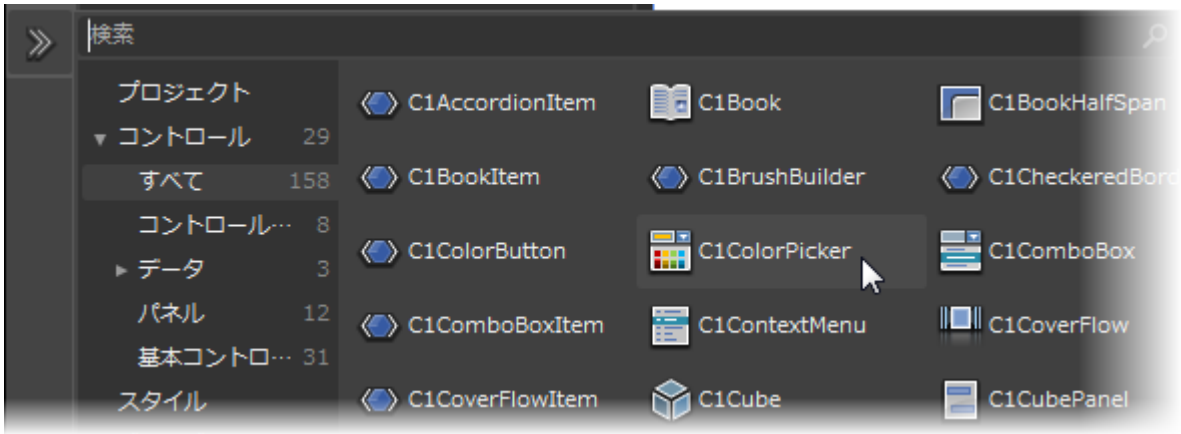
前の手順では、新しい Silverlight プロジェクトを作成し、アプリケーションにグラデーション付きの **Rectangle** コントロールを1つ追加しました。この手順では、引き続き、**Rectangle** のグラデーション塗りつぶしを制御する **C1ColorPicker** のコントロールを追加します。

次の手順に従います。

1. XAML ビューで、タグのすぐ下に次のマークアップを追加して、ページに **StackPanel** を追加します。

```
XAML
<StackPanel Height="Auto" Width="Auto" HorizontalAlignment="Center"
VerticalAlignment="Center" Orientation="Horizontal"></StackPanel>
```

2. ツールボックスで、**[アセット]**ボタン(二重山かっこアイコン)をクリックして、**[アセット]**ダイアログボックスを開きます。
3. **[アセット ライブラリ]**ダイアログボックスで、左ペインから**[コントロール]**項目を選択し、右ペインで**[C1ColorPicker]**アイコンをクリックします。



[C1ColorPicker]アイコンが、ツールボックスの**[アセット]**ボタンの下に表示されます。

4. <StackPanel></StackPanel> タグペアの内側をクリックし、**C1ColorPicker** アイコンをダブルクリックしてコントロールをパネルに追加します。

5. デザインビューで **C1ColorPicker** コントロールをクリックし、[プロパティ]ウィンドウに移動して、次のプロパティを設定します。
   - コード内でコントロールにアクセスできるように、**[Name]**を「c1cp1」に設定して名前を付けます。
   - **[Width]**を「65」に、**[Height]**を「50」に設定します。
   - **[HorizontalAlignment]**および**[VerticalAlignment]**を **Center** に設定して、コントロールをパネルの中央に配置します。
   - **Left**、**Right**、**Top**、**Bottom**、**Margin** の各プロパティを「5」に設定します。
   - **DropDownDirection** を **AboveOrBelow** に設定して、コントロールの開き方を制御します。
   - **Mode** を **Advanced** に設定して、詳細カラーピッカーのみを表示します。
   - **SelectedColor** を **Black**（または "#FF000000"）に設定します。

   XAML は次のようになります。

```
XAML
<c1:C1ColorPicker x:Name="c1cp1" Width="65" Height="50"
HorizontalAlignment="Center"
VerticalAlignment="Center" Margin="5"
DropDownDirection="AboveOrBelow"Mode="Advanced"
SelectedColor="Black"/>
```

6. 次の XAML を <c1:C1ColorPicker x:Name="c1cp1"/> タグの直後に追加して、2つめの **C1ColorPicker** コントロールを StackPanel に追加します。

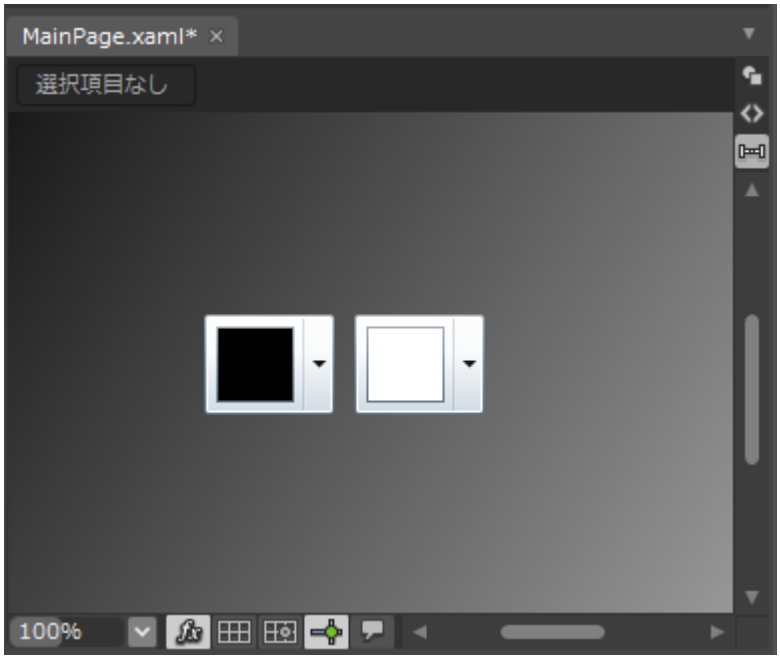
```
XAML
<c1:C1ColorPicker x:Name="c1cp2" Width="65" Height="50"
HorizontalAlignment="Center"
VerticalAlignment="Center" Margin="5" DropDownDirection="BelowOrAbove"
Mode="Both"
SelectedColor="White"/>7re>
```

   デフォルト値に設定されていた **Mode**、**DropDownDirection** などのプロパティが異なる値に設定されていることに注意してください。

7. XAML ビューで、<UserControl> タグの **Width="640" Height="480"** を **Width="Auto" Height="Auto"** に置き換えます。次のようになります。

```
XAML
<UserControl xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
xmlns:c1=http://schemas.componentone.com/winfx/2006/xaml
x:Class="C1ColorPicker.MainPage" Width="Auto" Height="Auto">
```

   ページのデザインビューは次の図のようになります。



これで、アプリケーションのユーザーインターフェイスが正しくセットアップされましたが、ここでアプリケーションを起動して色を選択しても、カラーピッカーは何も実行しません。次の手順では、コントロールに機能を追加するコードをアプリケーションに追加します。

手順 3:コードの追加

これまでの手順では、アプリケーションのユーザーインターフェイスを設定し、いくつかのコントロールをアプリケーションに追加しました。この手順では、アプリケーションにコードを追加して完成させます。

次の手順に従います。

1. 左側の **C1ColorPicker** コントロール(**c1cp1**)をクリックして選択します。
2. [プロパティ]ウィンドウで、稲妻の**[イベント]**アイコンをクリックしてコントロールのイベントを表示します。
3. **SelectedColorChanged** イベントの横にあるテキストボックスをダブルクリックし、コードビューに切り替えてイベントハンドラを作成します。
4. コードビューで、次の Imports 文または using 文をページの先頭に追加します。

Visual Basic

```
Imports C1.Silverlight
Imports C1.Silverlight.Extended
```

C#

```
Imports C1.Silverlight
Imports C1.Silverlight.Extended
```

5. C1ColorPicker1_SelectedColorChanged イベントハンドラにコードを追加します。次のようになります。

Visual Basic

```
Private Sub UpdateGradient()
    If c1cp1 IsNot Nothing And c1cp2 IsNot Nothing Then
        Me.col1.Color = Me.c1cp1.SelectedColor
        Me.col2.Color = Me.c1cp1.SelectedColor
    End If
End Sub
```

C#

```
void UpdateGradient()
{
 if (c1cp1 != null & c1cp2 != null)
{
this.col1.Color = this.c1cp1.SelectedColor;
      this.col2.Color = this.c1cp2.SelectedColor;
   }
}
```

6. Page コンストラクタの直後に次のコードを追加して、グラデーションの値を更新します。
7. **c1cp1_SelectedColorChanged** イベントハンドラにコードを追加します。次のようになります。

Visual Basic

```
Private Sub c1cp1_SelectedColorChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles
c1cp1.SelectedColorChanged
    UpdateGradient()
End Sub
```

C#

```
private void c1cp1_SelectedColorChanged(object sender,
C1.Silverlight.PropertyChangedEventArgs<System.Windows.Media.Color> e)
{
    UpdateGradient();
}
```

8. デザインビューに戻ります。
9. 右側の **C1ColorPicker** コントロール（**c1cp2**）をクリックして選択します。
10. [プロパティ]ウィンドウで、**SelectedColorChanged** イベントの横にあるテキストボックスをダブルクリックしてコードビューに切り替え、イベントハンドラを作成します（イベントが表示されていない場合は、稲妻の[イベント]アイコンをクリックしてコントロールのイベントを表示する必要があります）。
11. **c1cp2_SelectedColorChanged** イベントハンドラにコードを追加します。次のようになります。

Visual Basic

```
Private Sub c1cp2_SelectedColorChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles c1cp1.SelectedColorChanged
    UpdateGradient()
End Sub
```

C#

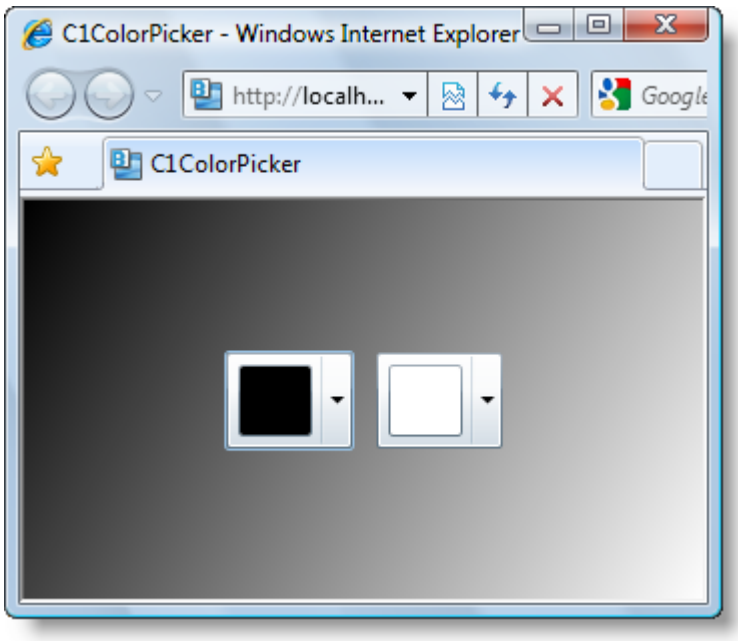
```
private void c1cp2_SelectedColorChanged(object sender,
C1.Silverlight.PropertyChangedEventArgs
<System.Windows.Media.Color> e)
{
    UpdateGradient();
}
```

この手順では、アプリケーションにコードを追加しました。次の手順では、アプリケーションを実行し、実行時の操作を確認します。

手順 4:アプリケーションの実行

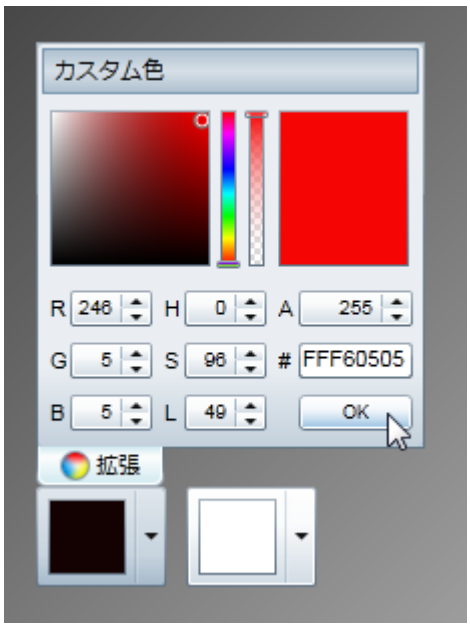
これまでに Silverlight アプリケーションを作成し、外観と動作をカスタマイズしたので、次にアプリケーションを実行します。アプリケーションを実行し、**ColorPicker for Silverlight** の実行時の動作を確認するには、次の手順に従います。

1. **[デバッグ]**メニューから**[デバッグ開始]**を選択して、実行時にアプリケーションがどのように表示されるかを確認します。アプリケーションは次の図のように表示されます。



2. 左のカラーピッカーのドロップダウン矢印をクリックします。ドロップダウンボックスの上にウィンドウが開き、詳細モードのみが表示されることを確認してください。コントロールに対して行った変更が反映されていることがわかります。詳細モードでは、ユーザーは任意の色を指定でき、複数の方法を使って色を選択できます。

3. **[赤]** などの色を選択し、**[OK]**をクリックします。

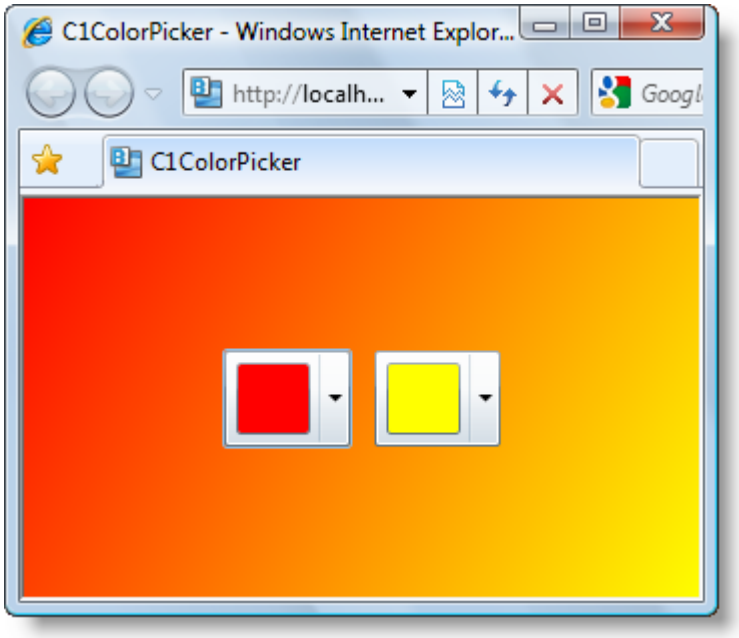


コントロールに対して選択した色と四角形に対して選択したグラデーションが選択内容を反映して変更されていることを確認してください。

4. 右のカラーピッカーのドロップダウン矢印をクリックします。

[基本]タブが表示されていることを確認します(デフォルト設定)。このタブには、**[パレットの色]**、**[標準の色]**、および**[最近使用した色]**が表示されます。任意の色を選択できますが、**[拡張]**タブに切り替えてカスタム色を選択することもできます。選択された色は、赤色の境界線で強調表示されます。

5. **[黄色]**などの色を選択します。選択された色が変化し、四角形の背景グラデーションも選択内容に合わせて変化します。



おめでとうございます!

これで **ColorPicker for Silverlight** クイックスタートは完了です。簡単な Silverlight アプリケーションを作成し、**ColorPicker for Silverlight** コントロールを追加およびカスタマイズし、コントロールの実行時機能をいくつか確認しました。

主な特長

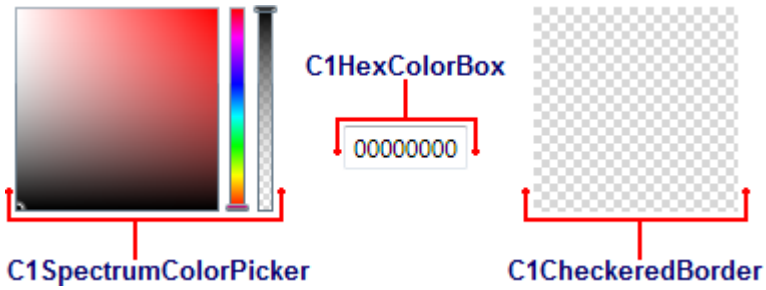
ColorPicker for WPF/Silverlight を使用して、機能豊富でカスタマイズされたアプリケーションを作成できます。以下の主要な機能をうまく利用して、**ColorPicker for WPF/Silverlight** を最大限に活用してください。

- **20 以上の高度な設計の定義済みパレットから選択する**
  ColorPicker には、Microsoft Office で使用されるテーマに適した 20 以上の定義済みカラーパレットがあります。各パレットの色は調和がとれており、これを使用することで、洗練された高度な外観を備えたアプリケーションを作成できます。

- **カスタム色のための組み込みカラーエディタ**
  ColorPicker には、カラーエディタがあります。このエディタを使用することにより、エンドユーザーは、現在のパレットにない色を作成できます。RGB カラーモデルまたは HLS カラーモデルを使用でき、透過性もサポートされています。

- **複数のビュー**
  C1ColorPicker は、色の選択で単純なビューと高度なビューの両方をサポートします。

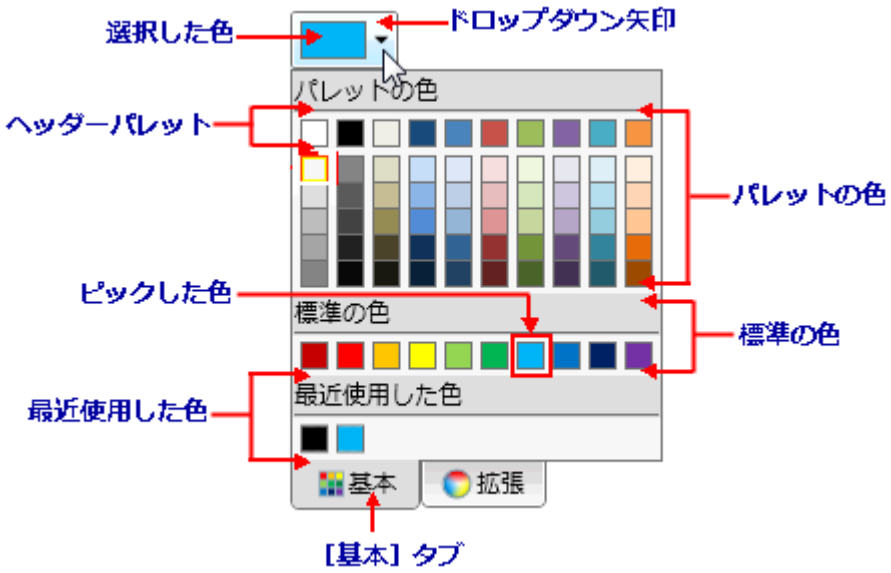
- **構成可能なパーツ**
  このコントロールの各パーツを **C1ColorPicker** とは独立して使用して、カスタムコントロールを作成できます。**C1SpectrumColorPicker** コントロールを使用すると、**C1ColorPicker** コントロールの高度な色選択機能にアクセスできます。**C1HexColorBox** コントロールを使用すると、16 進コードエントリのデータを検証できます。**C1CheckeredBorder** を使用すると、透過色を簡単に表示できます。



- **独自のカスタムパレットを作成する**
  利用可能なカラーパレットがアプリケーションに適していない場合は、独自のカスタムカラーパレットを作成できます。実行時に、指定したカラーパレットからのみ色を選択できるようにユーザーを制限することもできます。

基本 ColorPicker モード

デフォルトでは、**C1ColorPicker** コントロールは、そのドロップダウン矢印をクリックすると**[基本]**タブが開いた状態で開きます。**[基本]**タブの外観の例を次の図に示します。

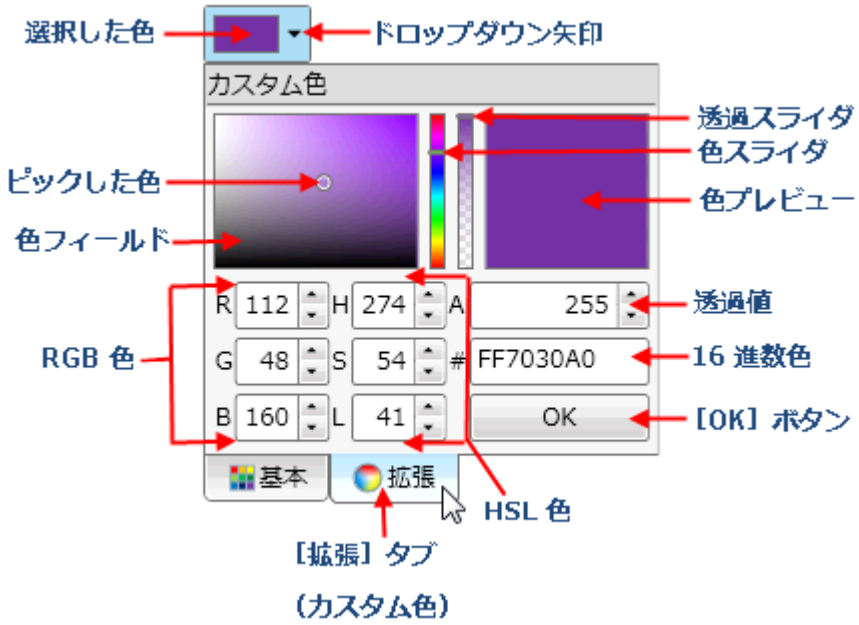


[基本]タブには、次のオプションとセクションがあります。

- **ドロップダウン矢印**: ドロップダウン矢印をクリックして、C1ColorPicker コントロールのウィンドウを開きます。ドロップダウンウィンドウを表示する場所については、「ドロップダウンの方向」を参照してください。
- **[基本]タブ**: 事前選択した色を実行時に使用するには、[基本]タブをクリックします。カスタム色を選択する場合は、[拡張]タブをクリックします。[基本]タブを表示するには、Mode プロパティを Basic または Both に設定する必要があります。
- **選択した色**: 現在選択されている色がカラーピッカーのウィンドウに表示されます。
- **ピックした色**: 現在ピックされている色がカラーリストに赤色の境界線付きで表示されます。
- **パレットの色:** パレットの色には、現在選択されているカラーパレットが反映されます。パレットは、Palette プロパティを設定することによって選択できます。
- **ヘッダーパレット**: これらの色は、パレットの基本色です。パレットの色の拡張リストは、通常、これらの基本色のバリエーションです。
- **標準の色**: 10 色の標準の色が表示されます。標準の色は、暗い赤レンガ色、赤、オレンジ、黄色、薄緑、緑、空色、青、濃紺、および紫です。
- **最近使用した色**: 最近選択した色が最大 10 色まで表示されます。デフォルトでは、このセクションは表示されますが、ShowRecentColors プロパティを **False** に設定すると、最近使用した色を非表示にできます。詳細については、「最近使用した色」を参照してください。

拡張 ColorPicker モード

デフォルトでは、**C1ColorPicker** コントロールは、そのドロップダウン矢印をクリックすると[拡張]タブを使用できる状態で開きます。**[基本]**タブはデフォルトで表示されますが、コントロールの最下部にある[拡張]をクリックすると、**[拡張]**ビューを選択できます。**[拡張]**ビューの外観の例を次の図に示します。



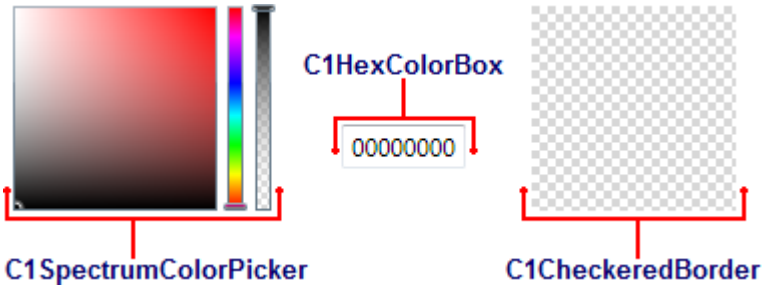
[拡張]タブには、次のオプションとセクションがあります。

- **色フィールド/ピックした色**: [色フィールド]によって色の範囲のトーンを選択できます。[ピックした色]は、現在選択されている色を示します。[色スライダ]を移動して、一般色ファミリを選択してから、[色フィールド]で選択した色を調整します。
- **色スライダ**: このスライダによって色スペクトルから色を選択できます。[色スライダ]を移動して、一般色を選択してから、[色フィールド]で選択した色を調整します。
- **透過スライダ**: このスライダによって色の透過性を設定できます。色は、不透過にすることも、部分的または完全な透過にすることもできます。[透過スライダ]を移動して透過性を選択すると、[透過値]ボックスも変化します。このスライダは、ShowAlphaChannel プロパティが True(デフォルト)に設定されている場合にのみ表示されます。
- **色プレビュー**: 現在選択している色をプレビューします。色の選択が完了したら、[OK]ボタンをクリックし、ドロップダウンボックスを閉じ、指定した色を[選択した色]に設定します。
- **透過値**: このボックスによって色の透過性を設定できます。透過性は、0(完全に透過)と 255(完全に不透過:デフォルト)の間の整数に設定できます。ShowAlphaChannel プロパティが False に設定されている場合、このボックスは淡色表示されます。
- **RGB 色**: 3つの数値ボックスにより、RGB(Red Green Blue)カラーモデルを使用して色を選択できます。
- **HSL 色**: 3つの数値ボックスにより、HSL(Hue Saturation Lightness)カラーモデルを使用して色を選択できます。
- **16 進数色**: 8桁の数値が表示されている場合は、最初の2桁は FF(不透過)から 00(透過)までの色の透過性を表し、下6桁は標準の 16 進数色の選択値を表します。ShowAlphaChannel プロパティが False に設定されていると、下6桁のみが表示されます(透過値は表示されません)。16 進数による色の選択については、「w3schools」を参照してください。
- **[OK]ボタン**: 色の選択が完了したら、[OK]ボタンをクリックし、ドロップダウンボックスを閉じ、指定した色を[選択した色]に設定します。
- **選択した色**: 現在選択されている色がカラーピッカーのウィンドウに表示されます。
- **ドロップダウン矢印**: ドロップダウン矢印をクリックして、C1ColorPicker コントロールのウィンドウを開きます。ドロップダウンウィンドウを表示する場所については、「ドロップダウンの方向」を参照してください。
- **[拡張]タブ**: [拡張]タブをクリックすると、実行時にカスタム色を選択できます。[基本]タブをクリックすると、事前選択された色が表示されます。[拡張]タブを表示するには、Mode プロパティを Advanced または Both に設定する必要が

あります。

その他のコントロール

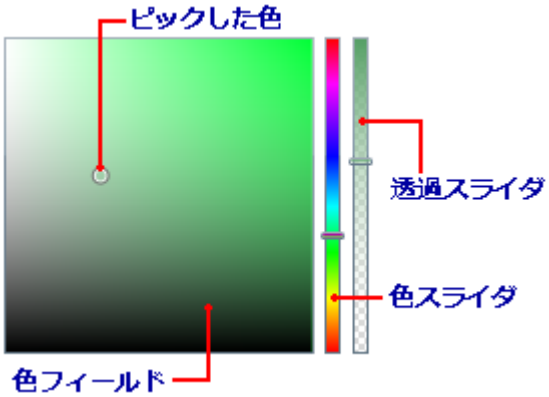
ColorPicker for WPF/Silverlight には、完全な機能を備えた **C1ColorPicker** コントロールのほかに、アプリケーションの色選択をカスタマイズするための C1ColorPicker コントロールのパーツが含まれます。



C1SpectrumColorPicker コントロールを使用すると、**C1ColorPicker** コントロールの高度な色選択機能にアクセスできます。**C1HexColorBox** コントロールを使用すると、16 進コードエントリのデータを検証できます。**C1CheckeredBorder** を使用すると、透過色を簡単に表示できます。以下のトピックでは、これらのパーツについて説明します。

C1SpectrumColorPicker

C1SpectrumColorPicker コントロールを使用すると、C1ColorPicker コントロールの高度な色選択機能にアクセスできます。**C1SpectrumColorPicker** コントロールは、次の図のようになります。



C1SpectrumColorPicker コントロールには、次のオプションとセクションがあります。

- **色フィールド/ピックした色**：**[色フィールド]**によって色の範囲のトーンを選択できます。**[ピックした色]**は、現在選択されている色を示します。[**色スライダ**]を移動して、一般色ファミリを選択してから、**[色フィールド]**で選択した色を調整します。
- **色スライダ**：このスライダによって色スペクトルから色を選択できます。**[色スライダ]**を移動して、一般色を選択してから、[色フィールド]で選択した色を調整します。
- **透過スライダ**：このスライダによって色の透過性を設定できます。色は、不透過にすることも、部分的または完全な透過にすることもできます。**[透過スライダ]**を移動して透過性を選択します。このスライダは、ShowAlphaChannel プロパティが **True**（デフォルト）に設定されている場合にのみ表示されます。

C1HexColorBox

C1HexColorBox コントロールは、16 進コードエントリのデータ検証機能を提供します。たとえば、基本 C1HexColorBox コントロールは、次の図のようになります。

A0FFF700

C1HexColorBox コントロールの外観は、通常のテキストボックスと同じです。デフォルトでは、**C1HexColorBox** コントロールは8桁の値で表示されます。8桁の値が表示されている場合は、最初の2桁は FF（不透過）から 00（透過）までの色の透過性を表し、下6桁は標準の 16 進数色の選択値を表します。16 進数による色の選択については、「w3schools」を参照してください。

ShowAlphaChannel プロパティを **False** に設定すると、下6桁のみが表示されます（透過値は含まれません）。

D45656

必要な場合は、**ShowSharpPrefix** プロパティを **True** に設定し、**C1HexColorBox** コントロールの先頭に '#' 記号を置くこともできます。

#0F00C48E

C1CheckeredBorder

C1CheckeredBorder コントロールを使用すると、アルファ値の設定により、透過性の異なる色を簡単に表示できます。デフォルトでは、コントロールは次の図のようになります。

C1CheckeredBorder コントロールは、透過色と不透過色の両方の色値をサポートします。

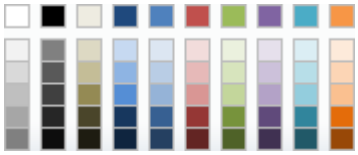
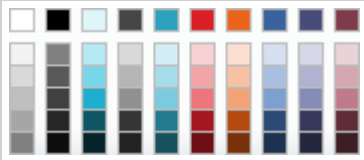
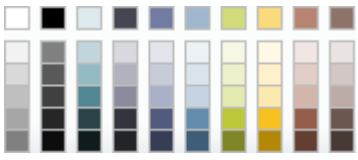
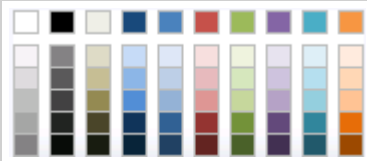
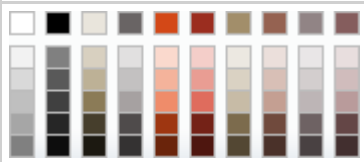
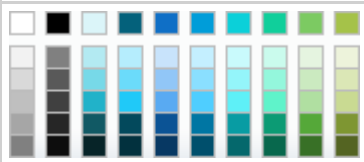
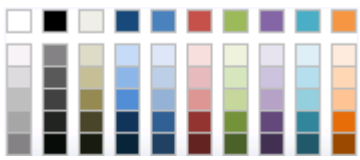
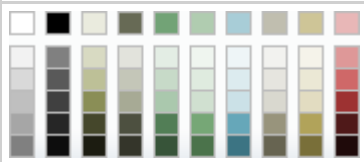
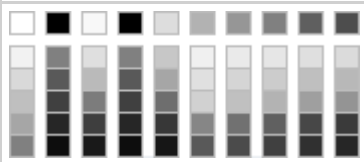
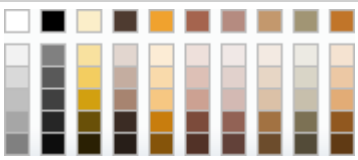
透過	不透過
	

利用可能な ColorPicker のパレット

ColorPicker for WPF/Silverlight には、Microsoft Office で使用されるテーマに適した 20 以上の定義済みカラーパレットがあります。各パレットの色は調和がとれており、これを使用することで、洗練された高度な外観を備えたアプリケーションを作成できます。カラーパレットを変更するには、**Palette** プロパティを設定します。詳細については、「パレットの設定」を参照してください。

次の組み込みパレットを使用できます。

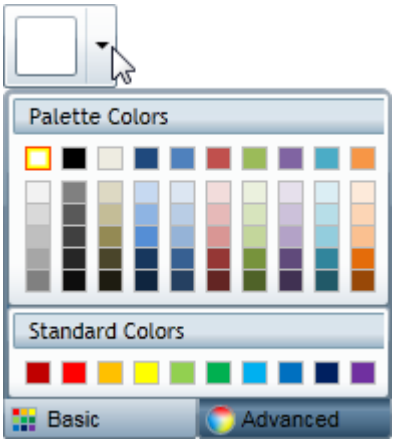
名前	パレット	名前	パレット

ひらめ 	オフィス 
シック 	キュート 
クール 	スパイス 
ビジネス 	アース 
デフォルト 	ペーパー 
ジャパネスク 	フレッシュ 
リゾート 	標準 
エコロジー 	テクノ 
GrayScale 	トラベル 
デザート 	アーバン 
メトロ 	ネオン 

モジュール	

最近使用した色

デフォルトでは、実行時にユーザーが **C1ColorPicker** コントロールの**[基本]**タブを表示する際、選択したカラーパレットと標準のカラーパレットを表示していると、タブには最近選択した色を一覧するセクションが含まれます。



最近使用した色を非表示にすることもできます。ShowRecentColors プロパティは、これらの色を表示するかどうかを設定します。詳細および例については、「最近使用した色の非表示」を参照してください。

ドロップダウンの方向

デフォルトでは、ユーザーが実行時に **C1ColorPicker** コントロールのドロップダウン矢印をクリックすると、コントロールの下にカラーピッカーが表示されます。コントロールの下に表示できない場合は、コントロールの上に表示されます。ただし、**DropDownDirection** プロパティを設定すると、カラーピッカーを表示する場所をカスタマイズできます。

DropDownDirection プロパティは、次のいずれかのオプションに設定できます。

イベント	説明
BelowOrAbove (デフォルト)	ドロップダウン **C1ComboBox** をヘッダーの下に表示するように試みます。表示できない場合は、その上に表示するように試みます。
AboveOrBelow	ドロップダウン **C1ComboBox** をヘッダーの上に表示するように試みます。表示できない場合は、その下に表示するように試みます。
ForceBelow	ドロップダウン **C1ComboBox** をヘッダーの下に強制的に表示します。
ForceAbove	**C1ComboBox** のコンテンツをヘッダーの上に強制的に表示します。

詳細および例については、「ドロップダウンウィンドウの方向の変更」を参照してください。

レイアウトおよび外観

以下のトピックでは、**C1ColorPicker** 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。テーマを使用することで、グリッドの外観をカスタマイズしたり、WPF の XAML ベースのスタイル設定を活用することができます。また、テンプレートを使用して、コントロールを書式設定およびレイアウトしたり、コントロールの操作をカスタマイズすることもできます。

ClearStyle プロパティ

次の表に、**C1ColorPicker** コントロールの ClearStyle をサポートするプロパティと、それらの説明を一覧します。

プロパティ	説明
Background	コントロールの背景を描画するブラシを取得または設定します。デフォルトの **Background** 色は LightBlue です。
FocusBrush	フォーカスがあるコントロールの外観を定義するために使用されるブラシです。
MouseOverBrush	マウスポインタが置かれているコントロールの外観を定義するために使用されるブラシです。
PressedBrush	選択されているコントロールの外観を定義するために使用されるブラシです。

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。

スタイル

ColorPicker for WPF/Silverlight の **C1ColorPicker** コントロールは、コントロールの外観を変更するために使用できるスタイルのプロパティを提供します。これらのスタイルの一部について、次の表で説明します。

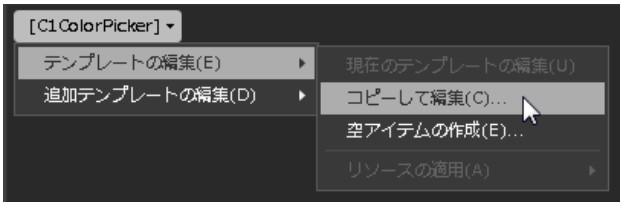
スタイル	説明
ColorContainerStyle	色のセクション（最近使用した色）を表示するために使用する **ItemsControl** のスタイルを設定/取得します。
FontStyle	フォントスタイルを取得または設定します。これは依存プロパティです。
Style	この要素のレンダリング時に使用されるスタイルを取得または設定します。これは依存プロパティです。

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。WPF 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、**ColorPicker 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、C1ColorPicker 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。

ComponentOne ClearStyle 技術

ComponentOne 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 WPF/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。

パネル内のレイアウト

WPF/Silverlight アプリケーションでは、付属するレイアウトプロパティを使用して、C1ColorPicker や他のコントロールを簡単にレイアウトできます。たとえば、Grid パネルでは **Row**、**ColumnSpan**、および **RowSpan** プロパティ、**Canvas** パネルでは **Left** および **Top** プロパティを使用して、コントロールをレイアウトできます。たとえば、**Grid** パネル内に配置された C1ColorPicker コントロールには、次の **Layout** プロパティがあります。

Grid パネル内で、C1ColorPicker コントロールのサイズ、配置、および場所を変更できます。

表示状態

Microsoft Expression Blend で、カスタム状態や状態グループを追加して、ユーザーコントロールの状態ごとに異なる外観を定義できます。たとえば、マウスが置かれたときのコントロールの表示状態を変更できます。 新しいテンプレートを作成し、新しいテンプレートパーツを追加することで、表示状態を表示および編集できます。これで、そのパーツで利用可能な表示状態が**[表示状態]**ウィンドウに表示されます。

よく使用される状態としては、項目の通常の外観を示す **Normal**、マウスが置かれている項目を示す **MouseOver**、有効でない項目を示す **Disabled** などがあります。フォーカスの状態には、項目にフォーカスがないときの **Unfocused**、項目にフォーカスがあるときの **Focused** などがあります。

XAML 要素

ColorPicker for WPF/Silverlight をインストールすると、いくつかの補助 XAML 要素が同時にインストールされます。これらの要素にはテンプレートやテーマが含まれており、**ColorPicker for WPF/Silverlight** インストールディレクトリに格納されています。これらの要素をプロジェクトに組み込んで、たとえば、デフォルトのテーマに基づく独自のテーマを作成できます。

含まれる補助 XAML 要素

ColorPicker for WPF/Silverlight には、次の補助 XAML 要素が含まれています。

要素	フォルダ	説明
generic.xaml	XAML	コントロールのさまざまなスタイルと初期スタイルのテンプレートを指定します。

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、**C1ColorPicker** 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。**ColorPicker 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、**ColorPicker for WPF/Silverlight** 製品を使用して特定のタスクを行うための方法を提供します。

また、タスク別ヘルプトピックは、新しい WPF 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。

パレットの設定

ColorPicker for WPF/Silverlight には、Microsoft Office で使用されるテーマに適した 20 以上の定義済みカラーパレットがあります。パレットの選択の詳細については、「利用可能な ColorPicker のパレット」を参照してください。カラーパレットを変更するには、**Palette** プロパティを設定します。

Palette プロパティを設定するには、次の手順に従います。

1. ツールボックスに移動し、ボタンアイコンをダブルクリックして、コントロールをプロジェクトに追加します。
2. フォーム上でボタンのサイズと位置を変更します。
3. [プロパティ]ウィンドウに移動し、ボタンの Content プロパティを「パレットの変更」に設定します。
4. ボタンをダブルクリックしてコードビューに切り替え、Button_Click イベントハンドラを作成します。
5. Button_Click イベントハンドラにコードを追加します。次のようになります。

Visual Basic

```
Private Sub Button1_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)
    ' カラーパレットを設定します。
    Me.C1ColorPicker.Palette =
ColorPalette.GetColorPalette(Office2007ColorTheme.GrayScale)
End Sub
```

C#

```
private void button1_Click(object sender, System.Windows.RoutedEventArgs e){
// カラーパレットを変更します。    this.c1ColorPicker.Palette =
ColorPalette.GetColorPalette(Office2007ColorTheme.GrayScale);}
```

アプリケーションの実行と動作の確認

次のことを確認します。

1. **C1ColorPicker** コントロールのドロップダウン矢印をクリックして、デフォルトパレットが表示されることを確認します。
2. [パレットの変更]ボタンをクリックし、**C1ColorPicker** コントロールのドロップダウン矢印をもう一度クリックします。グレースケールパレットが表示されることを確認します。

カスタムパレットの作成

ColorPicker for WPF/Silverlight には、Microsoft Office で使用されるテーマに合わせた 20 以上の定義済みカラーパレットがありますが、定義済みパレットを使用する代わりに独自のカスタムカラーパレットを作成することもできます。次の手順では、カスタムパレットを作成し、ボタンをクリックしたときにそのパレットが **C1ColorPicker** コントロールに適用されるようにします。

カスタムパレットを作成するには、次の手順に従います。

1. ツールボックスに移動し、ボタンアイコンをダブルクリックして、コントロールをプロジェクトに追加します。
2. フォーム上でボタンのサイズと位置を変更します。
3. **[プロパティ]**ウィンドウに移動し、ボタンの **Content** プロパティを「パレットの変更」に設定します。
4. ボタンをダブルクリックしてコードビューに切り替え、**Button_Click** イベントハンドラを作成します。
5. **Button_Click** イベントハンドラにコードを追加します。次のようになります。

Visual Basic

```
Private Sub Button1_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)
    ' カラーパレットを設定します。
    Dim cp1 as New ColorPalette("Pittsburgh")
    cp1.Clear()
    cp1.Add(Color.FromArgb(255, 0, 0, 0))
    cp1.Add(Color.FromArgb(255, 99, 107, 112))
    cp1.Add(Color.FromArgb(255, 255, 255, 255))
    cp1.Add(Color.FromArgb(255, 247, 181, 18))
    cp1.Add(Color.FromArgb(255, 253, 200, 47))
    cp1.Add(Color.FromArgb(255, 43, 41, 38))
    cp1.Add(Color.FromArgb(255, 149, 123, 77))
    cp1.Add(Color.FromArgb(255, 209, 201, 157))
```

```
    cp1.Add(Color.FromArgb(255, 0, 33, 71))
    cp1.Add(Color.FromArgb(255, 99, 177, 229))
    c1ColorPicker1.Palette = cp1
End Sub
```

C#

```
private void button1_Click(object sender, System.Windows.RoutedEventArgs e)
{
    // カラーパレットを設定します。
    ColorPalette cp1 = new ColorPalette("Pittsburgh");
    cp1.Clear();
    cp1.Add(Color.FromArgb(255, 0, 0, 0));
    cp1.Add(Color.FromArgb(255, 99, 107, 112));
    cp1.Add(Color.FromArgb(255, 255, 255, 255));
    cp1.Add(Color.FromArgb(255, 247, 181, 18));
    cp1.Add(Color.FromArgb(255, 253, 200, 47));
    cp1.Add(Color.FromArgb(255, 43, 41, 38));
    cp1.Add(Color.FromArgb(255, 149, 123, 77));
    cp1.Add(Color.FromArgb(255, 209, 201, 157));
    cp1.Add(Color.FromArgb(255, 0, 33, 71));
    cp1.Add(Color.FromArgb(255, 99, 177, 229));
    c1ColorPicker1.Palette = cp1;
}
```

ボタンをクリックすると、**ColorPicker** のカラーパレットがカスタムパレットに変更されます。

アプリケーションの実行と動作の確認

次のことを確認します。

1. **C1ColorPicker** コントロールのドロップダウン矢印をクリックして、デフォルトパレットが表示されることを確認します。
2. [パレットの変更]ボタンをクリックし、**C1ColorPicker** コントロールのドロップダウン矢印をもう一度クリックします。カスタムパレットが表示されることを確認します。

背景色の変更

Background プロパティは、**C1ColorPicker** コントロールの背景色の値を取得または設定します。デフォルトでは、**C1ColorPicker** コントロールには **Background** プロパティが設定されません。ただし、これを設計時、XAML、およびコードでカスタマイズできます。

Blend での設計時

実行時に **Background** プロパティを設定するには、次の手順に従います。

1. **C1ColorPicker** コントロールをクリックして選択します。
2. **[プロパティ]**ウィンドウに移動します。**[背景]**ドロップダウン矢印をクリックし、カラーピッカーで**赤**などの色を選択します。

XAML の場合

たとえば、**Background** プロパティを **Red** に設定するには、<c1:C1ColorPicker> タグに Background="Red" を追加します。次のようになります。

XAML

```
<c1:C1ColorPicker Name="C1ColorPicker1" Margin="296,98,273,0" Height="45"
VerticalAlignment="Top" Background="Red"/7gt;
```

コードの場合

たとえば、**Background** プロパティを **Red** に設定するには、プロジェクトに次のコードを追加します。

Visual Basic

```
Me.C1ColorPicker1.Background = System.Windows.Media.Brushes.Red
```

C#

```
this.c1ColorPicker1.Background = System.Windows.Media.Brushes.Red;
```

アプリケーションの実行と動作の確認

C1ColorPicker コントロールの背景が赤で表示されます。

ドロップダウンウィンドウの方向の変更

デフォルトでは、ユーザーが実行時に **C1ColorPicker** コントロールのドロップダウン矢印をクリックすると、コントロールの下にカラーピッカーが表示されます。コントロールの下に表示できない場合は、コントロールの上に表示されます。ただし、カラーピッカーを表示する場所をカスタマイズできます。ドロップダウン矢印の方向の詳細については、「ドロップダウンの方向」を参照してください。

Blend での設計時

ドロップダウンウィンドウの方向を実行時に変更するには、次の手順に従います。

1. **C1ColorPicker** コントロールをクリックして選択します。
2. [プロパティ]ウィンドウに移動し、**DropDownDirection** ドロップダウン矢印をクリックします。
3. ForceAbove などのオプションを選択します。これにより、**DropDownDirection** プロパティが選択したオプションに設定されます。

XAML の場合

たとえば、ドロップダウンウィンドウの方向を変更するには、<c1:C1ColorPicke> タグに DropDownDirection="ForceAbove" を追加します。次のようになります。

XAML

```
<c1:C1ColorPicker Name="C1ColorPicker1" Margin="296,98,273,0" Height="45"
VerticalAlignment="Top" DropDownDirection="ForceAbove"/>
```

コードの場合

たとえば、ドロップダウンウィンドウの方向を変更するには、プロジェクトに次のコードを追加します。

Visual Basic

```
Me.C1ColorPicker1.DropDownDirection = DropDownDirection.ForceAbove
```

C#

```
this.c1ColorPicker1.DropDownDirection = DropDownDirection.ForceAbove;
```

これで、**DropDownDirection** プロパティが **ForceAbove** に設定されます。

アプリケーションの実行と動作の確認

C1ColorPicker コントロールのドロップダウン矢印をクリックすると、コントロールの上にドロップダウンウィンドウが表示されます。

最近使用した色の非表示

デフォルトでは、**C1ColorPicker** コントロールのカラーピッカーウィンドウの[基本]タブに、最近使用した色が表示されます。詳細については、「最近使用した色」を参照してください。XAML およびコードでの設計時に、最近使用した色を表示しないように選択することもできます。

Blend での設計時

最近使用した色が実行時に表示されないようにするには、次の手順に従います。

1. **C1ColorPicker** コントロールをクリックして選択します。
2. **[プロパティ]**ウィンドウに移動します。
3. **ShowRecentColors** プロパティを見つけ、これを **False** に設定します。

XAML の場合

最近使用した色が実行時に表示されないようにするには、<c1:C1ColorPicker> タグに ShowRecentColors="False" を追加します。次のようになります。

XAML

```
<c1:C1ColorPicker Name="C1ColorPicker1" Margin="296,98,273,0" Height="45"
VerticalAlignment="Top" ShowRecentColors="False" />
```

コードの場合

最近使用した色が表示されないようにするには、プロジェクトに次のコードを追加します。

Visual Basic

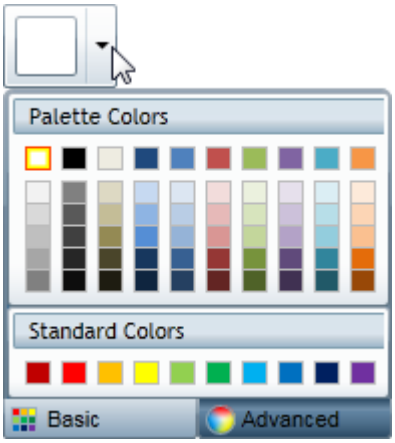
```
Me.C1ColorPicker1.ShowRecentColors = False
```

C#

```
this.c1ColorPicker1.ShowRecentColors = false;
```

アプリケーションの実行と動作の確認

C1ColorPicker コントロールのドロップダウン矢印をクリックして、[基本]タブに最近使用した色が表示されないことを確認します。



CoverFlow (Silverlight のみ)

CoverFlow for Silverlight を使用すると、アニメーション表示される 3D グラフィカルユーザーインターフェイスで、視覚的に項目を操作することができます。スクロールバーをスクロールさせることで、カバーフローを操作することができます。

メモ:このセクションの内容は、ComponentOne for Silverlight にのみ適用されます。

クイックスタート

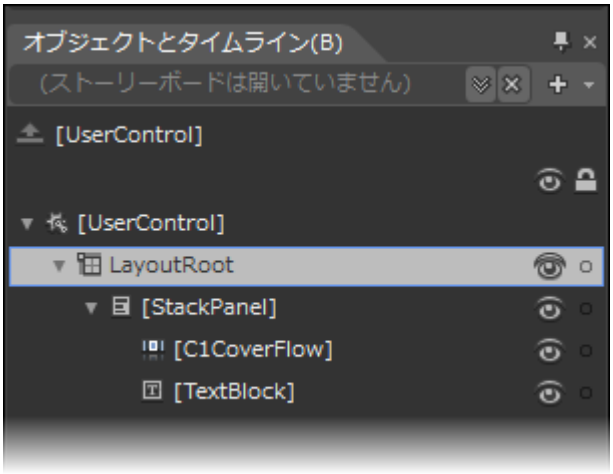
このクイックスタートは、**CoverFlow for Silverlight** を初めて使用するユーザーのために用意されています。このクイックスタートでは、最初に Expression Blend で **C1CoverFlow** コントロールを含む新しいプロジェクトを作成します。コントロールをプロジェクトに追加したら、いくつかのプロパティを設定し、コンテンツパネルにコンテンツを追加して、コントロールをカスタマイズします。その後、プロジェクトを実行して、このクイックスタートで作成したプロジェクトの結果を確認します。

手順 1:アプリケーションの作成

この手順では、最初に Expression Blend で **CoverFlow for Silverlight** を使用する Silverlight アプリケーションを作成します。また、**StackPanel** コントロールと **TextBlock** コントロール、および3つの画像を含むフォルダをプロジェクトに追加します。

次の手順に従います。

1. Expression Blend で、**[ファイル]**→**[新しいプロジェクト]**を選択します。
2. **[新しいプロジェクト]**ダイアログボックスで、左ペインからプロジェクトの種類として[Silverlight]を選択し、右ペインから**[Silverlight アプリケーション + Web サイト]**を選択します。
3. プロジェクトの**[名前]**と**[場所]**を入力し、ドロップダウンボックスで[言語]を選択し、**[OK]**をクリックします。Blend によって作成された新しいアプリケーションが開き、デザインビューに **MainPage.xaml** ファイルが表示されます。
4. 次の手順に従って、**StackPanel** コントロールをプロジェクトに追加します。
    a. メニューから**[ウィンドウ]**→**[アセット]**を選択して**[アセット]**パネルを開きます。
    b. **[アセット]**パネルで、検索バーに**「StackPanel」**と入力します。StackPanel コントロールのアイコンが表示されます。
    c. **[StackPanel]**アイコンをダブルクリックしてコントロールをプロジェクトに追加します。
5. 次の手順に従って、**C1CoverFlow** コントロールを StackPanel コントロールに追加します。
    a. **[オブジェクトとタイムライン]**パネルで**[StackPanel]**を選択します。
    b. **[アセット]**パネルで、検索バーに「C1CoverFlow」と入力します。**C1CoverFlow** コントロールのアイコンが表示されます。
    c. **C1CoverFlow** アイコンをダブルクリックして、コントロールを **StackPanel** に追加します。
6. 次の手順に従って、TextBlock コントロールを StackPanel コントロールに追加します。
    a. **[オブジェクトとタイムライン]**パネルで**[StackPanel]**を選択します。
    b. **[アセット]**パネルで、検索バーに「TextBlock」と入力します。**TextBlock** コントロールのアイコンが表示されます。
    c. **[TextBlock]**アイコンをダブルクリックして、コントロールを **StackPanel** に追加します。**[オブジェクトとタイムライン]**タブのレイアウト階層は次のようになります。



7. 次の手順に従って、プロジェクトに画像を追加します。

   a. **[プロジェクト]**パネルでプロジェクトを右クリックしてコンテキストメニューを開き、**[新しいフォルダーの追加]**を選択します。新しいフォルダには「Images」と名前を付けます。

   b. **Images** フォルダを右クリックし、[既存のアイテムの追加]を選択します。**[既存のアイテムの追加]**ダイアログボックスが開きます。

   c. 次の場所に移動します。
   Documents\ComponentOne Samples\Silverlight\QuickStart\QuickStart

   d. 3つの適当な画像を選択します。

   e. **[開く]**をクリックして**[既存のアイテムの追加]**ダイアログボックスを閉じ、画像をフォルダに追加します。画像が **Images** フォルダに追加されます。

この手順では、プロジェクトを作成し、それに **C1CoverFlow**、**StackPanel**、および **TextBlock** コントロールを追加しました。また、3つの画像を含むフォルダもプロジェクトに追加しました。次の手順では、いくつかのプロパティを設定してコントロールをカスタマイズします。

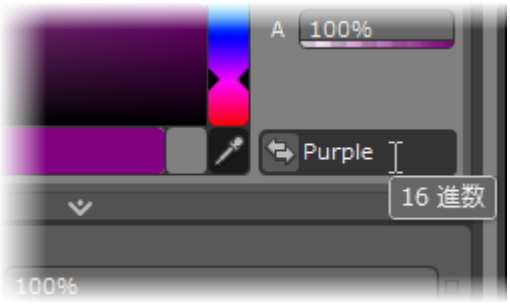
手順 2:コントロールの設定

前の手順では、Blend で、3つのコントロール(**C1CoverFlow** コントロール、**StackPanel** コントロール、**TextBlock** コントロール)と画像のフォルダを含む Silverlight プロジェクトを作成しました。この手順では、いくつかのプロパティを設定して、プロジェクトに追加した3つのコントロールをカスタマイズします。

次の手順に従います。

1. **[オブジェクトとタイムライン]**パネルで**[StackPanel]**を選択し、**[プロパティ]**ウィンドウで次のプロパティを設定します。
   - **Width** プロパティを "Auto" に設定します。
   - **Height** プロパティを "Auto" に設定します。
2. [オブジェクトとタイムライン]パネルで[C1CoverFlow]を選択し、[プロパティ]ウィンドウで次のプロパティを設定します。
   - **Name** プロパティを "C1CoverFlow1" に設定します。
   - **Width** プロパティを "470" に設定します。
   - **Height** プロパティを "250" に設定します。
3. **[オブジェクトとタイムライン]**パネルで**[TextBlock]**を選択し、次のプロパティを設定します。

- **Name** プロパティを "TextBlock1" に設定します。
- **Text** プロパティを "ここに製品情報が表示されます" に設定します。
- **[16 進数]**テキストボックスに「Purple」と入力して[Enter]キーを押し、**Foreground** プロパティを紫に設定します。



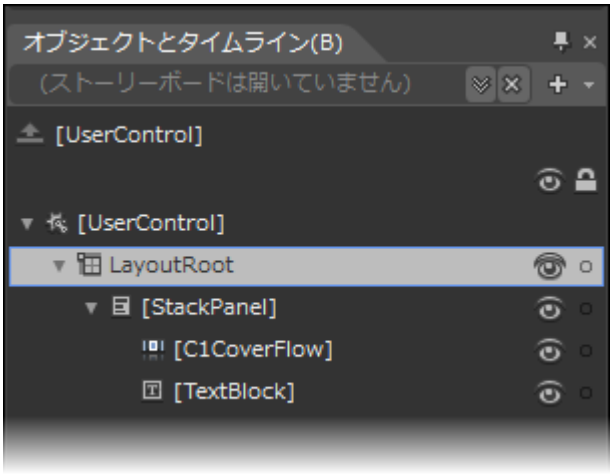
この手順では、**StackPanel**、**C1CoverFlow**、および **TextBlock** コントロールをカスタマイズしました。次の手順では、**C1CoverFlow** コントロールに項目を追加します。

手順 3:項目の追加

前の手順では、いくつかのプロパティを設定し、**StackPanel**、**C1CoverFlow**、および **TextBlock** コントロールをカスタマイズしました。この手順では、4つの項目、具体的には **TextBox** コントロールと3つの画像を **C1CoverFlow** コントロールに追加します。また、プロパティを設定して、各項目をカスタマイズします。

次の手順に従います。

1. 次の手順に従って、TextBox コントロールを **C1CoverFlow** コントロールに追加します。
   a. **[オブジェクトとタイムライン]**パネルで **C1CoverFlow1** を選択します。
   b. [アセット]パネルで、検索バーに「TextBox」と入力します。**TextBox** コントロールのアイコンが表示されます。
   c. **[TextBox]**アイコンをダブルクリックして、コントロールを **C1CoverFlow** コントロールに追加します。
2. 次の手順に従って、コントロールに最初の画像を追加します。
   a. **[オブジェクトとタイムライン]**パネルで **C1CoverFlow1** を選択します。
   b. **[プロジェクト]**パネルで、**Images** フォルダを展開します(まだ展開されていない場合)。
   c. **cover1.jpg** をダブルクリックして、画像を **C1CoverFlow** コントロールに追加します。
   d. **cover2.jpg** をダブルクリックして、画像を **C1CoverFlow** コントロールに追加します。
   e. **cover3.jpg** をダブルクリックして、画像を **C1CoverFlow** コントロールに追加します。**[オブジェクトとタイムライン]**タブのレイアウトは次のようになります。



3. [オブジェクトとタイムライン]パネルで[TextBox]を選択し、[プロパティ]ウィンドウで次のプロパティを設定します。
   - [16 進数]テキストボックスに「#FFB3AED8」と入力して、Background プロパティをうす紫色に設定します。
   - **Width** プロパティを "150" に設定します。
   - **Height** プロパティを "150" に設定します。
   - プロパティを次の文字列に設定します。"製品画像のリストをスクロールすると、選択された製品の名前がテキストボックスに表示されます。"
4. [オブジェクトとタイムライン]パネルで[Image]"cover1" を選択し、[プロパティ]ウィンドウで次のプロパティを設定します。
   - **Width** プロパティを "150" に設定します。
   - **Height** プロパティを "150" に設定します。
5. [オブジェクトとタイムライン]パネルで[Image]"cover2" を選択し、[プロパティ]ウィンドウで次のプロパティを設定します。
   - **Width** プロパティを "150" に設定します。
   - **Height** プロパティを "150" に設定します。
6. [オブジェクトとタイムライン]パネルで[Image]"cover3" を選択し、[プロパティ]ウィンドウで次のプロパティを設定します。
   - **Width** プロパティを "150" に設定します。Height プロパティを "150" に設定します。

この手順では、**C1CoverFlow** コントロールに4つの項目を追加した後、項目のプロパティを設定しました。次の手順では、プロジェクトにコードを追加します。

手順 4:コードの追加

前の手順では、**C1CoverFlow** コントロールに4つの項目を追加し、項目のプロパティを設定しました。この手順では、**SelectedIndexChanged** イベントハンドラを作成し、そのイベントハンドラに、**SelectedIndex** プロパティの値をチェックするコードを追加します。「手順 1:アプリケーションの作成」で追加した **TextBlock** コントロールは、ユーザーが製品画像を選択すると、**Text** プロパティの文字列値が変更されます。

次の手順に従います。

1. [オブジェクトとタイムライン]パネルで C1CoverFlow1 を選択します。

2. [プロパティ]ウィンドウで、[イベント]ボタンをクリックします。
3. **SelectedIndexChanged** イベントハンドラを見つけ、テキストボックス内をダブルクリックします。
   C1CoverFlow1_SelectedIndexChanged イベントハンドラがプロジェクトに追加されます。
4. コードコメントを次のコードに置き換えます。

Visual Basic

```
If C1CoverFlow1.SelectedIndex = 1 Then
    TextBlock1.Text = "ComponentOne Studio Enterprise"
ElseIf C1CoverFlow1.SelectedIndex = 2 Then
    TextBlock1.Text = "SPREAD for Windows Forms"
ElseIf C1CoverFlow1.SelectedIndex = 3 Then
    TextBlock1.Text = "ActiveReports for .NET"
Else
    TextBlock1.Text = Nothing
End If
```

C#

```
if (C1CoverFlow1.SelectedIndex == 1)
{
    TextBlock1.Text = "ComponentOne Studio Enterprise";
}
else if (C1CoverFlow1.SelectedIndex == 2)
{
    TextBlock1.Text = "SPREAD for Windows Forms";
}
else if (C1CoverFlow1.SelectedIndex == 3)
{
    TextBlock1.Text = "ActiveReports for .NET";
}
else
{
    TextBlock1.Text = null;
}
```

この手順では、イベントハンドラとコードをプロジェクトに追加しました。次の手順では、プログラムを実行して、このクイックスタートのトピックの結果を確認します。

手順 5：アプリケーションの実行

ここまでの4つの手順では、**StackPanel**、**C1CoverFlow**、および **TextBlock** コントロールを含むプロジェクトを作成し、コントロールのプロパティを設定した後、項目を **C1CoverFlow** コントロールに追加し、コードをプロジェクトに追加しました。後は、プロジェクトを実行するだけです。

次の手順に従います。

1. [F5]キーを押してプロジェクトを実行し、次のようにプロジェクトが表示されることを確認します。

ここに製品情報が表示されます

2. スクロールバーの[次へ]ボタンをクリックして選択項目を変更し、選択した製品名が **TextBlock** に表示されることを確認します。

ActiveReports for .NET

3. スクロールバーの[次へ]ボタンをもう一度クリックして選択項目を変更し、選択した製品名が **TextBlock** に表示されることを確認します。

SPREAD for Windows Forms

4. スクロールバーの[次へ]ボタンをもう一度クリックして選択項目を変更し、選択した製品名が **TextBlock** に表示されることを確認します。

ComponentOne Studio Enterprise

おめでとうございます!

これで、**CoverFlow for Silverlight** クイックスタートは終了です。コントロールの詳細については、「C1CoverFlow コントロールの基本overFlow コントロールの基本」と「タスク別ヘルプ」を参照してください。

主な特長

CoverFlow for 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。次の主な特長を利用して、**CoverFlow for Silverlight** を最大限に活用してください。

- **項目角度の設定**
  選択されていない画像を遠近法の角度で表示することで、アルバムが積み重ねられているように見える効果を作成できます。遠近法の角度は、1つのプロパティを設定するだけで簡単に変更できます。詳細については、「項目角度」を参照してください。

- **項目間距離の設定**
  項目間の距離および視点から項目までの距離を変更できます。詳細については、「視点距離」、「選択項目の距離」、および「項目間距離」を参照してください。

- **項目の速度の設定**
  ユーザーが項目を選択したりコントロールをスクロールする際の速度を制御できます。項目の角度が変化する速度や、カメラの位置が変わる速度を制御することもできます。詳細については、「速度の設定」を参照してください。
- **項目サイズの設定**
  カバーのサイズを簡単に設定できます。
- **仮想化**
  C1CoverFlow 内の大量の項目を簡単に表示できます。画像がオンデマンドで読み込まれます。
- **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、それを使って UI にスタイルを追加できます。

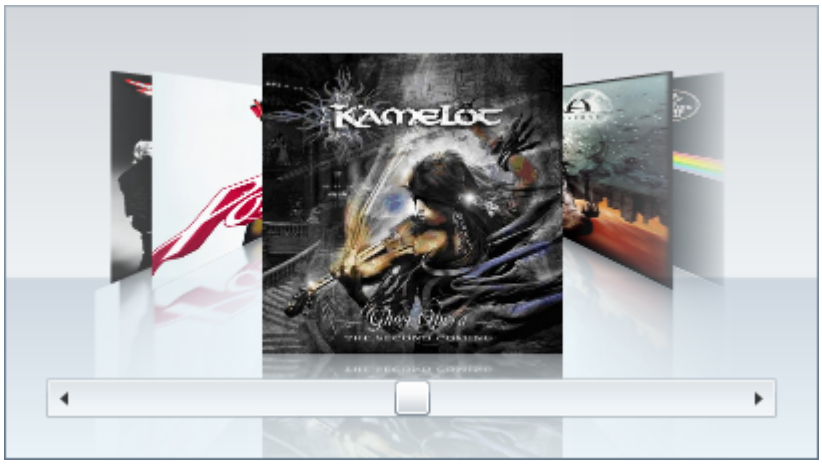
視点距離

視点距離とは、視点から **C1CoverFlow** 項目までの距離です。視点距離により、透視投影される項目の遠近の程度が決まります。

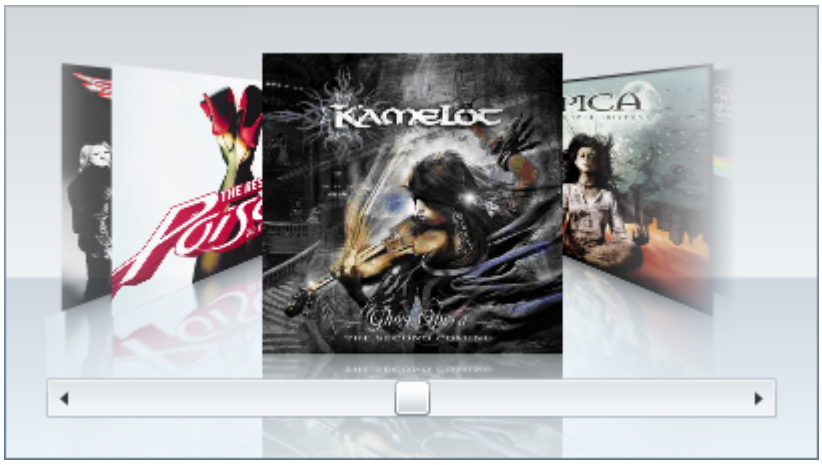
視点距離を変更するには、**EyeDistance** プロパティを設定します。このプロパティの値は、現在のカバーサイズに対する相対的な値です。つまり、値 0.5 に設定すると、視点距離は CoverFlow 項目のサイズの半分になります。デフォルトでは、このプロパティは値 1.5 に設定されます。

次に、**EyeDistance** プロパティの設定例を示します。

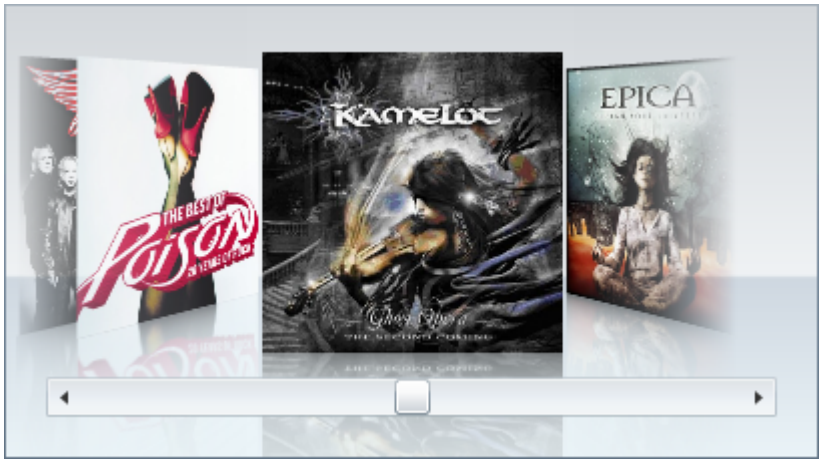
EyeDistance = 0.5



EyeDistance = 1



EyeDistance = 3

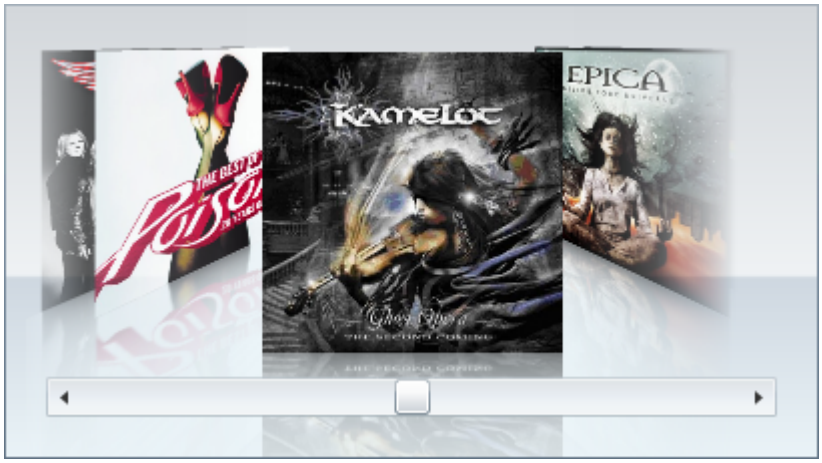


視点高さ

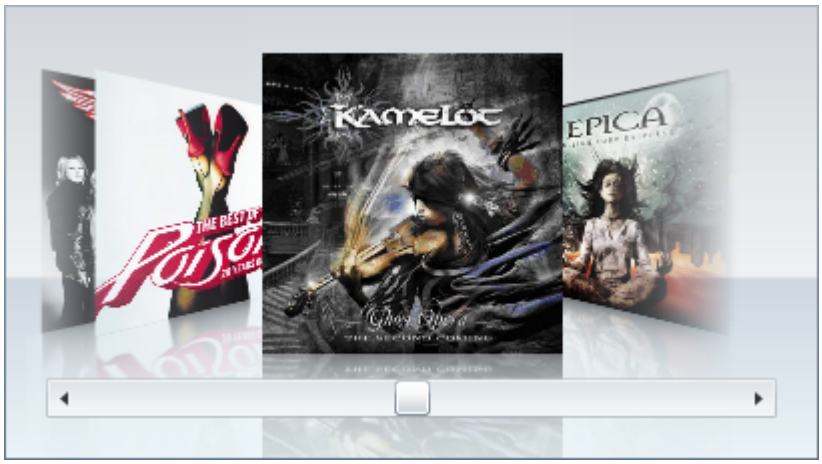
EyeHeight プロパティは、カメラが置かれる高さを設定し、それに応じて項目の表示を調整します。このプロパティの値は、項目サイズに対する相対的な値です。つまり、値 0.5 では、カメラは垂直方向の中央に配置されます。値 1 ではカメラは下端に、値 0 では上端に配置されます。このプロパティのデフォルト値は、0.25 です。

次に、**EyeHeight** プロパティの設定例を示します。

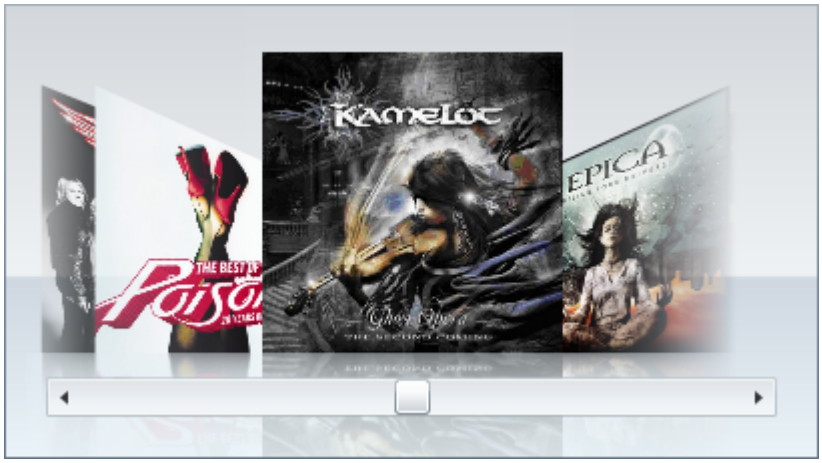
EyeHeight = 0



EyeHeight = 0.5



EyeHeight= 1

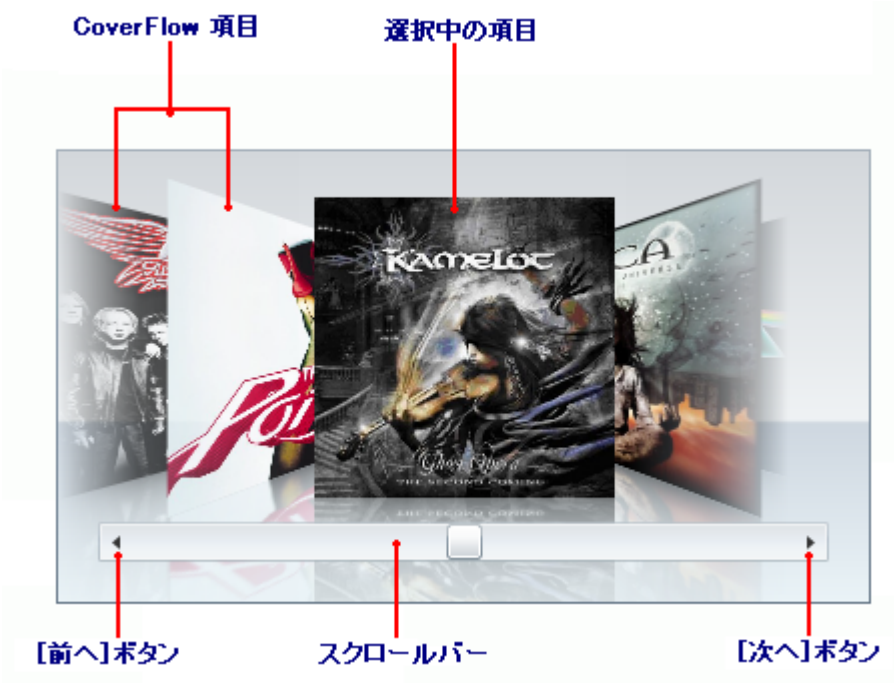


C1CoverFlow コントロールの基本

C1CoverFlow は、アニメーション表示される 3D ユーザーインターフェイスです。項目コントロールなので、テキスト、画像、コントロールを保持できます。

ユーザーは、スクロールバーを使って **C1CoverFlow** コントロール内をブラウズできます。または、そのままリスト内の項目をクリックすることもできます。リストの項目をクリックすると、その項目がスライドしてコントロールの中央に移動し、その項目にフォーカスが移り、それと共に他の項目も表示されます。

次の図に、**C1CoverFlow** コントロールの要素を示します。



- **CoverFlow 項目**
  テキスト、画像、コントロールなど任意の要素を CoverFlow 項目にすることができます。CoverFlow 項目は簡単に作成できます。子項目を **C1CoverFlow** コントロールに追加するだけです。

- **選択中の項目**
  選択された項目は、常にコントロールの中央に表示されます。ユーザーが項目をスクロールすると、選択された項目はそれに伴って変わります。ユーザーは、項目をクリックして選択することもできます。選択された項目を変更するには、**SelectedIndex** プロパティを設定します。

- **スクロールバー**
  スクロールバーを使用すると、ユーザーは CoverFlow 項目のリストをスクロールできます。スクロールバーは、**C1CoverFlow** テンプレートで削除できます(「スクロールバーを削除する」を参照)。
- **[前へ]ボタン**
  [前へ]ボタンを使用すると、一度に1つ左の項目にスクロールできます。[前へ]ボタンはスクロールバー項目の一部です。つまり、スクロールバーを削除すると、[前へ]ボタンも削除されます。
- **[次へ]ボタン**
  [次へ]ボタンを使用すると、一度に1つ右の項目にスクロールできます。[次へ]ボタンはスクロールバー項目の一部です。つまり、スクロールバーを削除すると、[次へ]ボタンも削除されます。

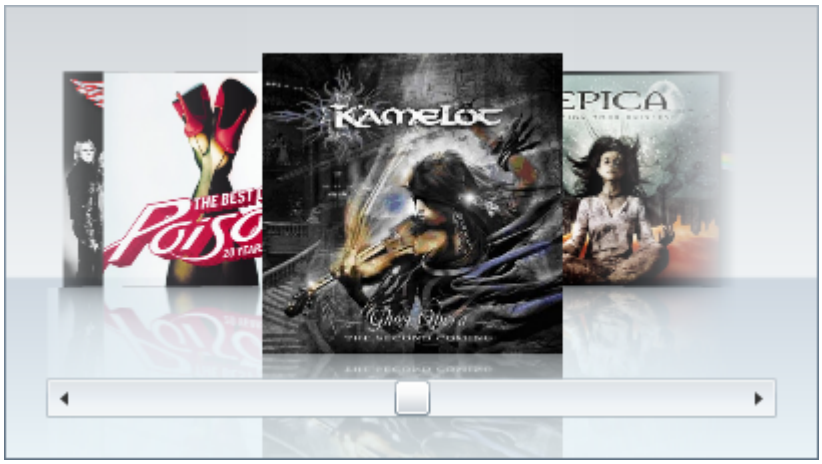
以下のトピックでは、**C1CoverFlow** コントロールのいくつかの機能について概要を説明します.

項目角度

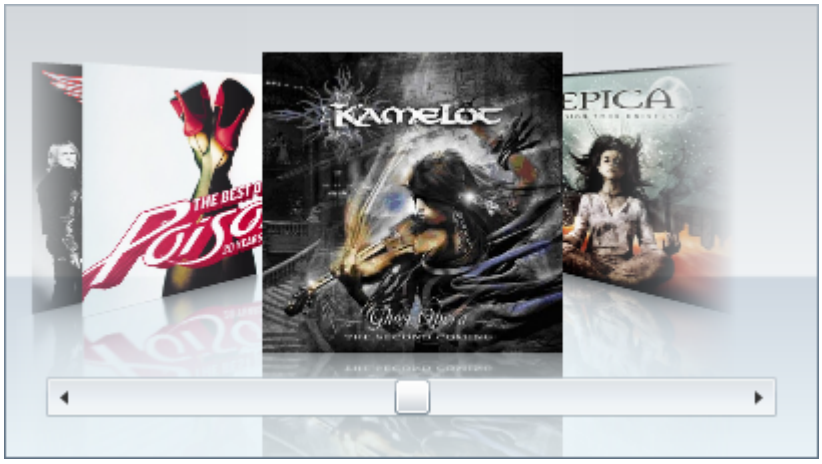
ItemAngle プロパティは、コントロールの選択されていない項目の角度を設定します。ItemAngle プロパティの値は度単位の角度として解釈されるので、ItemAngle プロパティを "180" に設定すると、画像は水平方向に反転します。デフォルトの項目角度は 60 で、これは、選択されていない項目を 60 度の角度に回転させます。

次に、**ItemAngle** プロパティの設定例を示します。

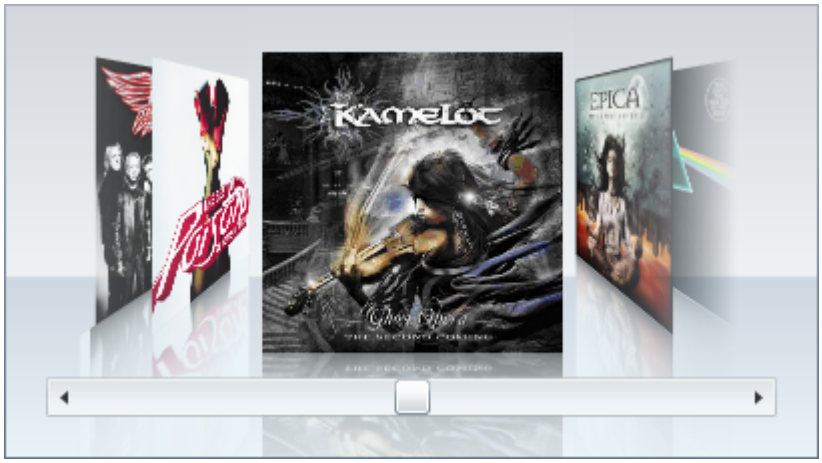
ItemAngle = 1



ItemAngle = 45



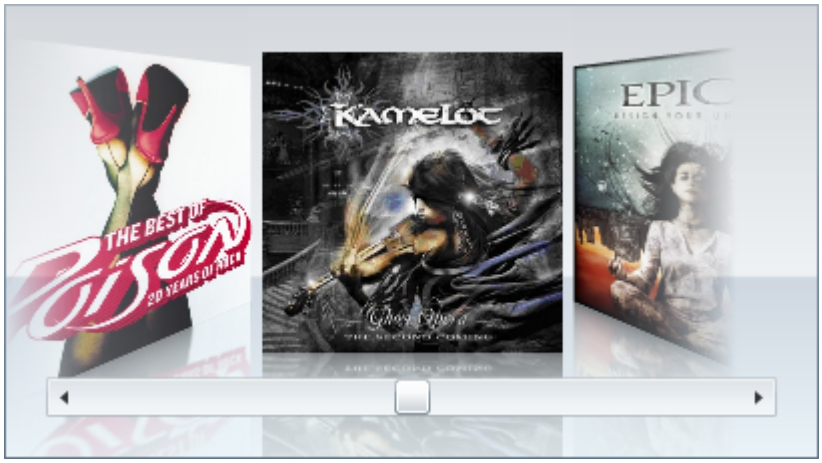
ItemAngle = 90



選択項目のオフセット

SelectedItemOffset プロパティは、見ている人にとって、選択されている項目が選択されていない項目に比べてどの程度近くなるかを決定します。このプロパティの値は、現在の項目サイズに対する相対的な値です。つまり、値 0.5 の場合、項目はその項目のサイズの半分の距離だけオフセットされます。

次に、**SelectedItemOffset** プロパティの設定例を示します。

SelectedItemOffset = 0

SelectedItemOffset = .5

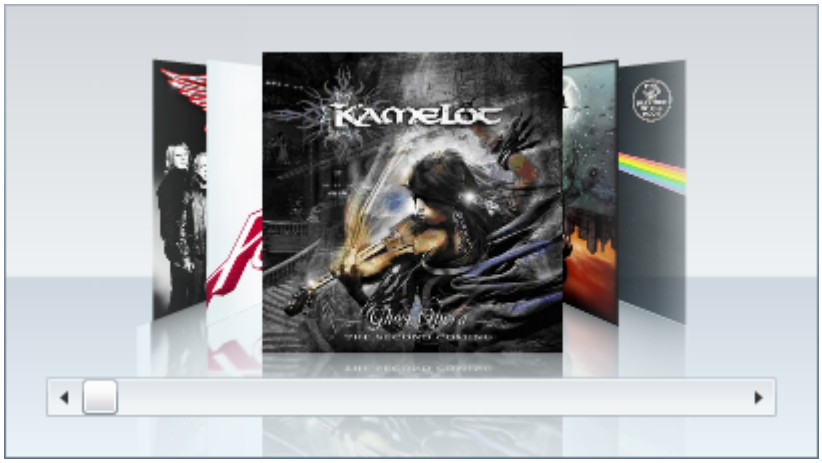
SelectedItemOffset = 1

選択項目の距離

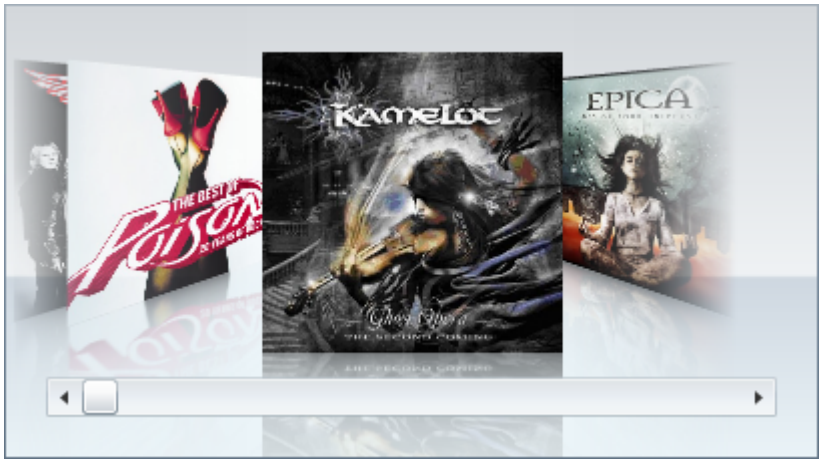
SelectedItemDistance プロパティは、選択されている項目とそのすぐ隣にある項目の間の距離を設定します。このプロパティの値は、現在の項目サイズに対する相対的な値です。つまり、値 0.5 の場合、選択されている項目とその隣の項目の間の距離は、選択されている項目のサイズの半分になります。

次に、**SelectedItemDistance** プロパティの設定例を示します。

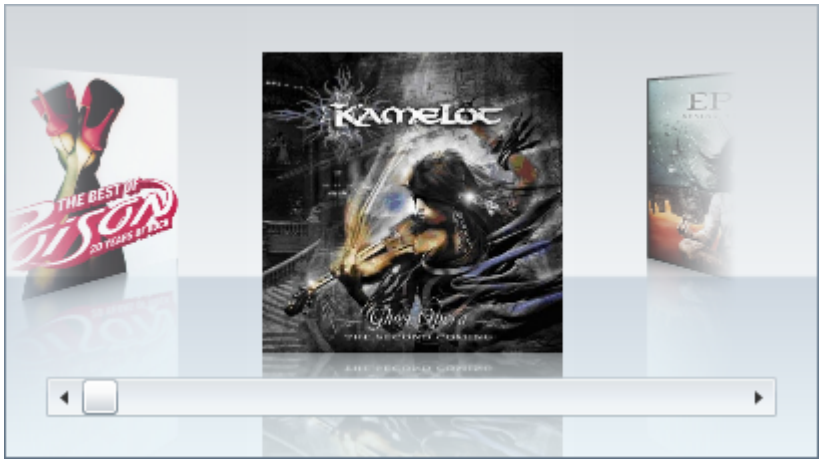
SelectedItemDistance = 0.5



SelectedItemDistance = 1



SelectedItemDistance = 1.5



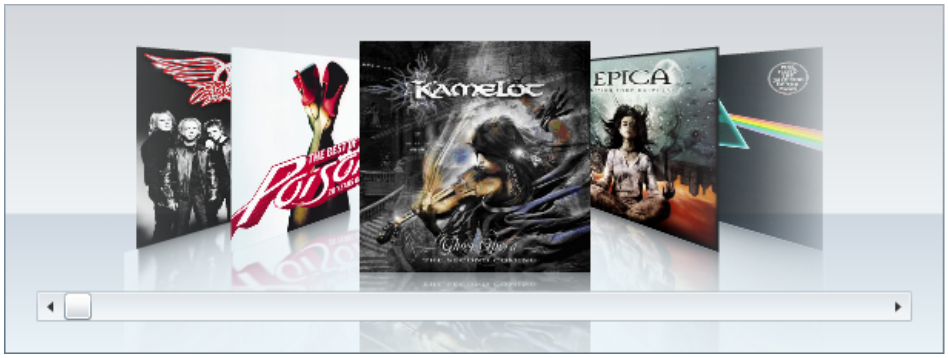
項目間距離

ItemDistance プロパティは、選択されていない項目間の距離を設定します。このプロパティは、現在の項目サイズに対する相対的な値です。したがって、この値が 0.5 の場合、項目間の距離は項目のサイズの半分になります。

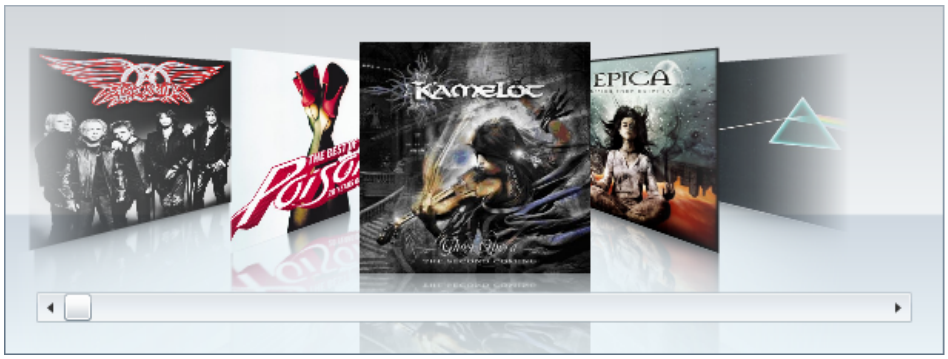
> **メモ**:`ItemDistance` プロパティでは、選択されている項目とその隣の項目の間の距離は変更されません。詳細については、「選択項目の距離」を参照してください。

次に、**ItemDistance** プロパティの設定例を示します。

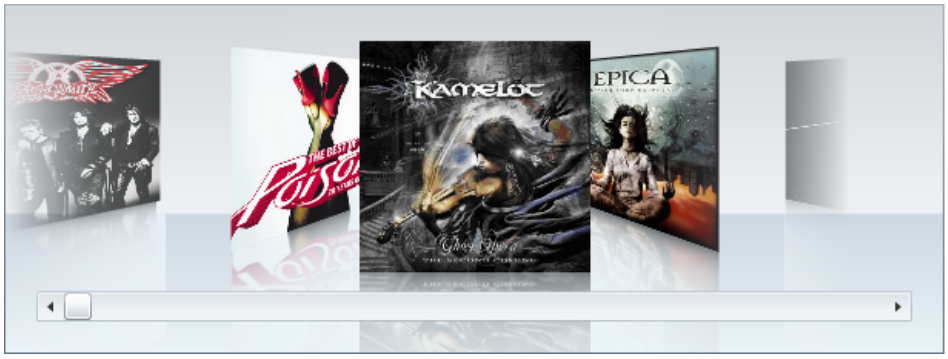
ItemDistance = 0.5



ItemDistance = 1



ItemDistance = 1.5



速度の設定

C1CoverFlow コントロールでは、3種類の速度、**ItemSpeed**、**ItemAngleSpeed**、および **EyeSpeed** を調整できます。次の表に、それぞれの速度プロパティの概要を示します。

プロパティ	説明
ItemSpeed プロパティ	ユーザーが項目を選択したりコントロールをスクロールしたときに、項目の位置が変化する速度を取得または設定します。この値は0より大きく、1以下にしてください。1を指定すると、直ちに変化します。
ItemAngleSpeed プロパティ	ユーザーが項目を選択したりコントロールをスクロールしたときに、項目の角度が変化する速度を取得または設定します。
EyeSpeed プロパティ	カメラの位置が変わる速度を取得または設定します。この値は0より大きく、1以下にしてください。1を指定すると、直ちに変化します。

XAML クイックリファレンス

このトピックでは、さまざまな **C1CoverFlow** タスクの実行に使用される XAML の概要を提供します。詳細については、「タスク別ヘルプ」セクションを参照してください。

C1CoverFlowItems から成る C1CoverFlow

次の XAML は、C1CoverFlow コントロールを作成し、その中に2つの **C1CoverFlowItem** をネストします。これらの **C1CoverFlowItem** の中には、さらに任意のコントロールをネストできます。

XAML

```
<c1:C1CoverFlow Height="278" Width="378">
    <c1:C1CoverFlowItem>
        <sdk:Calendar Height="172" Width="249" />
    </c1:C1CoverFlowItem>
    <Image Source="image1.jpg" />
    <c1:C1CoverFlowItem />
</c1:C1CoverFlow>
```

C1CoverFlow のテーマ

次の XAML は、**C1CoverFlow** コントロールに **ShinyBlue** テーマを適用します。

XAML

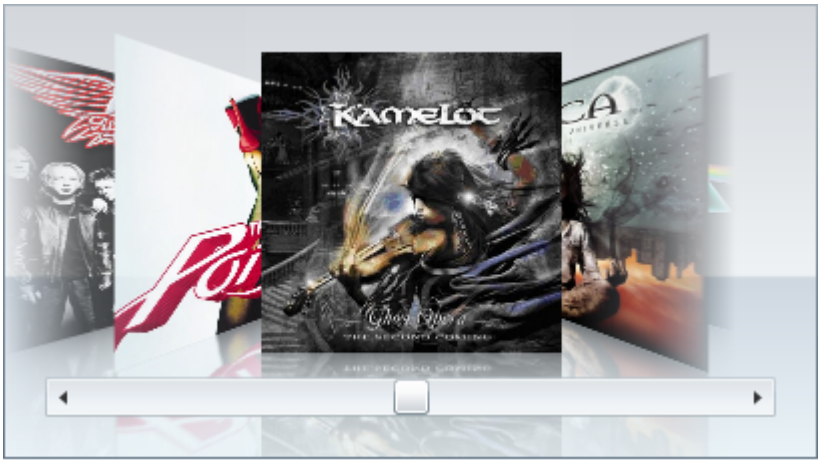
```
<c1:C1ThemeShinyBlue>
```

```
    <c1:C1CoverFlow Height="278" Width="378">
    </c1:C1CoverFlow>
</c1:C1ThemeShinyBlue>
```

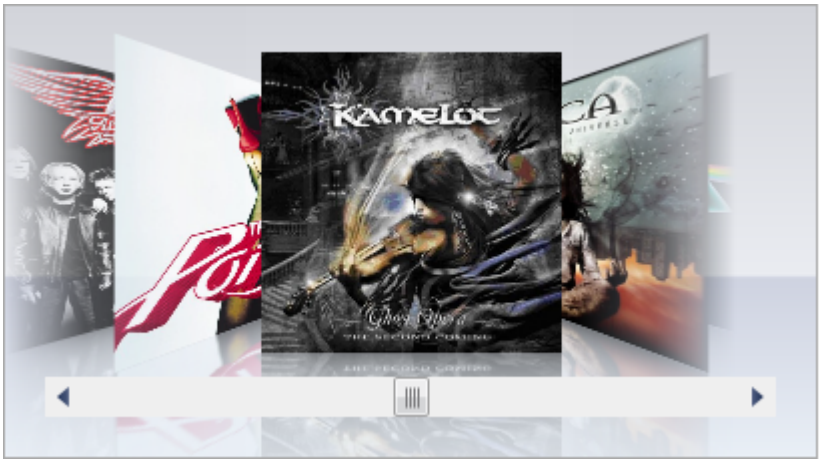
テーマ

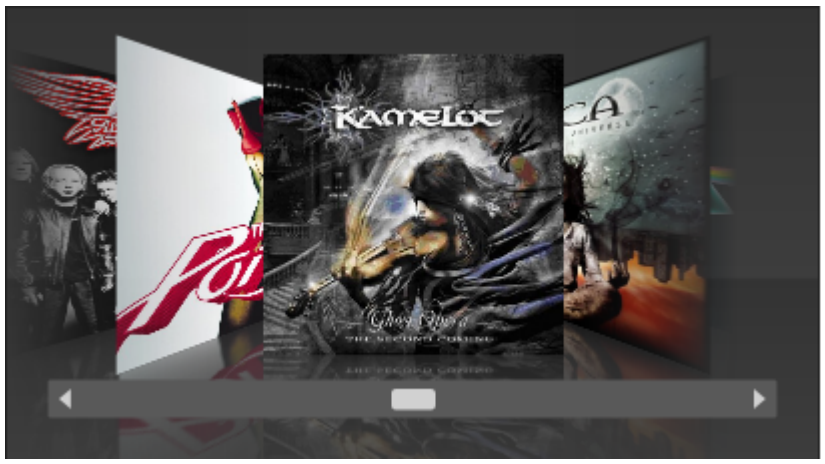
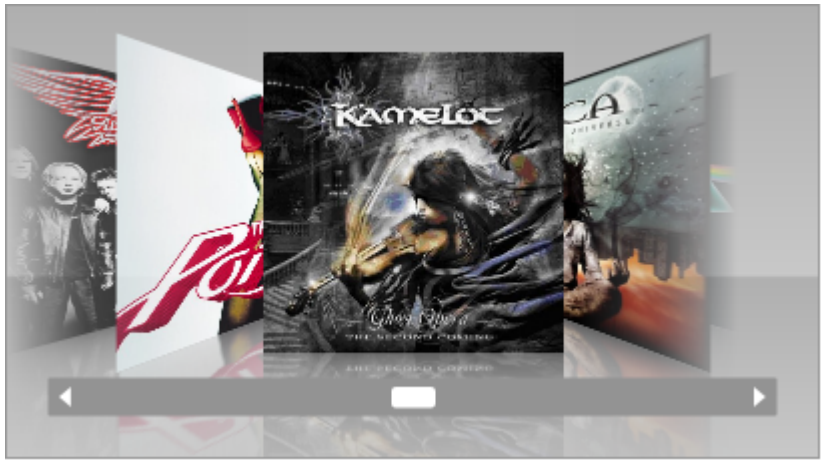
Silverlight のテーマは、いくつかのコントロールの外観を定義するイメージ設定のコレクションです。テーマはアプリケーション内の複数のコントロールに適用できるため、テーマを使用すると、スタイル設定作業を繰り返さなくても、一貫性のあるコントロールを作成できます。

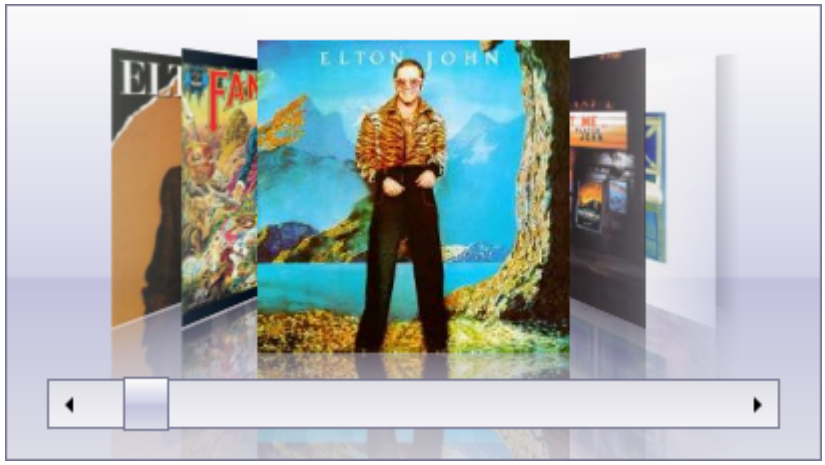
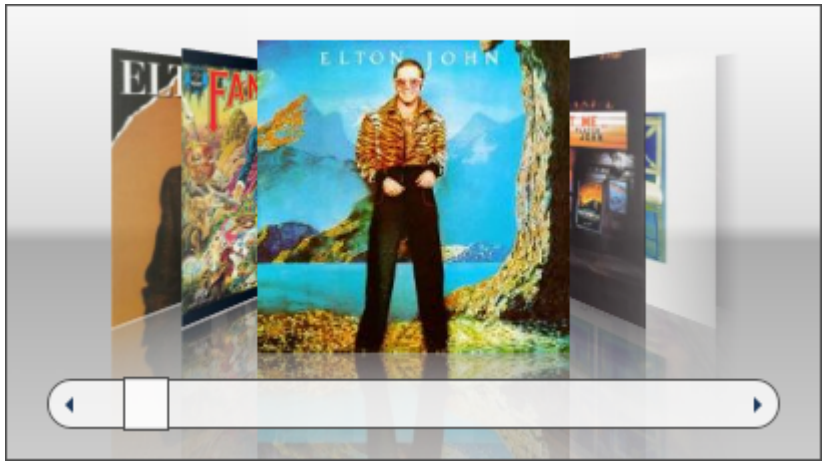
C1CoverFlow コントロールをプロジェクトに追加すると、コントロールはデフォルトの青色のテーマで表示されます。

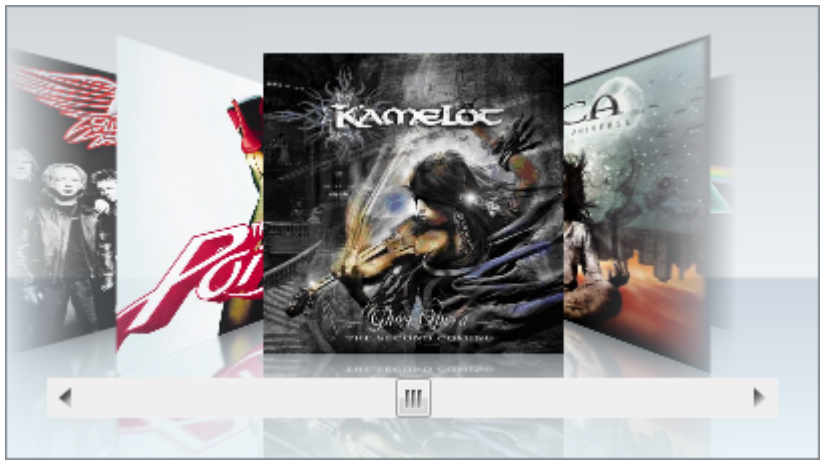


C1CoverFlow コントロールには、付属している Silverlight の複数のテーマ（BureauBlack、ExpressionDark、ExpressionLight、RainierOrange、ShinyBlue、WhistlerBlue）の中から1つテーマを設定することもできます。次の表に、各テーマのサンプルを示します。

完全なテーマ名	外観
C1ThemeBureauBlack	

C1ThemeCosmopolitan	
C1ThemeExpressionDark	
C1ThemeExpressionLight	
C1ThemeOffice2007Black	

C1ThemeOffice2007Blue	
C1ThemeOffice2007Silver	
C1ThemeOffice2010Black	
C1ThemeOffice2010Blue	

C1ThemeOffice2010Silver	
C1ThemeRainierOrange	
C1ThemeShinyBlue	
C1ThemeWhistlerBlue	

C1CoverFlow コントロールへのテーマの追加に関するタスク別ヘルプについては、「テーマを使用する」を参照してください。

ClearStyle

CoverFlow for Silverlight 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、グリッド全体のスタイルを簡単に設定できます。

次の表に、**C1CoverFlow** コントロールのブラシのプロパティの概要を示します。

ブラシ	説明
Background	コントロールの背景のブラシを取得または設定します。
ButtonForeground	ボタンの前景テキストのブラシを取得または設定します。
ButtonBackground	ボタンの背景テキストのブラシ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 System.Windows.Media.Brush を取得または設定します。
PressedBrush	クリックされたコントロール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。

いくつかのプロパティを設定することで、**C1CoverFlow** コントロールの外観を完全に変更できます。たとえば、**Background** は、CoverFlow のヘッダーの背景色を設定します。**Background** プロパティを "#FF490C0D" に設定すると、**C1CoverFlow** コントロールは次のようになります。

ComponentOne の ClearStyle 技術は、このように簡単に使用できます。

テンプレート

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」コントロールだからです。Silverlight アプリケーションで独自のユーザーインターフェイス（UI）、つまり「外観」を設計するのと同様に、**CoverFlow for Silverlight** によって管理されるデータにも独自の 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、C1CoverFlow 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。

C1CoverFlowItem テンプレートを編集する場合は、C1CoverFlowItem 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。

> **メモ**: メニューを使って新しいテンプレートを作成する場合は、テンプレートが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。

Template プロパティを使ってテンプレートをカスタマイズできます。

追加のテンプレート

デフォルトのテンプレートのほかに、**C1CoverFlow** コントロールと **C1CoverFlowItem** 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 C1CoverFlow コントロールまたは C1CoverFlowItem 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。

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 でのプログラミングに精通しており、**C1CoverFlow** コントロールの一般的な使用方法を理解していることを前提としています。**CoverFlow for 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、**CoverFlow for Silverlight** 製品を使って特定のタスクを行うための方法を提供します。

また、タスク別ヘルプトピックは、新しい 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。

リフレクションを操作する

以下のトピックでは、**C1CoverFlow** コントロールの項目の下に表示されるリフレクションを削除、変更、スタイル設定する方法について説明します。

リフレクションへぼかし効果を追加する

このトピックでは、Expression Blend を使用して、**C1CoverFlow** コントロール項目のリフレクションにぼかし効果を追加します。

次の手順に従います。

1. C1CoverFlow コントロールを右クリックし、[追加テンプレートの編集]→[生成されたアイテムコンテナーの編集(ItemContainerStyle)]→[コピーして編集]を選択します。**[Style リソースの作成]**ダイアログボックスが開きます。
2. [名前(キー)]フィールドに「AddBlurEffect」と入力します。
3. [OK]をクリックしてテンプレートを作成し、[Style リソースの作成]ダイアログボックスを閉じます。[オブジェクトとタイムライン]パネルで、子項目として Reflector コントロールを含む Template が表示されていることを確認します。
4. [オブジェクトとタイムライン]パネルで、Reflector を選択して[プロパティ]ウィンドウにプロパティを表示したら、次の手順に従います。
    a. **Miscellaneous** ノードを展開します。
    b. **[ReflectionEffects(Collection)]**の省略符ボタンをクリックします。[効果コレクションエディタ: ReflectionEffects]ダイアログボックスが開きます。

c. **[別のアイテムを追加]**をクリックして、**[オブジェクトの選択]**ダイアログボックスを開きます。

d. **BlurEffect** を選択し、**[OK]**をクリックしてぼかし効果を追加し、**[オブジェクトの選択]**ダイアログボックスを閉じます。

e. **BlurEffect** の Radius プロパティを "45" に設定します。

f. **[OK]**をクリックして、**[効果コレクションエディタ:ReflectionEffects]**ダイアログボックスを閉じます。

このトピックの作業結果

次の図は、強いぼかし効果を設定したときに **C1CoverFlow** コントロール項目のリフレクションがどのように表示されるかを示しています。

リフレクションを変更する

このトピックでは、Expression Blend を使用して、**C1CoverFlow** コントロール項目のリフレクションを変更します。

次の手順に従います。

1. C1CoverFlow コントロールを右クリックし、[追加テンプレートの編集]→[生成されたアイテムコンテナーの編集(ItemContainerStyle)]→[コピーして編集]を選択します。**[Style リソースの作成]**ダイアログボックスが開きます。
2. [名前(キー)]フィールドに「ModifyReflection」と入力します。
3. [OK]をクリックしてテンプレートを作成し、[Style リソースの作成]ダイアログボックスを閉じます。[オブジェクトとタイムライン]パネルで、子項目として Reflector コントロールを含む Template が表示されていることを確認します。
4. [オブジェクトとタイムライン]パネルで、Reflector を選択して[プロパティ]ウィンドウにプロパティを表示したら、次の手順に従います。

   a. **[その他]** ノードを展開します。

   b. **[ReflectionEffects(コレクション)]**の省略符ボタンをクリックします。**[Effect コレクションエディター: ReflectionEffects]**ダイアログボックスが開きます。

   c. **ReflectionOpacityEffect** のプロパティを任意の値に設定します。この例では、次のように設定します。

      - **Coefficient** プロパティを "5" に設定します。
      - **Offset** プロパティを "0" に設定します。

このトピックの作業結果

次の図は、**C1CoverFlow** コントロール項目の変更後のリフレクションを示しています。

リフレクション効果を削除する

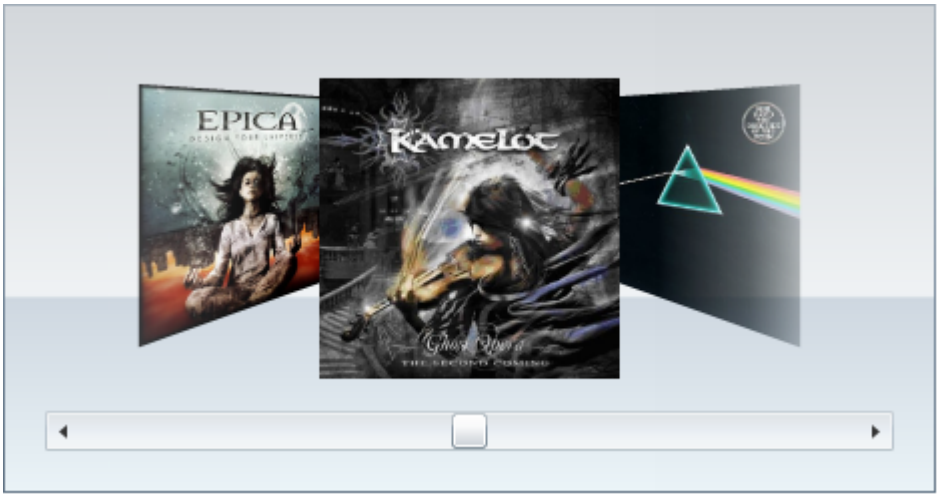
項目のリフレクションを削除するには、**ItemContainerStyle** テンプレートのコピーを作成し、**Reflector** コントロールを **ContentPresenter** コントロールに置き換えます。このトピックは、少なくとも1つの項目を含む **C1CoverFlow** コントロールがプロジェクトに追加されていることを前提としています。

次の手順に従います。

1. C1CoverFlow コントロールを右クリックし、[追加テンプレートの編集]→[生成されたアイテムコンテナーの編集（ItemContainerStyle）]→[コピーして編集]を選択します。**[Style リソースの作成]**ダイアログボックスが開きます。
2. [名前（キー）]フィールドに「Remove Reflection」と入力します。
3. **[OK]**をクリックしてテンプレートを作成し、[Style リソースの作成]ダイアログボックスを閉じます。[オブジェクトとタイムライン]パネルで、子項目として Reflector コントロールを含む Template が表示されていることを確認します。
4. [アセット]パネルを開き、検索ボックスに「ContentPresenter」と入力します。
5. [ContentPresenter]アイコンをダブルクリックして、Reflector コントロールを ContentPresenter コントロールに置き換えます。リフレクションが削除されていることを確認します。
6. ContentPresenter コントロールを選択して[プロパティ]ウィンドウに移動し、変換ノードを展開します。
7. Projection プロパティの[詳細オプション]ボタンをクリックし、[カスタム式]を選択します。
8. [カスタム式]テキストボックスに「{TemplateBinding ContentProjection}」と入力します。これにより、C1CoverFlow コントロール内の項目は、コントロールがデフォルトの Reflector コントロールを使用していたときと同じ方法で投影されます。

このトピックの作業結果

次の図は、リフレクションを削除した **C1CoverFlow** コントロールの項目がどのように表示されるかを示しています。



スクロールバーを操作する

以下のトピックでは、**C1CoverFlow** コントロールのスクロールバーの削除、サイズ変更、色の設定を行う方法について説明します。

スクロールバーを削除する

このトピックでは、Expression Blend を使って **C1CoverFlow** コントロールのスクロールバーを削除します。

次の手順に従います。

1. C1CoverFlow コントロールを右クリックし、[テンプレートの編集]→[コピーして編集]を選択します。**[Style リソースの作成]**ダイアログボックスが開きます。
2. [名前(キー)]フィールドに「RemoveScrollbar」と入力します。
3. [OK]をクリックしてテンプレートを作成し、[Style リソースの作成]ダイアログボックスを閉じます。
4. [オブジェクトとタイムライン]パネルで、[Grid]ノードを展開します。
5. Scroll を選択し、[Delete]キーを押します。C1CoverFlow コントロールのスクロールバーが削除されます。

このトピックの作業結果

次の図は、スクロールバーを削除した **C1CoverFlow** コントロールを示しています。

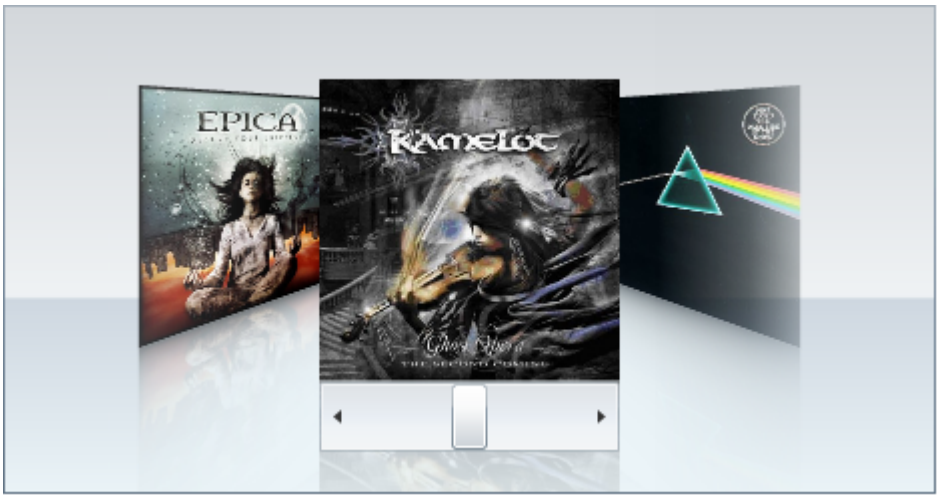
スクロールバーのサイズを変更する

このトピックでは、Expression Blend を使って **C1CoverFlow** コントロールのスクロールバーのサイズを変更します。

1. C1CoverFlow コントロールを右クリックし、[テンプレートの編集]→[コピーして編集]を選択します。**[Style リソースの作成]**ダイアログボックスが開きます。
2. [名前(キー)]フィールドに「ResizeScrollbar」と入力します。
3. [OK]をクリックしてテンプレートを作成し、[Style リソースの作成]ダイアログボックスを閉じます。
4. [オブジェクトとタイムライン]パネルで、[Grid]ノードを展開します。
5. Scroll を選択して[プロパティ]ウィンドウにプロパティリストを表示してから、次のプロパティを設定します。
    - Width プロパティを "150" に設定します。
    - Height プロパティを "35" に設定します。

このトピックの作業結果

次の図は、幅を 150 ピクセル、高さを 35 ピクセルに設定したスクロールバーを含む **C1CoverFlow** コントロールを示しています。



画像を追加する

このトピックでは、Blend、XAML、およびコードで、**C1CoverFlow** コントロールに画像を追加する方法について説明します。

Blend の場合

次の手順に従います。

1. プロジェクトに C1CoverFlow コントロールを追加します。
2. [オブジェクトとタイムライン]パネルで[C1CoverFlow]を選択します。
3. [アセット]パネルで、検索フィールドに「Image」と入力します。>[Image]アイコンをダブルクリックして、Image コントロールを C1CoverFlow コントロールに追加します。
4. [オブジェクトとタイムライン]パネルで[Image]を選択します。
5. [プロパティ]ウィンドウで、Source の省略符ボタンをクリックします。**[既存のアイテムを追加]**ダイアログボックスが開きます。

6. 画像が保存されている場所に移動し、画像ファイルを選択し、**[開く]**をクリックして画像を **Image** コントロールに追加します。

XAML の場合

次の手順に従います。

1. **C1CoverFlow** コントロールの終了タグを追加して、XAML を次のようにします。

```
XAML
<c1:C1CoverFlow Margin="0,0,205,200"></c1:C1CoverFlow>
```

2. 次の XAML を <lc1:C1CoverFlow> タグと </c1:C1CoverFlow> タグの間に配置し、追加する画像ファイルの名前で「YourImage.png」を置き換えます。

```
XAML
<Image Height="100" Width="100" Source="YourImage.png"/>
```

コードの場合

次の手順に従います。

1. XAML ビューで、x:Name="C1CoverFlow1" を タグに追加します。これで、このコントロールをコードから呼び出すための一意の識別子が指定されます。
2. MainPage.xaml コードページ（MainPage.xaml.cs または MainPage.xaml.vb）を開きます。このページの拡張子は、プロジェクトに選択した言語によって異なります。
3. 次の名前空間をインポートします。

VisualBasic

```
Imports System.Windows.Media.Imaging
```

C#

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

4. InitializeComponent メソッドの下に次のコードを追加します。

VisualBasic

```
' Image コントロールを作成します
Dim Image1 As New Image()
' ビットマップ画像を作成し、そのソースとして画像を追加します
Dim BitMapImage1 As New BitmapImage()
BitMapImage1.UriSource = New Uri("Epica.jpg", UriKind.RelativeOrAbsolute)
' ビットマップ画像を Image コントロールのソースとして追加します
Image1.Source = BitMapImage1
'C1CoverFlow コントロールに Image コントロールを追加します
```

```
C1CoverFlow1.Items.Add(Image1)
```

C#

```
// Image コントロールを作成します
Image Image1 = new Image();
// ビットマップ画像を作成し、そのソースとして画像を追加します
BitmapImage BitMapImage1 = new BitmapImage();
BitMapImage1.UriSource = new Uri("Epica.jpg", UriKind.RelativeOrAbsolute);
// ビットマップ画像を Image コントロールのソースとして追加します
Image1.Source = BitMapImage1;
//C1CoverFlow コントロールに Image コントロールを追加します
C1CoverFlow1.Items.Add(Image1);
```

5. プログラムを実行します。

コレクションへ連結する

C1CoverFlow コントロールを項目のコレクションに連結できます。このセクションでは、コントロールを String 型の **ObservableCollection** に連結します。連結先のコレクションは、データコンテキストとして渡されます。このトピックは、Microsoft Expression Blend で作業していること、およびプロジェクトに C1CoverFlow コントロールが追加してあることを前提としています。

次の手順に従います。

1. C1CoverFlow コントロールを選択し、次のプロパティを設定します。
   - **Width** プロパティを "400" に設定します。
   - **Height** プロパティを "200" に設定します。
   - **SelectedIndex** プロパティを "2" に設定します。
2. 次の手順に従って、プロジェクトに画像を追加します。
   a. **[プロジェクト]**パネルでプロジェクトを右クリックしてコンテキストメニューを開き、**[新しいフォルダーの追加]**を選択します。新しいフォルダには「Images」と名前を付けます。
   b. **iImages** フォルダを右クリックし、**[既存のアイテムの追加]**を選択します。**[既存のアイテムの追加]**ダイアログボックスが開きます。
   c. 次の場所に移動します。
   Documents\ComponentOne Samples\Silverlight\QuickStart\QuickStart
   d. 製品サンプル「QuickStart」に含まれる **cover1.jpg、cover2.jpg、cover3.jpg、cover4.jpg、cover5.jpg** を選択します。
   e. **[開く]**をクリックして**[既存のアイテムの追加]**ダイアログボックスを閉じ、画像をフォルダに追加します。画像が **Images** フォルダに追加されます。
3. MainPage.xaml のコードページ（MainPage.xaml.cs または MainPage.xaml.vb）を開きます。
4. Initialize() メソッドの下に次のコードを追加します。

VisualBasic

```
' ObservableCollection を作成します
Dim bluRayCovers As New System.Collections.ObjectModel.ObservableCollection(Of
String)()
bluRayCovers.Add("Images/cover1.jpg")
bluRayCovers.Add("Images/cover2.jpg")
bluRayCovers.Add("Images/cover3.jpg")
bluRayCovers.Add("Images/cover4.jpg")
bluRayCovers.Add("Images/cover5.jpg")
' コレクションをデータコンテキストとしてコントロールに渡します
Me.DataContext = bluRayCovers
```

C#

```
// ObservableCollection を作成します
System.Collections.ObjectModel.ObservableCollectionbluRayCovers = new
System.Collections.ObjectModel.ObservableCollection();
bluRayCovers.Add("Images/cover1.jpg");
bluRayCovers.Add("Images/cover2.jpg");
bluRayCovers.Add("Images/cover3.jpg");
bluRayCovers.Add("Images/cover4.jpg");
bluRayCovers.Add("Images/cover5.jpg");
//コレクションをデータコンテキストとしてコントロールに渡します
this.DataContext = bluRayCovers;
```

5. 次の手順に従って、コントロールにコレクションを連結します。

   a. デザインビューに戻ります。

   b. C1CoverFlow コントロールを選択して、**[プロパティ]**ウィンドウにプロパティのリストを表示します。

   c. **ItemsSource** プロパティの横の**[詳細オプション]**ボタンをクリックし、**[カスタム式]**を選択します。

   d. **[カスタム式]**フィールドを "{Binding}" に設定します。これで、後の手順で作成するテンプレートに **DataContext** を直接渡すように **ItemsSource** が設定されます。

6. 次の手順に従って、Image コントロールを含む DataTemplate を作成してから、Image コントロールの Source プロパティをコレクションに連結します。

   a. C1CoverFlow コントロールを右クリックし、**[追加テンプレートの編集]**→**[生成されたアイテムの編集(ItemTemplate)]**→**[空アイテム 作成]**を選択します。**[DataTemplate リソースの作成]**ダイアログボックスが開きます。

   b. **[名前(キー)]**フィールドに「ImageTemplate」と入力します。

   c. **[OK]**をクリックして**[DataTemplate リソースの作成]**ダイアログボックスを閉じ、テンプレートを作成します。

   d. **[アセット]**パネルで検索バーに**「Image」**と入力します。次に、**[Image]**アイコンをダブルクリックして Image コントロールをテンプレートに追加します。

   e. **Image** コントロールを選択して**[プロパティ]**ウィンドウにプロパティリストを表示してから、次の手順に従います。

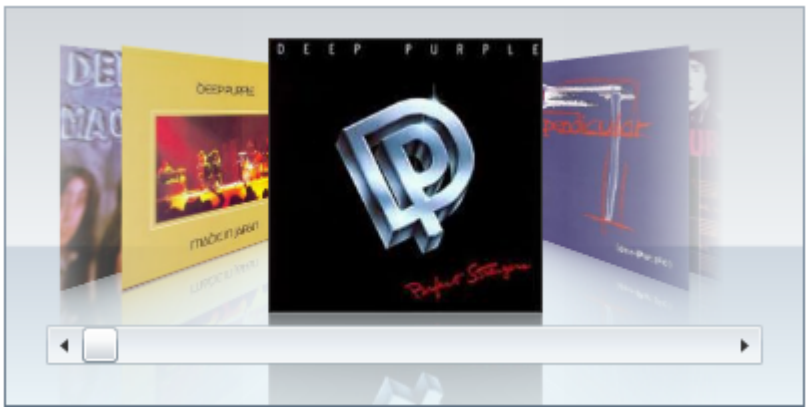
      - **Width** プロパティのグリフ をクリックし、**Width** プロパティを "**Auto**" に設定します。

- **Height** プロパティのグリフ をクリックし、**Height** プロパティを "**Auto**" に設定します。
- **Source** プロパティの **[詳細オプション]**ボタンをクリックして**[カスタム式]**を選択し、**[カスタム式]**フィールドを "{Binding}" に設定します

7. プロジェクトを実行し、コレクションで指定した画像が **C1CoverFlow** コントロールの項目として表示されることを確認します。

このトピックの作業結果

このトピックの結果は、次の図のようになります。



両側の項目の角度を変更する

C1CoverFlow コントロールの選択された項目の両側に表示される項目の角度を変更するには、**ItemAngle** プロパティに数値を設定します（詳細については、「項目角度」を参照）。このトピックでは、**ItemAngle** プロパティを "15" に設定して、サイド項目が 15 度の角度で回転して表示されるようにします。

Blend の場合

次の手順に従います。

1. **C1CoverFlow** コントロールを選択します。
2. ［プロパティ］ウィンドウで、**ItemAngle** プロパティを "15" に設定します。

XAML の場合

ItemAngle="1" を <c1:C1CoverFlow> タグに追加します。マークアップは次のようになります。

XAML

```
<c1:C1CoverFlow Margin="0,0,87,233" ItemAngle=”15”>
```

コードの場合

次の手順に従います。

1. XAML ビューで、x:Name="C1CoverFlow" を タグに追加します。これで、このコントロールをコードから呼び出すための一意の識別子が指定されます。
2. InitializeComponent メソッドの下に次のコードを追加します。

VisualBasic

```
C1CoverFlow1.ItemAngle = 15
```

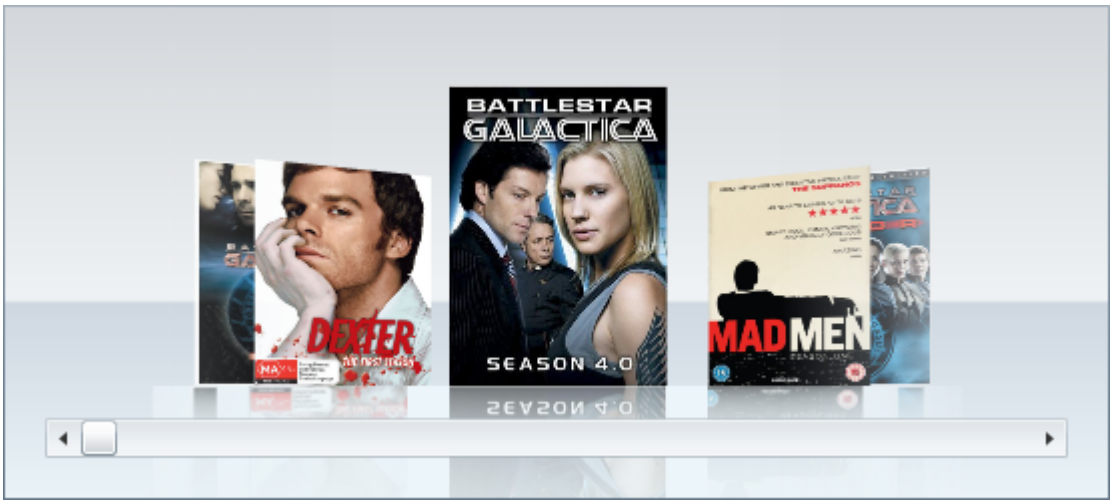
C#

```
C1CoverFlow1.ItemAngle = 15;
```

3. プログラムを実行します。

このトピックの作業結果

次の図は、ItemAngle プロパティを "**15**" に設定した C1CoverFlow コントロールを示しています。



カメラの垂直位置を変更する

カメラの垂直位置を変更するには、**EyeHeight** プロパティを設定します（詳細については、「視点高さ」を参照）。このトピックでは、EyeHeight プロパティを "1" に設定して、カメラが **C1CoverFlow** の項目を下から表示するようにします。

Blend の場合

次の手順に従います。

1. **C1CoverFlow** コントロールを選択します。
2. [プロパティ]ウィンドウで、**EyeHeight** プロパティを "1" に設定します。

XAML の場合

EyeHeight="1" を <c1:C1CoverFlow> タグに追加します。マークアップは次のようになります。

XAML

```
<c1:C1CoverFlow Margin="0,0,87,233" EyeHeight="1">
```

コードの場合

次の手順に従います。

1. XAML ビューで、x:Name="C1CoverFlow1" を <c1:C1CoverFlow> タグに追加します。これで、このコントロールをコー

ドから呼び出すための一意の識別子が指定されます。

2. InitializeComponent メソッドの下に次のコードを追加します。

VisualBasic

```
C1CoverFlow1.EyeHeight = 1
```

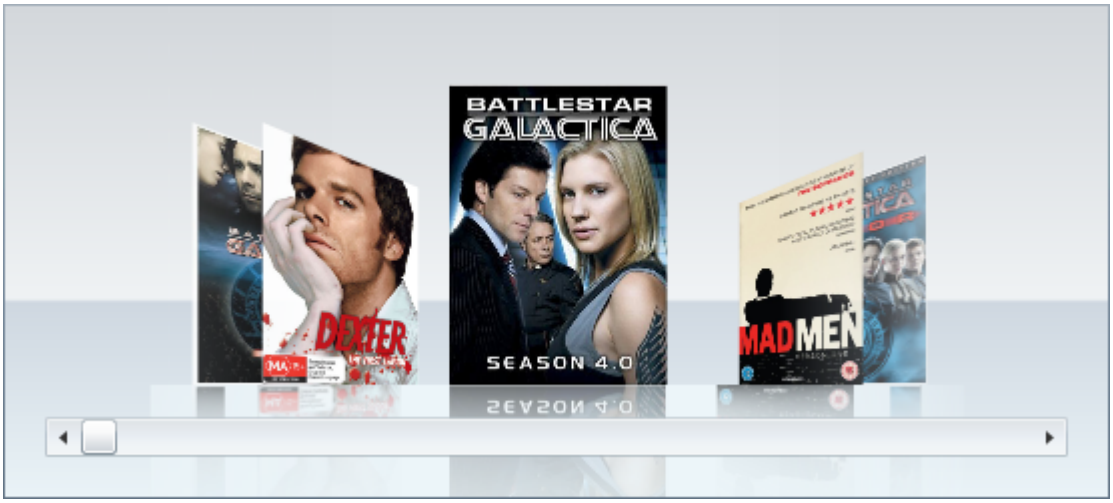
C#

```
C1CoverFlow1.EyeHeight = 1;
```

3. プログラムを実行します。

このトピックの作業結果

次の図は、EyeHeight プロパティを "**1**" に設定した C1CoverFlow コントロールを示しています。



選択項目と両側の項目の間の距離を設定する

SelectedItemDistance プロパティは、コントロールの中央に表示される選択されている項目と、その両側に表示される項目との間の距離を設定します（詳細については、「**選択項目の距離**」を参照）。このトピックでは、**SelectedItemDistance** プロパティを "0.4" に設定して、選択項目とサイド項目の間の距離が選択項目のサイズの 10 分の4になるようにします。

Blend の場合

次の手順に従います。

1. **C1CoverFlow** コントロールを選択します。
2. [プロパティ]ウィンドウで、**SelectedItemDistance** プロパティを "0.4" に設定します。

XAML の場合

SelectedItemDistance="0.4" を <c1:C1CoverFlow&> タグに追加します。マークアップは次のようになります。

```
XAML
```

```
<c1:C1CoverFlow Margin="0,0,87,233" SelectedItemDistance=”0.4”>
```

コードの場合

次の手順に従います。

1. XAML ビューで、x:Name="C1CoverFlow" を タグに追加します。これで、このコントロールをコードから呼び出すための一意の識別子が指定されます。
2. InitializeComponent メソッドの下に次のコードを追加します。

VisualBasic

```
C1CoverFlow1.SelectedItemDistance = 0.4
```

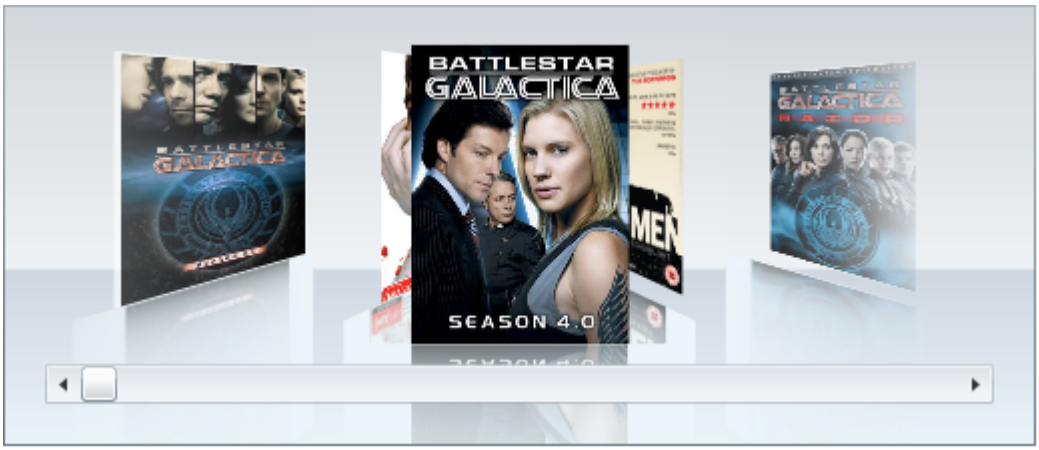
C#

```
C1CoverFlow1.SelectedItemDistance = 0.4;
```

3. プログラムを実行します。

このトピックの作業結果

次の図は、SelectedItemDistance プロパティを "0.4" に設定した C1CoverFlow コントロールを示しています。



テーマを使用する

C1CoverFlow コントロールには、デフォルトで明るい青色のテーマが設定されていますが、全部で6つのテーマ(「テーマ」を参照)を適用できます。このトピックでは、C1CoverFlow コントロールのテーマを **C1ThemeRanierOrange** に変更します。

Blend の場合

次の手順に従います。

1. [アセット]パネルをクリックします。
2. 検索バーに、「C1ThemeRainierOrange」と入力します。[C1ThemeRainierOrange]アイコンが表示されます。
3. **[C1ThemeRainierOrange]**アイコンをダブルクリックしてプロジェクトに追加します。

4. 検索バーに「C1CoverFlow」と入力して C1CoverFlow コントロールを検索します。
5. C1CoverFlowアイコンをダブルクリックして C1CoverFlow コントロールをプロジェクトに追加します。
6. [オブジェクトとタイムライン]タブで[C1CoverFlow]を選択し、これをドラッグアンドドロップ操作で[C1ThemeRainierOrange]の下に配置します。
7. プロジェクトを実行します。

Visual Studio の場合

次の手順に従います。

1. Visual Studio で、.xaml ページを開きます。
2. タグの間にカーソルを置きます。
3. [ツール]パネルで、[C1ThemeRainierOrange]アイコンをダブルクリックしてテーマを宣言します。このタグは次のようになります。 <my:C1ThemeRainierOrange<
4. <my:C1ThemeRainierOrange> タグと </my:C1ThemeRainierOrange> タグの間にカーソルを置きます。
5. [ツール]パネルで、C1CoverFlowアイコンをダブルクリックして、コントロールをプロジェクトに追加します。そのタグはタグの子として表示され、マークアップは次のようになります。

```
XAML
<my:C1ThemeRainierOrange>
    <c1:C1CoverFlow x:Name="C1CoverFlow1"></c1:C1CoverFlow>
</my:C1ThemeRainierOrange>
```

6. プロジェクトを実行します。

このトピックの作業結果

次の図は、**C1ThemeRainierOrange** テーマが使用された **C1CoverFlow** コントロールです。

Expander

Expander for WPF/Sliverlight を使用すると、貴重な画面領域を節約できます。**Expander for WPF/Silverlight** には **C1Expander** というコントロールがあり、これにより、テキスト、画像、およびコントロールを含めることができる展開/折りたたみ可能な情報パネルを作成できます。4つの展開方向から選択し、Microsoft Expression Blend で外観をカスタマイズして、コントロールのスタイルを全面的に制御します。

クイックスタート

このクイックスタートは、**Expander 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、最初に 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 **C1Expander** コントロールを追加し、**C1Expander** コントロールのコンテンツ領域にコンテンツを追加します。

手順 1:アプリケーションの作成

この手順では、最初に Visual Studio で **Expander for WPF/Silverlight** を使用する WPF/Silverlight アプリケーションを作成します。

次の手順に従います。

1. Visual Studio で、**[ファイル]**→**[新しいプロジェクト]**を選択します。
2. **[新しいプロジェクト]**ダイアログボックスの左ペインから言語を選択し、テンプレートリストから**[WPF/Silverlight アプリケーション]**を選択します。プロジェクトの**名前**を入力し、**[OK]**をクリックします。**[新しい WPF/Silverlight アプリケーション]**ダイアログボックスが表示されます。
3. **[OK]**をクリックすると、**[新しい WPF/Silverlight アプリケーション]**ダイアログボックスが閉じ、プロジェクトが作成されます。
4. プロジェクトの XAML ウィンドウで、**<UserControl>** タグの **DesignWidth**="**400**" **DesignHeight**="**300**" を **DesignWidth**="**Auto**" **DesignHeight**="**Auto**" に変更して、UserControl をサイズ変更します。次のようになります。

   XAML

   ```
   <UserControl x:Class="SilverlightApplication1.MainPage"
      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"
       mc:Ignorable="d" d:DesignWidth="Auto" d:DesignHeight="Auto">
   ```

   これで、**UserControl** は、中に置かれた内容に合わせてサイズ変更されるようになります。
5. プロジェクトの XAML ウィンドウで、カーソルを <Grid> タグと </Grid> タグの間に置き、1回クリックします。
6. **ツール**パネルに移動し、**C1Expander** アイコンをダブルクリックして、コントロールをグリッドに追加します。XAML マークアップは次のようになります。

   XAML

   ```
   <Grid x:Name="LayoutRoot">
   <c1:C1Expander></c1:C1Expander>
   </Grid>
   </UserControl>
   ```

これで、**C1Expander** コントロールを含む WPF/Silverlight アプリケーションを作成できました。次の手順では、**C1Expander** コントロールの外観と動作をカスタマイズします。

手順 2:コントロールの設定

前の手順では、WPF/Silverlight プロジェクトを作成し、**C1Expander** コントロールを追加しました。この手順では、C1Expander コントロールの動作と外観をカスタマイズします。

次の手順に従います。

1. <c1:C1Expander> タグに Height="75" を追加して、コントロールの高さを設定します。XAML マークアップは次のようになります。
   <c1:C1Expander **Height="75">**
2. <c1:C1Expander> タグに Width="140" を追加して、コントロールの幅を設定します。XAML マークアップは次のようになります。
   <c1:C1Expander **Height="75" Width="140">**
3. <c1:C1Expander> タグに Header="Expander クイックスタート" を追加して、コントロールのヘッダーテキストを設定します。XAML マークアップは次のようになります。
   <c1:C1Expander Height="75" Width="140" **Header="Expander クイックスタート">**
4. <c1:C1Expander> タグに Background="Aqua" を追加して、コンテンツ領域の背景色を設定します。XAML マークアップは次のようになります。
   <c1:C1Expander Height="75" Width="140" Header="Expander クイックスタート" **Background="Aqua">**
5. <c1:C1Expander> タグに ExpandDirection="Up" を追加します。これにより、C1Expander コントロールが上からではなく下から展開されます。これはデフォルトです。XAML マークアップは次のようになります。
   <c1:C1Expander Height="75" Width="140" Header="Expander クイックスタート" Background="Aqua" **ExpandDirection="Up">**

この手順では、**C1Expander** コントロールの外観と動作をカスタマイズしました。次の手順では、コントロールにコンテンツを追加します。

手順 3:コンテンツの追加

前の手順では、**C1Expander** コントロールの外観と動作をカスタマイズしました。この手順では、**C1Expander** コントロールのコンテンツ領域にコントロールを追加してからプロジェクトを実行して、このクイックスタートで作成したアプリケーションの実行時機能を確認します。

次の手順に従います。

1. カーソルを <c1:C1Expander> タグと </c1:C1Expander> タグの間に置き、[Enter]キーを押します。
2. **ツール**ボックスに移動し、**[StackPanel]**アイコンをダブルクリックして、**C1Expander** コントロールに **StackPanel** コントロールを追加します。XAML マークアップは次のようになります。

XAML

```
<c1:C1Expander Height="200" Width="250" Background="Aqua" Header="Expander クイックスタート"
 ExpandDirection="Up">
<StackPanel></StackPanel>
</c1:C1Expander>
```

このクイックスタートのコンテンツ領域には複数のコントロールを追加するので、**C1Expander** コントロールのコンテンツ領域に **StackPanel** コントロールを追加しています。**C1Expander** コントロールはコンテンツコントロールであり、一度に1つの子項目しか受け取ることができません。複数の子項目を受け取ることができるパネルベースのコントロールを **C1Expander** コントロールに追加すると、この制限を回避できます。

3. パネルに追加されたすべてのコンテンツが中央に配置されるように、<StackPanel> タグに HorizontalAlignment="Center" を追加します。XAML マークアップは次のようになります。
<StackPanel HorizontalAlignment="Center">

4. カーソルを <StackPanel> タグと </StackPanel> タグの間に置き、[Enter]キーを押します。

5. **ツール**パネルに移動し、**[TextBlock]**アイコンをダブルクリックして、**StackPanel** コントロールにコントロールを追加します。**StackPanel** のコンテンツに合計3つの **TextBlock** コントロールが追加されるように、この手順を2回繰り返します。XAML マークアップは次のようになります。

XAML

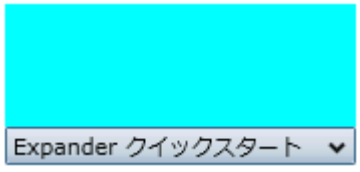
```
<StackPanel HorizontalAlignment="Center" >
    <TextBlock></TextBlock>
    <TextBlock></TextBlock>
    <TextBlock></TextBlock>
</StackPanel>
```

6. 最初の <TextBlock> タグに Text="コントロール 1" を追加し、2番目の <TextBlock> タグに Text="コントロール 2" を追加し、3番目の <TextBlock> タグに Text="コントロール 3" を追加します。XAML マークアップは次のようになります。

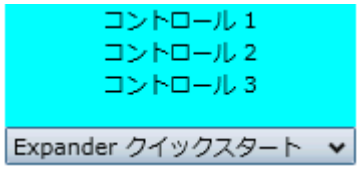
XAML

```
<TextBlock Text="コントロール 1"></TextBlock>
    <TextBlock Text="コントロール 2"></TextBlock>
    <TextBlock Text="コントロール 3"></TextBlock>
```

7. **[デバッグ]**メニューから**[デバッグ開始]**を選択し、実行時にアプリケーションがどのように表示されるかを確認します。エキスパンダのコンテンツが表示されないことを確認します。



C1Expander コントロールのヘッダーをクリックしてコンテンツを展開します。コンテンツ領域に追加した3つの TextBox コントロールが表示されることを確認します。



おめでとうございます!

これで、**Expander for WPF/Silverlight** クイックスタートは終了です。このクイックスタートでは、**Expander for WPF/Silverlight** のアプリケーションを作成してカスタマイズし、コントロールのコンテンツ領域にコンテンツを追加し、コントロールの実行時機能を確認しました。

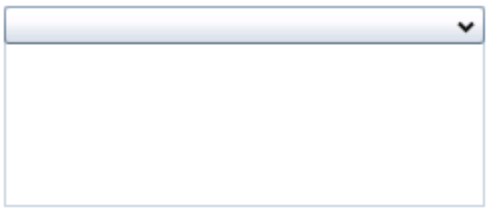
主な特長

Expander for WPF/Silverlight を使用して、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。以下の主な特長をうまく利用して、Expander for WPF/Silverlight を最大限に活用してください。

- **ページのロード時の展開**
  IsExpanded プロパティを使用すると、ページのロード時に **C1Expander** コントロールを展開するかどうかを選択できます。IsExpanded プロパティはデフォルトで **True** に設定され、コントロールは初期状態で展開されて表示されます。詳細については、「展開可能性」を参照してください。
- **展開方向**
  C1Expander コントロールは、4方向に展開できます。**ExpandDirection** プロパティでコントロールの展開方向を指定します。**Top**、**Right**、**Bottom**、または Left を設定できます。詳細については、「展開方向」を参照してください。
- **カスタムヘッダー**
  C1Expander コントロールのヘッダーは、テキストとコントロールの両方を使ってカスタマイズできます。カスタマイズ可能なヘッダー要素の詳細については、「エキスパンダのヘッダー」を参照してください。
- **項目の整理方法の設定**
  Expander では、スペースを最大限に活用できます。**C1Expander** のサイズと位置を設定し、必要になるまで項目を非表示にできます。

Expander の使い方

C1Expander コントロールはヘッダーコンテンツコントロールの1つで、テキスト、画像、およびコントロールを格納する展開/折りたたみ可能なペインを提供します。プロジェクトに追加された **C1Expander** コントロールは、空白のコンテンツ領域を含むヘッダーとして表示されます。デフォルトでは、コントロールのインターフェイスは次の図のように表示されます。



プロジェクトに **C1Expander** コントロールを追加した後は、そのコントロールにヘッダーとコンテンツを追加できます。また、コントロールの展開可能性や方向などの動作を変更することもできます。以下のトピックでは、**C1Expander** コントロールの要素および機能について説明します。

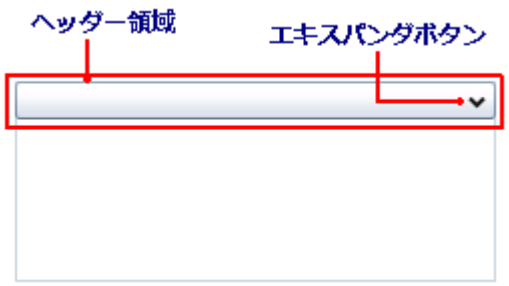
エキスパンダの要素

このセクションでは、**C1Expander** コントロールを構成する要素について画像を使用してわかりやすく説明します。このコントロールは、ヘッダーバーとコンテンツパネルの2つの要素で構成されます。この2つの要素が組み合わされて、完全な **C1Expander** コントロールになります。

エキスパンダのヘッダー

デフォルトでは、**C1Expander** コントロールのヘッダー要素はコントロールの上部に表示され、ヘッダーの右側にエキスパンダボタンが表示されます。C1Expander コントロールをページ内に配置したとき、最初は、このヘッダー要素にテキストは含まれません。

次の図に、**C1Expander** コントロールのヘッダー領域を示します。



ヘッダー要素にテキストを追加するには、**Header** プロパティに文字列を設定します。テキストを追加したら、いくつかのフォントプロパティ(「テキストのプロパティ」を参照)を使用して、テキストのスタイルを指定できます。このヘッダーに Silverlight コントロールを追加することもできます。ヘッダーへのコンテンツの追加に関するタスク別ヘルプについては、「ヘッダーへコンテンツを追加する」を参照してください。

ヘッダー要素とエキスパンダボタンの位置は、コントロールの展開方向によって異なります。展開方向の詳細については、「展開方向」トピックを参照してください。

属性構文とプロパティ要素構文

C1Expander ヘッダーに単純な要素(書式設定されていない文字列など)を追加する場合は、次に示すように、XAML マークアップの一般的な XML 属性を使用できます。

XAML

```
<c1:C1Expander Header="Hello World"/>
```

ただし、コンテンツ領域に、グリッドやパネルなどの複雑な要素を追加することもできます。このような場合は、次に示すように、プロパティ要素構文を使用できます。

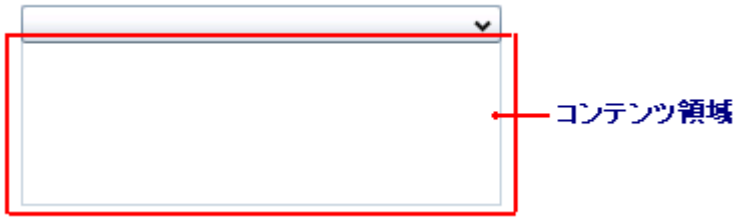
XAML

```
<c1:C1Expander ExpandDirection="Down" Width="150" Height="55" Name="C1Expander1">
    <c1:C1Expander.Header>
        <Grid HorizontalAlignment="Stretch">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
            <TextBlock Text="C1Expander Header" />
        </Grid>
    </c1:C1Expander.Header>
</c1:C1Expander>
```

エキスパンダのコンテンツ領域

最初、**C1Expander** コントロールのコンテンツ領域は空です。コンテンツ領域には、グリッド、テキスト、画像、および任意のコントロールを追加できます。Blend の場合、簡単なドラッグアンドドロップ操作を使用して、コントロールのコンテンツ領域に要素を追加したり、コントロール内の要素を移動することができます。

次の図に、C1Expander コントロールのコンテンツ領域を示します。



コンテンツ領域にテキストを追加するには、**C1Expander** コントロールの **Content** プロパティを設定するか、**TextBox** 要素をコンテンツ領域に追加します。実行時にコンテンツ領域に Silverlight 要素を追加することは簡単です。要素を追加するには、簡単なドラッグアンドドロップ操作または XAML のいずれかを使用できます。実行時にコントロールを追加する場合は、C# または Visual Basic コードを使用できます。

C1Expander などのコンテンツコントロールは、子要素を一度に1つだけ受け入れることができます。ただし、この問題は、C1Expander コントロールの子要素としてパネルベースのコントロールを追加することによって回避できます。**StackPanel** コントロールなどのパネルベースのコントロールは、複数の要素を保持できます。パネルベースのコントロール自身が複数の要素を保持できるため、これを使用すると、C1Expander コントロールはコントロールを1つだけ保持できるという制限を満たしつつ、コンテンツ領域に複数のコントロールを表示できます。

コンテンツ領域へのコンテンツの追加に関するタスク別ヘルプについては、「コンテンツ領域へコンテンツを追加する」を参照してください。

属性構文とプロパティ要素構文

C1Expander コンテンツ領域に単純な要素（書式設定されていない文字列、1つのコントロールなど）を追加する場合は、次に示すように、XAML マークアップの一般的な XML 属性を使用できます。

XAML

```
<c1:C1Expander Content="Hello World"/>
```

ただし、コンテンツ領域に、グリッドやパネルなどの複雑な要素を追加することもできます。このような場合は、次に示すように、プロパティ要素構文を使用できます。

XAML

```
<c1:C1Expander ExpandDirection="Down" Width="150" Height="55" Name="C1Expander1">
    <c1:C1Expander.Content>
        <StackPanel>
            <TextBlock Text="Hello"/>
            <TextBlock Text="World"/>
        </StackPanel>
    </c1:C1Expander.Content>
</c1:C1Expander>
```

展開と折りたたみ

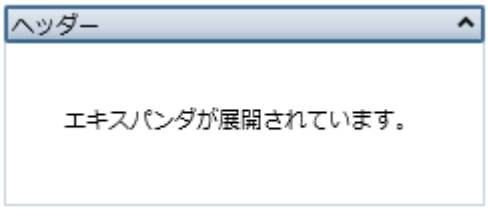
C1Expander コントロールの展開と折りたたみをカスタマイズするには、いくつかのオプションがあります。以下のセクションでは、初期展開状態、コントロールを展開する方向、および **C1Expander** コントロールの展開を抑止する方法について説明します。

初期展開状態

デフォルトでは、**IsExpanded** プロパティは **False** に設定されます。これにより、ページがロードされると、コントロールは折りたたまれた状態で表示されます。ページのロード時にコントロールを展開した状態にする場合は、**IsExpanded** プロパティを

True に設定します。

次の表に、この2つの展開状態の違いを示します。

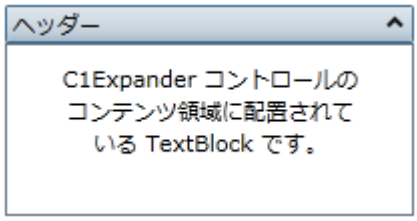
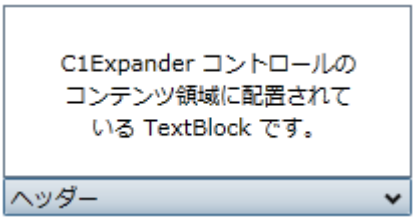
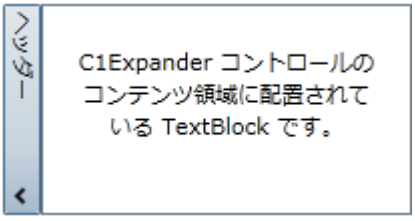
IsExpanded	結果
IsExpanded=False	
IsExpanded=True	

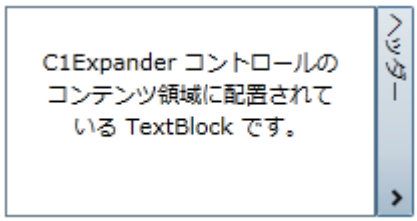
展開状態は、Blend、XAML、またはコードで設定できます。

展開方向

C1Expander コントロールには、**ExpandDirection** プロパティを使って展開方向を指定するオプションがあります。コントロールの展開方向を設定するほかに、ExpandDirection を変更すると、コントロールのコンテンツ領域に対するヘッダーの方向も変更されます。デフォルトでは、ExpandDirection プロパティは **Down** に設定され、コントロールは上から下に展開されます。

次の表に、**ExpandDirection** のそれぞれの設定状態を示します。

ExpandDirection	結果
Down	
Up	
Right	

Left	

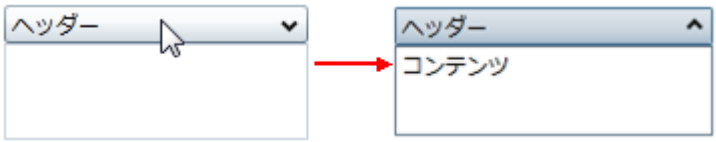
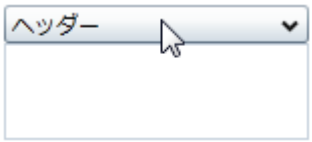
折りたたみ/展開方向は、Blend、XAML、またはコードで設定できます。展開方向の変更に関するタスク別ヘルプについては、「展開方向を変更する」を参照してください。

展開可能性

デフォルトでは、**C1Expander** コントロールの IsExpandable プロパティは **True** に設定され、ヘッダーバーをクリックするとコンテンツを展開できます。特定のイベントが発生するまで展開しないようにする必要がある場合などは、コントロールの展開を抑止する必要があります。このような場合は、IsExpandable プロパティを **False** に設定します。

IsExpandable プロパティを **False** に設定すると、ユーザーはコントロールを操作できなくなります。IsExpandable プロパティを False に設定すると、ユーザーがコントロールの上にマウスポインタを置いても何も起こりません。**IsExpandable** プロパティを **False** に設定すると、ユーザーがヘッダーをクリックしても、コントロールは折りたたまれたままになります。

次の表に、IsExpandable プロパティのそれぞれの設定の効果を示します。

IsExpandable	結果
True	
False	

XAML クイックリファレンス

このトピックでは、**C1Expander** 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。

次の XAML マークアップは、**TextBlock** コンテンツを含むエキスパンダを作成する方法を示します。

XAML

```
<c1:C1Expander Height="122" HorizontalAlignment="Left" Name="c1Expander1"
VerticalAlignment="Top" Width="179">
   <TextBlock Height="23" HorizontalAlignment="Left" Name="textBlock1"
Text="TextBlock" VerticalAlignment="Top">
       Hello World!
   </TextBlock>
</c1:C1Expander>
```

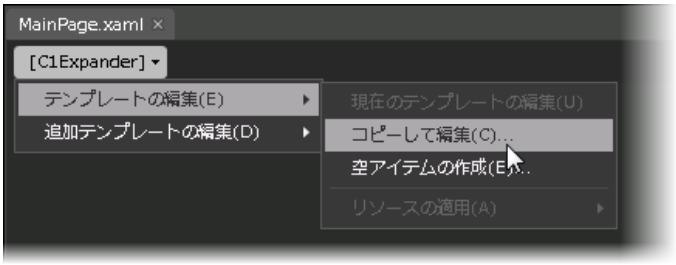
テンプレート

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」コントロールだからです。WPF/Silverlight アプリケーションで独自のユーザーインターフェイス（UI）、つまり「外観」を設計するのと同様に、**Expander for WPF/Silverlight** によって管理されるデータにも独自の 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、Expression Blend で、C1Expander 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。



メモ: メニューを使っ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。

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。

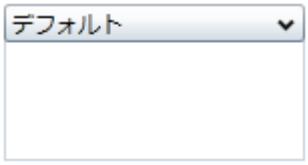
追加のテンプレート

テンプレートに加えて、**C1Expander** コントロールには追加テンプレートがいくつか含まれています。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 **C1Expander** 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。

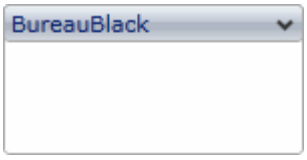
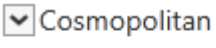
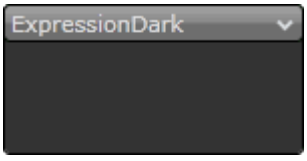
テーマ

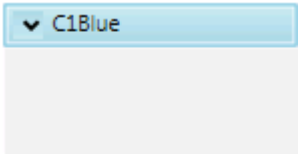
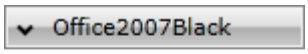
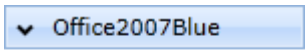
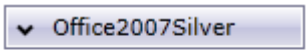
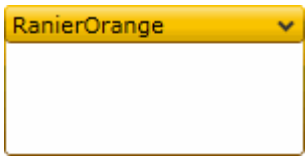
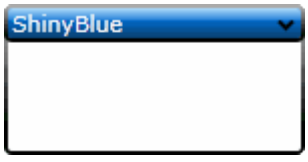
WPF/Silverlight テーマは、いくつかのコントロールの外観を定義するイメージ設定のコレクションです。テーマはアプリケーション内の複数のコントロールに適用できるため、テーマを使用すると、スタイル設定作業を繰り返さなくても、一貫性のあるコントロールを作成できます。

C1Expander コントロールをプロジェクトに追加すると、コントロールはデフォルトの青色のテーマで表示されます。



C1Expander コントロールには、Silverlight の複数のテーマ（BureauBlack、ExpressionDark、ExpressionLight、RainierOrange、ShinyBlue、WhistlerBlue）の中から1つテーマを設定できます。次の表に、テーマごとのサンプルを示します。

完全なテーマ名	外観
C1ThemeBureauBlack	
C1ThemeCosmopolitan	
C1ThemeExpressionDark	

C1ThemeExpressionLight	
C1Blue (WPF のみ)	
C1ThemeOffice2007Black	
C1ThemeOffice2007Blue	
C1ThemeOffice2007Silver	
C1ThemeOffice2010Black	
C1ThemeOffice2010Blue	
C1ThemeOffice2010Silver	
C1ThemeRainierOrange (Silverlight のみ)	
C1ThemeShinyBlue	
C1ThemeWhistlerBlue	

C1Expander コントロールにいずれかのテーマを追加するには、マークアップでこのコントロールに対してテーマを宣言しま

す。

HtmlHost (Silverlight のみ)

HtmlHost for Silverlight を使用して、Silverlight 内部から HTML および任意の URI コンテンツをレンダリングします。HTML ホストコントロール(**C1HtmlHost**)には、任意の HTML コンテンツをホストし、任意の URI や HTML テキストからコンテンツを表示できるフレームが用意されています。

メモ:このセクションの内容は、ComponentOne for Silverlight にのみ適用されます。

クイックスタート

このクイックスタートは、**HtmlHost for 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、**C1HtmlHost** コントロールを使って Silverlight アプリケーションで Web サイトを表示する Silverlight アプリケーションを作成します。Visual Studio で新しいプロジェクトを作成し、コントロールを追加してカスタマイズします。次に、**C1HtmlHost** コントロールで可能な実行時の操作を表示します。

手順 1:アプリケーションの作成

この手順では、HtmlHost for Silverlight を使って Web サイトを表示する Silverlight アプリケーションを Visual Studio で作成します。新しい Silverlight プロジェクトを作成し、コントロールをアプリケーションに追加します。

コントロールを設定して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、**[ファイル]**→**[新しいプロジェクト]**を選択します。
2. **[新しいプロジェクト]**ダイアログボックスの左ペインから言語を選択し、テンプレートリストから[Silverlight アプリケーション]を選択します。プロジェクトの[名前]を入力し、[OK]をクリックします。[新しい Silverlight アプリケーション]ダイアログボックスが表示されます。
3. **[OK]**をクリックしてデフォルト設定を受け入れ、[新しい Silverlight アプリケーション]ダイアログボックスを閉じると、プロジェクトが作成されます。MainPage.xaml ファイルが開きます。
4. プロジェクトの XAML ウィンドウで、カーソルを <Grid> タグと </Grid> タグの間に置き、1回クリックします。
5. Visual Studio のツールボックスに移動し、[StackPanel]アイコンをダブルクリックして、ページにパネルを追加します。
6. MainPage.xaml ファイルの <StackPanel> タグを更新します。次のようになります。
   <StackPanel Name="StackPanel1" Margin="10" Orientation="Vertical"></StackPanel>
   このマークアップは、マージンを追加し、パネルの方向を設定します。
7. プロジェクトの XAML ウィンドウで、カーソルを <StackPanel> タグと </StackPanel> タグの間に置き、1回クリックします。コントロールを StackPanel に追加します。
8. Visual Studio のツールボックスに移動し、[C1HtmlHost]アイコンをダブルクリックして、パネルにコントロールを追加します。
9. MainPage.xaml ファイルの <c1:C1HtmlHost/> タグを更新します。次のようになります。
   <c1:C1HtmlHost Name="C1HtmlHost1" Margin="5" SourceUri="http://www.grapecity.com/japan/" />
   このマークアップは、コントロールの名前を指定し、マージンを追加します。また、アプリケーションがロードされるときに **C1HtmlHost** コントロールに最初に表示される Web サイトまたはページを決定する SourceUri プロパティを設定します。
10. 次のマークアップを XAML ビューの タグの下に追加します。

```
XAML
```

```
<StackPanel Name="StackPanel2" Orientation="Horizontal">
    <TextBlock Height="23" HorizontalAlignment="Left" Margin="5"
Name="TextBlock1" Text="Source URI:" VerticalAlignment="Top" />
    <TextBox Height="23" Name="TextBox1" Width="200" Margin="5"
Text="http://www.grapecity.com/japan/" />
    <Button Content="ソース URI の設定" Height="23" Name="Button1" Margin="5" />
</StackPanel>
```

このマークアップは、**TextBlock**、**TextBox**、および **Button** を含む **StackPanel** を追加します。実行時に、ユーザーがテキストボックスに URL を入力してボタンを押すと、入力した Web サイトが C1HtmlHost コントロールに表示されます。マークアップは次のようになります。

XAML

```
<Grid x:Name="LayoutRoot" Background="White">
    <StackPanel Margin="10" Name="StackPanel1" Orientation="Vertical">
        <c1:C1HtmlHost Name="C1HtmlHost1" Margin="5"
SourceUri="http://www.componentone.com" c1:C1NagScreen.Nag="True" />
        <StackPanel Name="StackPanel2" Orientation="Horizontal">
            <TextBlock Height="23" HorizontalAlignment="Left" Margin="5"
Name="TextBlock1" Text="Source URI:" VerticalAlignment="Top" />
            <TextBox Height="23" Name="TextBox1" Width="200" Margin="5"
Text="http://www.grapecity.com/japan/" />
            <Button Content="ソース URI の設定" Height="23" Name="Button1"
Margin="5" />
        </StackPanel>
    </StackPanel>
</Grid>
```

11. ソリューションエクスプローラに移動して、YourProject.Web ノード（YourProject はプロジェクトの名前）を展開します。次に、YourProjectTestPage.aspx ファイル（YourProject はプロジェクトの名前）をダブルクリックして開きます。
12. .aspx ファイルで、 タグまでスクロールし、次のパラメータをパラメータリストの <object> タグと </object> タグの間に追加します。

XAML

```
<param name="windowless" value="true" />
```

13. すべての変更を保存して、MainPage.xaml ページに戻ります。

ここまでの成果

Silverlight アプリケーションを正しく作成および設定し、**C1HtmlHost** コントロールを含むコントロールをページに追加しました。次の手順では、機能を追加するコードをアプリケーションに追加します。

手順 2：コードの追加

前の手順で、Silverlight アプリケーションを設定しました。ただし、この時点でアプリケーションを実行しても、ボタンとテキストボックスは動作しません。この手順では、引き続き、機能を追加するコードをアプリケーションに追加します。

次の手順に従います。

1. ソリューションエクスプローラに移動し、MainPage.xaml ファイルを右クリックして[コードの表示]を選択し、コードビューに切り替えます。

2. コードビューで、次の import 文をコードページの先頭に追加します。

VisualBasic

```
Imports C1.Silverlight.Extended
```

C#

```
using C1.Silverlight.Extended;
```

3. MainPage.xaml.cs または MainPage.xaml.vb ファイルで MainPage クラスの他のすべてのメソッドの下に、次のイベントハンドラを追加します。

VisualBasic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs)
Handles Button1.Click
    Me.C1HtmlHost1.SourceUri = New Uri(String.Format(TextBox1.Text))
End Sub
```

C#

```
private void Button1_Click(object sender, RoutedEventArgs e)
{
    Cc1HtmlHost1.SourceUri = new Uri(string.Format(TextBox1.Text));
}
```

このコードは、ボタンの Click イベントを処理して、**C1HtmlHost** コントロールをカスタマイズします。

ここまでの成果

この手順では、機能を追加するコードをアプリケーションに追加しました。実行時に、ユーザーがテキストボックスに URL を入力してボタンをクリックすると、選択した Web サイトが **C1HtmlHost** コントロールによって表示されます。次の手順では、アプリケーションを実行し、HtmlHost for Silverlight の実行時の操作をいくつか確認します。

手順 3：アプリケーションの実行

Silverlight アプリケーションを作成して設定し、機能を追加するコードをアプリケーションに追加しました。最後に、アプリケーションを実行します。アプリケーションの実行時の操作を確認するには、次の手順に従います。

1. メニューから**[デバッグ]**→**[デバッグ開始]**を選択して、アプリケーションを実行します。アプリケーションは次の図のように表示されます。

C1HtmlHost コントロールに ComponentOne Web サイトがロードされた状態で、アプリケーションが表示されます。

2. テキストボックスに「http://www.google.co.jp」と入力し、[ソース URI の設定]ボタンをクリックします。C1HtmlHost コントロールに、Google の Web サイトが表示されます。
3. テキストボックスに「Silverlight」と入力し、[Google 検索]ボタンをクリックします。

検索結果を示すページが表示されます。

4. 検索結果リンクをクリックします。コントロールに、選択したページが表示されることを確認します。

5. 次に、リンクをクリックし、**C1HtmlControl** を使ってインターネット上のさまざまなページに移動します。C1HtmlControl によって表示されるページが Web ブラウザに表示されるページと同じように機能することを確認してください。

ここまでの成果

おめでとうございます!

これで **HtmlHost for Silverlight** クイックスタートは終了です。**HtmlHost for Silverlight** を使用して、選択した Web サイトを表示する簡単なアプリケーションを作成しました。

HtmlHost for Silverlight の機能の詳細については、「HtmlHost の使い方」トピックを参照してください。具体的なカスタマイズ例については、「タスク別ヘルプ」トピックを参照してください。

主な特長

HtmlHost for Silverlight には、次の主な特長があります。

- **ブラウザ内表示のサポート**
  HTML コンテンツをブラウザ内に表示できます。
- **HTML コンテンツの表示**
  Silverlight プラグイン内に既存の HTML コンテンツを表示するときには実際のブラウザを使用するため、正確なレンダリングと操作が可能です。
- **URI からの HTML のロード**
  HtmlHost for Silverlight では、アプリケーションサーバー以外に、使用可能な任意の URI から HTML コンテンツをロードすることができます。
- **テキストからの HTML のロード**
  文字列として格納されている HTML コンテンツを表示できます。
- **HTML コンテンツへのアクセス**
  HtmlHost for Silverlight では、コンテンツがすべてロードされるとイベントが発生します。また、開発者は、Silverlight のブラウザオブジェクトモデルを使ってコンテンツにアクセスすることができます。
- **Silverlight レイアウトシステムのサポート**

HtmlHost for Silverlight では、Silverlight のコントロールレイアウトに従って、そのコンテンツが並べ替えらえます。

- **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、それを使って UI にスタイルを追加できます。

HTML コンテンツの表示

C1HtmlHost コントロールは、HTML の iFrame タグのように動作し、Silverlight ページ内に HTML コンテンツを表示できるようにします。これは、ページを構成する HTML 内に島のように Silverlight プラグインが配置されるという一般的なシナリオの逆です。C1HtmlHost コントロールを使用すると、Silverlight ページ内に島のように HTML を配置できます。

多くの場合、Silverlight アプリケーションでは、Silverlight プラグイン自体の中に既存の HTML コンテンツを表示するため（ホストページの DOM で要素を制御するのではなく。ただし、Silverlight ではこれも可能）、この概念は役立ちます。

HtmlHost for Silverlight は iFrame を使って動作します。**C1HtmlHost** コントロールは、JavaScript を通して iFrame を追加し、Silverlight プラグインを通して C1HtmlHost コントロールがある位置に iFrame を正確に配置します。次に、このコントロールはレイアウト内の変更を監視し、iFrame を継続的に更新します。

ページによっては、C1HtmlHost コントロールに表示できない場合があります。たとえば、表示対象のページが iFrame オブジェクトで表示できないようにカスタマイズされている場合などです。C1HtmlHost コントロールは、その動作の性質上、Silverlight プラグインの上位にある iFrame オブジェクトを使用するため、これは予期されることです。

コンテンツの挿入

C1HtmlHost には、次の2つの方法でコンテンツを挿入できます。

- SourceHtml プロパティを使用して、コントロール内に表示する HTML 文字列を指定します。このオプションは、アプリケーションで、HTML コンテンツを構築したりロードする場合に便利です。
- SourceUri プロパティを使用して、コントロール内に表示する URL を指定します。このオプションは、アプリケーションで、指定された URL に既にあるコンテンツを表示する場合に便利です。

C1HtmlHost コントロールには重要な要件があります。Silverlight プラグイン内に HTML コンテンツを表示するには、プラグインの **Windowless** プロパティを **"True"** に設定する必要があります。ここでアプリケーションを実行すると、次のようなエラーメッセージが表示されます。

> **メモ**: このコントロールを使用するには、`Silverlight` プラグインの **`Windowless`** パラメータを **`True`** に設定する必要があります。

コントロールがプラグインのプロパティを変更することはできないため、ページの作成者がこれを行う必要があります。必要な変更を行うには、プラグインを作成するページを開き、**Windowless** プロパティを **"True"** に設定する行を追加します。詳細については、「ウィンドウレスモード」を参照してください。

ウィンドウレスモード

HtmlHost for Silverlight では、Silverlight プラグインの **Windowless** パラメータを **True** に設定する必要があります。ウィンドウレスモードでは、Silverlight プラグインは独自のレンダリングウィンドウを使用しません。代わりに、プラグインコンテンツがブラウザウィンドウに直接表示されます。このため、プラグインとそのコンテンツの両方で透過背景画像が指定されている場合、Silverlight コンテンツが HTML コンテンツと重なって、または混ざっているように見せることができます。

Windowless パラメータを設定するには、次の手順に従います。

1. **C1HtmlHost** コントロールを含む Silverlight アプリケーションを作成します。
2. ソリューションエクスプローラに移動し、**YourProject.Web** ノードを展開します。YourProject はプロジェクトの名前です。
3. ソリューションエクスプローラで、Silverlight プラグインが宣言されているページをダブルクリックします。たとえば、

YourProjectTestPage.aspx ファイルまたは **YourProjectTestPage.html** ファイルをダブルクリックしてページを開きます。YourProject はプロジェクトの名前です。

4. ページで、 タグまでスクロールし、次のパラメータをパラメータリストの <object> タグと </object> タグの間に追加します。
<param name="windowless" value="true" />

5. 変更を保存して、MainPage.xaml ページに戻ります。

Windowless パラメータが **True** に設定されていない場合、アプリケーションを実行する際に警告が表示されることがあります。

フレームの境界線

デフォルトでは、**C1HtmlHost** コントロールはフレーム内に表示されます。また、デフォルトでは **C1HtmlHost** コントロールの周囲に境界線が表示されます。たとえば、次の図では、**C1HtmlHost** コントロールの上側および左側にフレームの境界線が表示されています。

コントロールとアプリケーションをよりシームレスに統合したい場合などに、このフレームの境界線を非表示にすることができます。FrameBorder プロパティを **"False"** に設定すると、**C1HtmlHost** コントロールの周囲のフレームが非表示になります。たとえば、次の図では、フレームが非表示になっています。

フレームを非表示にすると、**C1HtmlHost** コントロールとアプリケーションの背景との統合が高まります。

HtmlHost の使い方

HtmlHost for Silverlight では、Silverlight アプリケーション内に Web サイトまたは HTML コンテンツを簡単かつ信頼性の高い方法で表示することができます。次のトピックでは、**HtmlHost for Silverlight** での作業に役立つ情報を示します。

タスク別ヘルプ

次のタスク別ヘルプトピックは、ユーザーの皆様が Visual Studio および Expression Blend に精通しており、C1HtmlHost コントロールの一般的な使用方法を理解していることを前提としています。**HtmlHost for 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、**HtmlHost for Silverlight** 製品を使って特定のタスクを行うための方法を提供します。また、ほとんどのタスク別ヘルプトピックは、新しい Silverlight プロジェクトが作成されており、そのアプリケーションに C1HtmlHost コントロールが追加されていることを前提としています。

Web サイトを表示する

HtmlHost for Silverlight では、1つのプロパティを設定するだけで外部の Web サイトを表示することができます。SourceUri プロパティは、コントロールに表示するサイトの URI を指定します。**SourceUri** プロパティは、表示する任意の Web サイトに設定できます（外部のサイトも指定可能）。

設計時

Expression Blend で **SourceUri** プロパティを設定するには、次の手順に従います。

1. **C1HtmlHost** コントロールをクリックして選択します。
2. [プロパティ]ウィンドウに移動し、[SourceUri]項目を探します。
3. **[SourceUri]**項目の横にあるテキストボックスをクリックし、「http://www.componentone.com」と入力します。

XAML の場合

たとえば、**SourceUri** プロパティを設定するには、次に示すように SourceUri="" を <c1:C1HtmlHost> タグに追加します。

<c1:C1HtmlHost Name="c1HtmlHost1" SourceUri="http://www.componentone.com" />

コードの場合

たとえば、SourceUri プロパティを設定するには、次のコードをプロジェクトに追加します。

VisualBasic

```
Me.C1HtmlHost1.SourceUri = "http://www.componentone.com"
```

C#

```
this.C1HtmlHost1.SourceUri = "http://www.componentone.com";
```

ここまでの成果

SourceUri プロパティを設定し、実行時に ComponentOne Web サイトが表示されるように **C1HtmlHost** コントロールをカスタマイズしました。アプリケーションを実行し、実行時に **C1HtmlHost** コントロールに ComponentOne Web サイトが表示されることを確認します。

HTML コンテンツを表示する

HtmlHost for Silverlight では、HTML コンテンツを表示することができます。**SourceHtml** プロパティを、実行時に変換する HTML マークアップに設定します。たとえば、次のような手順が考えられます。

設計時

SourceHtml プロパティを設定するには、次の手順に従います。

1. **C1HtmlHost** コントロールをクリックして選択します。
2. [プロパティ]ウィンドウに移動し、**[SourceHtml]**項目を探します。
3. SourceHtml 項目の横のテキストボックスに、「ブラウザーに HTML でホストされています！」などのテキストを入力します。

XAML の場合

たとえば、SourceHtml プロパティを設定するには、次に示すように SourceHtml を <c1:C1HtmlHost> タグに追加します。

XAML

```
<c1:C1HtmlHost Name="c1HtmlHost1" SourceHtml="ブラウザーに &lt;b&gt;HTML&lt;/b&gt; でホストされています！"/>
```

コードの場合

たとえば、SourceHtml プロパティを設定するには、次のコードをプロジェクトに追加します。

VisualBasic

```
Me.C1HtmlHost1.SourceHtml="ブラウザーに &lt;b&gt;HTML&lt;/b&gt; でホストされています！"
```

C#

```
this.C1HtmlHost1.SourceHtml="ブラウザーに &lt;b&gt;HTML&lt;/b&gt; でホストされています！";
```

ここまでの成果

これで、実行時に **C1HtmlHost** コントロールに HTML テキストが表示されます。このマークアップでは、"<" および ">" が "<" 記号および ">" 記号として使用されています。このため、テキストを太字にするタグは、通常では"<b>"ですが、このマークアップでは "<b>" と記述されています。

フレームの境界線を非表示にする

また、デフォルトでは **C1HtmlHost** コントロールの周囲に境界線が表示されます。コントロールとアプリケーションをよりシームレスに統合したい場合などに、このフレームの境界線を非表示にすることができます。**HtmlHost for Silverlight** では、1つのプロパティを設定するだけでフレームの境界線を非表示にすることができます。**FrameBorder** プロパティでは、コントロールの周囲にフレームを表示するかどうかを指定します。

設計時

FrameBorder プロパティを設定するには、次の手順に従います。

1. **C1HtmlHost** コントロールをクリックして選択します。
2. **[プロパティ]**ウィンドウに移動し、**[FrameBorder]**項目を探します。
3. **FrameBorder** 項目の横にあるチェックボックスをオフにします。

XAML の場合

たとえば、**FrameBorder** プロパティを設定するには、次に示すように **FrameBorder="False"** を <c1:C1HtmlHost> タグに追加します。
<c1:C1HtmlHost Name="c1HtmlHost1" FrameBorder="False" />

コードの場合

たとえば、**FrameBorder** プロパティを設定するには、次のコードをプロジェクトに追加します。

Visual Basic

```
Me.C1HtmlHost1.FrameBorder = False
```

C#

```
this.C1HtmlHost1.FrameBorder = false;
```

ここまでの成果

実行時にコントロールの周りにフレームが表示されないように、**FrameBorder** プロパティを設定しました。アプリケーションを実行し、**C1HtmlHost** コントロールの周囲にフレームが表示されないことを確認します。

PropertyGrid

PropertyGrid for WPF/Silverlight は、Windows プラットフォームに組み込まれた標準の **PropertyGrid** コントロールの WPF バージョンです。このコントロールには 10 種類以上の組み込みエディタが用意されており、このコントロールを使用して任意のクラスを簡単に編集できます。**PropertyGrid for WPF/Silverlight** を使用して、任意の .NET オブジェクトのプロパティを参照および編集できます。

PropertyGrid for WPF クイックスタート

Microsoft の標準 **PropertyGrid** コントロールと同様に、**C1PropertyGrid** コントロールは **SelectedObject** プロパティに基づいて動作します。このプロパティを設定すると、コントロールにはオブジェクトのパブリックプロパティが表示され、それらをユーザーが編集できるようになります。このクイックスタートでは、WPF アプリケーションと、コントロールの情報を表示して編集するためのユーザーインターフェイスとなる **C1PropertyGrid** を定義します。

手順 1：アプリケーションの作成

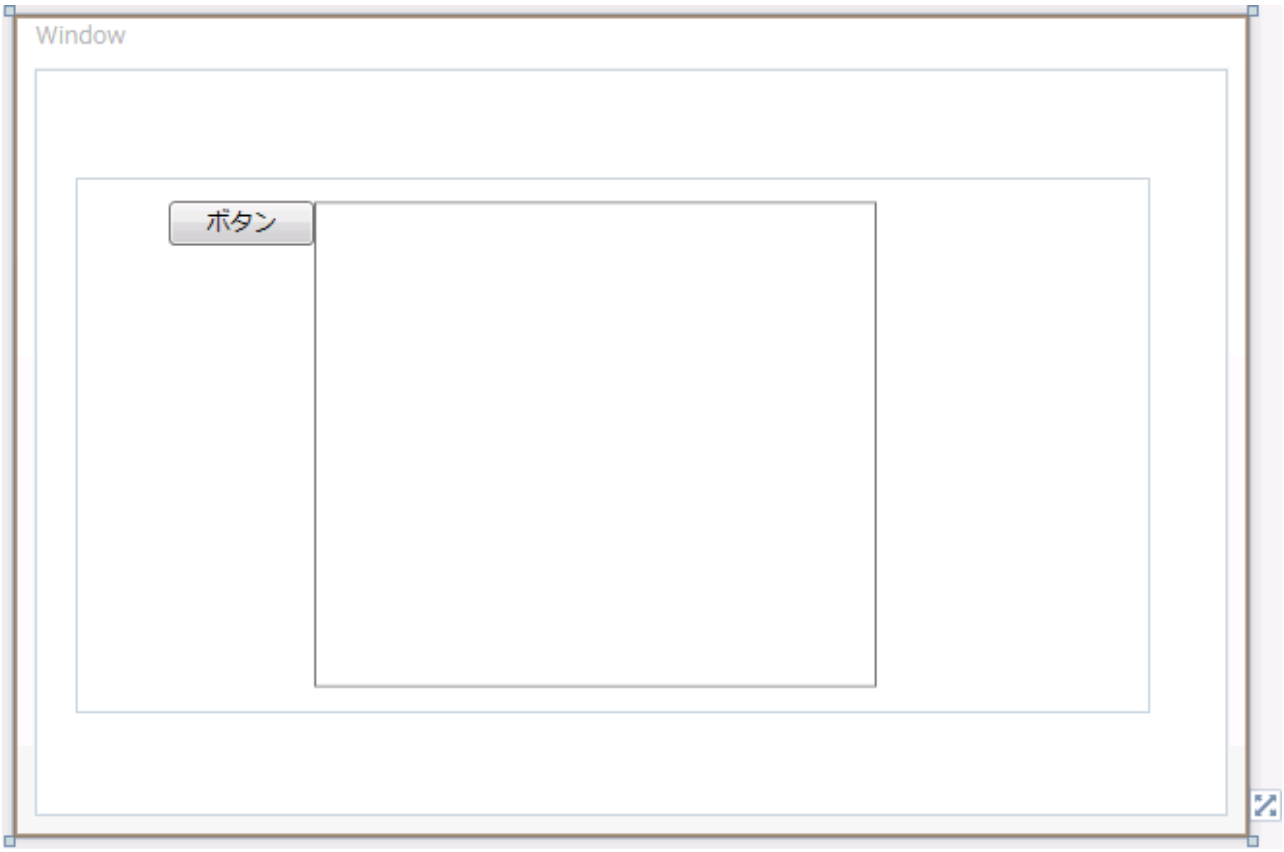
この手順では、**PropertyGrid for WPF** を使用して WPF アプリケーションを作成します。**C1PropertyGrid** コントロールをアプリケーションに追加すると、機能的な標準プロパティウィンドウのようなインターフェイスが完成します。これを使用して、ユーザーは、任意の WPF オブジェクトのプロパティやメソッドを参照したり編集することができます。プロジェクトをセットアップし、**C1PropertyGrid** コントロールをアプリケーションに追加するには、次の手順に従います。

1. Visual Studio で新しい WPF プロジェクトを作成します。
2. ソリューションエクスプローラで、**参照** 項目を右クリックし、**[参照の追加]**を選択します。**C1.WPF**、**C1.WPF.Extended** アセンブリを選択し、[OK]をクリックしてプロジェクトに参照を追加します。
3. Visual Studio のツールボックスに移動し、**[C1PropertyGrid]**アイコンをダブルクリックして、ウィンドウにコントロールを追加します。
4. Visual Studio のツールボックスで、**[Button]**項目をダブルクリックして標準のボタンコントロールをアプリケーションに追加します。C1PropertyGrid コントロールを使用して、ボタンのプロパティを設定および表示します。
5. ウィンドウのサイズを変更し、ウィンドウ内の **C1PropertyGrid** コントロールと **Button** コントロールのサイズと位置を変更します。
6. コントロールを選択し、System.InvalidOperationException: 内部エラー: 内部 WPF コードが、解除済みと既にマークされている BindingExpression を再アクティブ化しようとしました。ウィンドウで、Name を「TextButton」に設定します。
7. C1PropertyGrid コントロールを選択し、[プロパティ]ウィンドウで、Height を 250 に、Width を 290 に設定します。XAML は次のようになります。

XAML

```
<StackPanel HorizontalAlignment="Center" VerticalAlignment="Center"
Orientation="Horizontal">
<Button Height="23" Width="75" Margin="48,12,0,0" Name="TextButton"
HorizontalAlignment="Left"
 VerticalAlignment="Top" Content="ボタン"/>
<c1:C1PropertyGrid Name="c1PropertyGrid1" Height="250" Width="290" />
</StackPanel>
```

ページのデザインビューは次の図のようになります（フォームで C1PropertyGrid コントロールを選択している場合）。



これで、アプリケーションのユーザーインターフェイスのセットアップは終了しましたが、**C1PropertyGrid** コントロールには何もコンテンツが入っていません。次の手順では、**Button** コントロールの特定のプロパティを表示するように **C1PropertyGrid** コントロールを設定し、さらにアプリケーションにコードを追加して、コントロールに機能を追加します。

手順 2：アプリケーションのカスタマイズ

前の手順では、WPF アプリケーションを作成し、アプリケーションに **Button** コントロールと C1PropertyGrid コントロールを追加しました。この手順では、**Button** コントロールの特定のプロパティが表示されるように **C1PropertyGrid** コントロールをカスタマイズします。

C1PropertyGrid コントロールをカスタマイズして **Button** コントロールに接続するには、次の手順に従います。

1. XAML ビューに切り替えます。XAML で C1PropertyGrid コントロールを Button コントロールに連結し、それらのコントロールをカスタマイズします。
2. <c1:C1PropertyGrid> タグに SelectedObject="{Binding ElementName=TextButton, Mode=OneWay}" を追加します。次のようになります。

   XAML

   ```
   <c1:C1PropertyGrid Margin="130,12,30,12" Name="c1PropertyGrid1"
   SelectedObject="{Binding ElementName=TextButton, Mode=OneWay}">
   ```

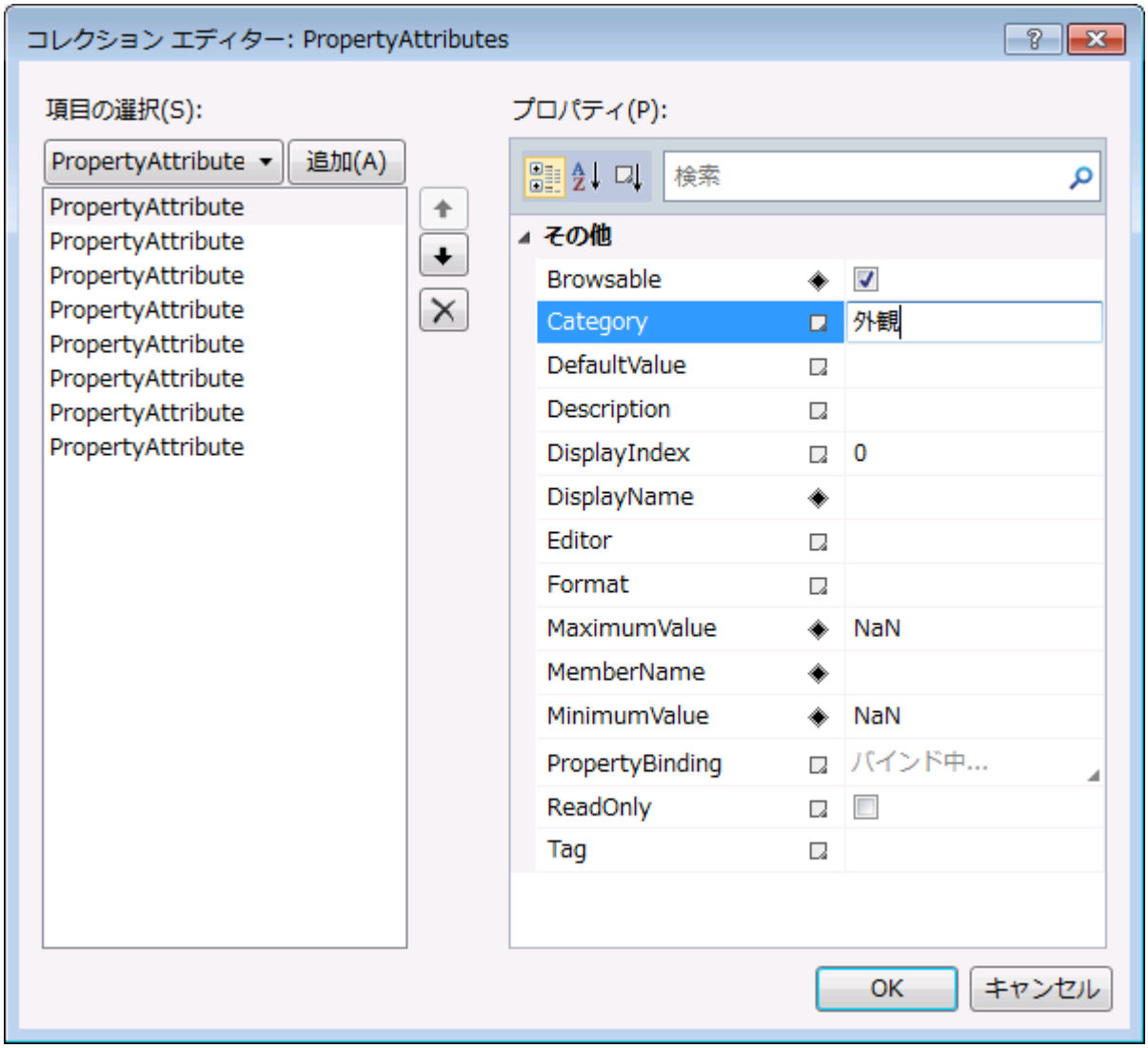
   デザインビューを見ると、C1PropertyGrid コントロールにボタンのすべてのプロパティが反映されていることがわかります。次の手順では、特定のプロパティのみが表示されるように、C1PropertyGrid コントロールをカスタマイズします。
3. **[プロパティ]**ウィンドウで[**AutoGenerateProperties**]チェックボックスをオフにします。これで指定したプロパティ以外は表示されなくなります。
4. **[プロパティ]**ウィンドウで[**PropertyAttributes**]コレクションを探し、その隣にある省略符ボタンをクリックします。**[コレクションエディター：PropertyAttributes]**ダイアログボックスが表示されます。

5. [コレクションエディター:PropertyAttributes]ダイアログボックスで、[追加]ボタンをクリックします。この手順をあと7回繰り返して、合計8個の PropertyAttribute 項目を作成します。

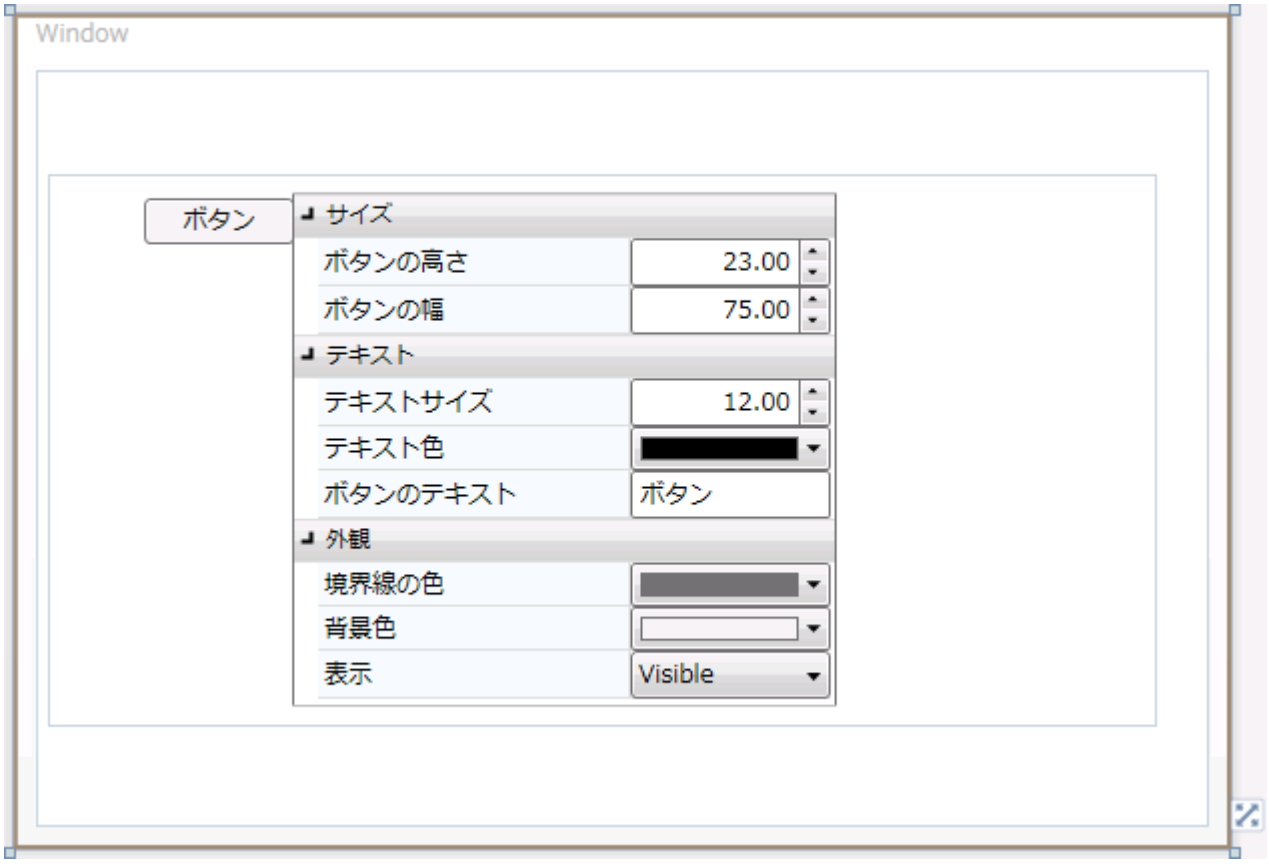
6. 今追加した項目の右側の[プロパティ]ペインで、次のプロパティを設定します。

PropertyAttribute	Category	DisplayName	MemberName
[0] PropertyAttribute	外観	背景色	Background
[1] PropertyAttribute	外観	境界線の色	BorderBrush
[2] PropertyAttribute	外観	表示	Visibility
[3] PropertyAttribute	サイズ	ボタンの高さ	Height
[4] PropertyAttribute	サイズ	ボタンの幅	Width
[5] PropertyAttribute	テキスト	ボタンのテキスト	Content
[6] PropertyAttribute	テキスト	テキスト色	Foreground
[7] PropertyAttribute	テキスト	テキストサイズ	FontSize

Category は、項目が表示されるセクションを指定します。**DisplayName** は、項目に表示される名前を示します。**MemberName** は、メンバの実際の名前を示します。



7. [OK]ボタンをクリックして、[コレクションエディター：PropertyAttributes]ダイアログボックスを閉じ、設定を変更します。設計時のページは次の図のようになります。

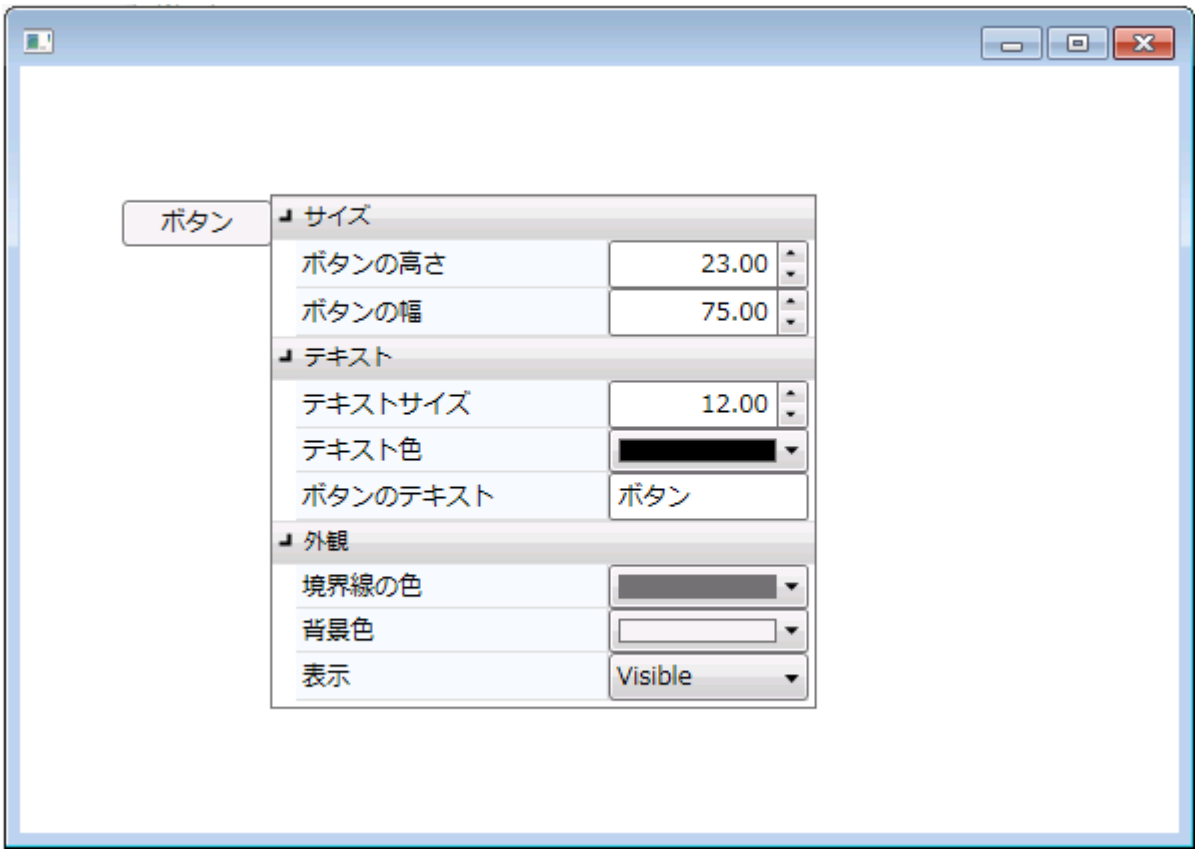


この手順では、**Button** コントロールの特定のプロパティが表示されるように **C1PropertyGrid** コントロールをカスタマイズしました。次の手順では、アプリケーションを実行し、実行時の操作をいくつか確認します。

手順 3：アプリケーションの実行

WPF アプリケーションを作成して、アプリケーションの外観をカスタマイズしました。次に、アプリケーションを実行します。アプリケーションを実行して **PropertyGrid for WPF** の実行時の動作を確認するには、次の手順に従います。

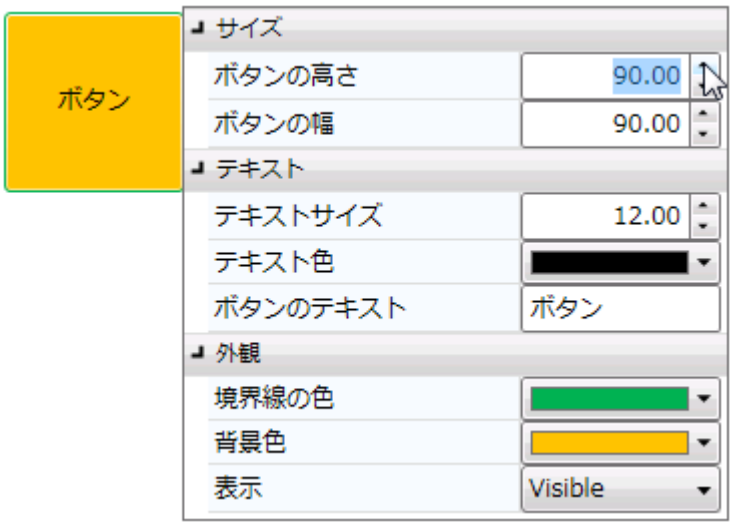
1. **[デバッグ]**メニューから**[デバッグ開始]**を選択し、実行時にアプリケーションがどのように表示されるかを確認します。アプリケーションは次の図のように表示されます。



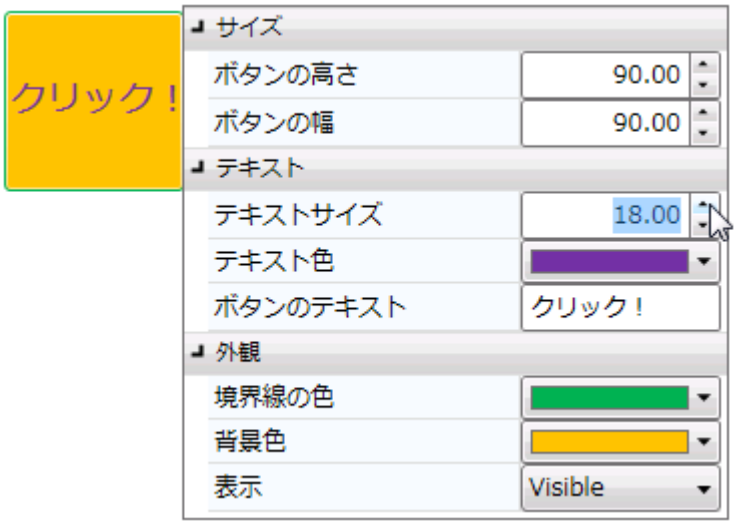
2. **背景色** のドロップダウン矢印をクリックし、表示されるカラーピッカーからオレンジなどの色を選択します。ボタンの背景色が選択した色に変わります。

3. **境界線の色** のドロップダウン矢印をクリックし、表示されるカラーピッカーから緑などの色を選択します。

4. **[ボタンの高さ]**および**[ボタンの幅]**数値ボックスに値を入力してボタンのサイズを変更します。たとえば、両方の値に「90」と入力します。アプリケーションは次の図のように表示されます。



5. **[ボタンのテキスト]**ボックスに「クリック！」などの文字列を入力します。
6. **テキスト色** のドロップダウン矢印をクリックし、表示されるカラーピッカーから紫などの色を選択します。
7. **[テキストサイズ]**の隣にある**上向き**または**下向き**矢印をクリックして、ボタンコントロールに表示されるテキストのサイズを変更します。たとえば、値を 18 に設定します。アプリケーションは次のようになります。



おめでとうございます。これで、**PropertyGrid for WPF** クイックスタートは終了です。このクイックスタートでは、ページに **C1PropertyGrid** コントロールと **Button** コントロールを追加し、**C1PropertyGrid** コントロールを Button にリンクしました。さらに、コントロールをカスタマイズし、**PropertyGrid for WPF** で実行時に可能な操作を見ました。

PropertyGrid for Silverlight クイックスタート

Microsoft の標準 **PropertyGrid** コントロールと同様に、**C1PropertyGrid** コントロールは **SelectedObject** プロパティに基づいて動作します。このプロパティを設定すると、コントロールにはオブジェクトのパブリックプロパティが表示され、それらをユーザーが編集できるようになります。このクイックスタートでは、Silverlight アプリケーションと、コントロールの情報を表示して編集するためのユーザーインターフェイスとなる **C1PropertyGrid** を定義します。

手順 1：アプリケーションの作成

この手順では、Microsoft Expression Blend で PropertyGrid for Silverlight を使って Silverlight アプリケーションを作成します。C1PropertyGrid コントロールをアプリケーションに追加すると、機能的な標準プロパティウィンドウのようなインターフェイス

が完成します。これを使用して、ユーザーは、任意の .NET オブジェクトのプロパティやメソッドを参照したり編集することができます。

プロジェクトをセットアップし、**C1PropertyGrid** 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. Expression Blend で、**[ファイル]**→**[新しいプロジェクト]**を選択します。
2. **[新しいプロジェクト]**ダイアログボックスで、左ペインからプロジェクトの種類として**[Silverlight]**を選択し、右ペインから**[Silverlight アプリケーション + Web サイト]**を選択します。プロジェクトの**[名前]**と**[場所]**を入力し、ドロップダウンボックスで[言語]を選択し、**[OK]**をクリックします。 新しいアプリケーションが作成され、MainPage.xaml ファイルがデザインビューで開きます。
3. **[プロジェクト]**ウィンドウに移動し、プロジェクトファイルリストで**[参照]**フォルダを右クリックします。コンテキストメニューから**[参照の追加]**を選択し、**C1.Silverlight.dll および C1.Silverlight.Extended.dll** アセンブリを見つけて選択し、[開く]をクリックします。ダイアログボックスが閉じ、プロジェクトに参照が追加されます。
4. ツールボックスで、**[アセット]**ボタン(二重山かっこアイコン)をクリックして、**[アセット]**ダイアログボックスを開きます。
5. **[アセットライブラリ]**ダイアログボックスで、左ペインから**[コントロール]**項目を選択し、右ペインで**[C1PropertyGrid]**アイコンをクリックします。**[C1PropertyGrid]**アイコンがツールボックスの**[アセット]**ボタンの下に表示されます。
6. **UserControl** のデザイン領域をクリックして選択します。Visual Studio とは異なり Blend では、次の手順に示すように、Silverlight コントロールを直接デザインサーフェスに追加できます。
7. ツールボックスの**[StackPanel]**ボタンをダブルクリックして、ページに追加します。**[StackPanel]**ボタンが表示されない場合は、[Grid]パネルボタンをクリックして StackPanel を選択する必要があります。
8. [プロパティ]ウィンドウで、**StackPanel** の次のプロパティを設定します。
   - **Height** および **Width** プロパティを "Auto" に設定します。
   - **Orientation** プロパティを **Horizontal** に設定します。
   - **HorizontalAlignment** および **VerticalAlignment** を **Center** に設定して、コントロールをパネルの中央に配置します。
9. **[オブジェクトとタイムライン]ペイン**で **StackPanel** を選択したまま、ツールボックスの標準**[Button]**コントロールをダブルクリックしてパネルに追加します。**C1PropertyGrid** コントロールを使用して、このコントロールのプロパティを設定します。
10. コントロールを選択し、[プロパティ]ウィンドウで、**Name** プロパティを "**button1**"、**Width** を **75**、**Height** を **290** に設定します。
11. **[オブジェクトとタイムライン]**ペインで **StackPanel** を選択し、ツールボックスの[C1PropertyGrid]アイコンをダブルクリックして、このコントロールをパネルに追加します。
12. **[オブジェクトとタイムライン]**ペインで **C1PropertyGrid** コントロールをクリックし、[プロパティ]ウィンドウに移動して、次のプロパティを設定します。
    - コード内でコントロールにアクセスできるように、**Name** を "c1propertygrid1" に設定して名前を付けます。
    - **Width** を "250" に、**Height** を "290" に設定します。
    - **HorizontalAlignment** および **VerticalAlignment** を Center に設定して、コントロールをパネルの中央に配置します。

    XAML は次のようになります。

```
XAML
```

```
<StackPanel HorizontalAlignment="Center" VerticalAlignment="Center"
Orientation="Horizontal">
    <Button Width="75" Height="290" Content="ボタン"/>
    <c1:C1PropertyGrid Height="250" Width="290" HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</StackPanel>
```

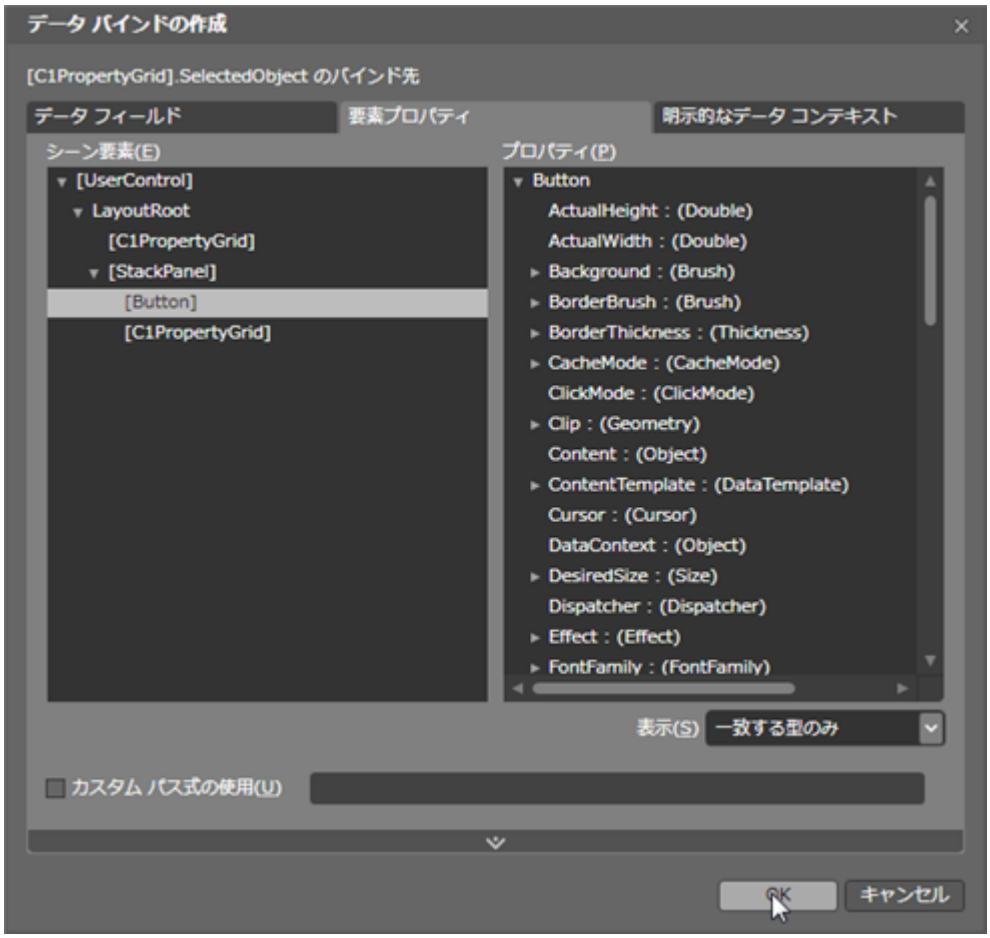
これで、アプリケーションのユーザーインターフェイスのセットアップは終了しましたが、**C1PropertyGrid** コントロールには何もコンテンツが入っていません。次の手順では、**Button** コントロールの特定のプロパティを表示するように **C1PropertyGrid** コントロールを設定し、さらにアプリケーションにコードを追加して、コントロールに機能を追加します。

手順 2:コントロールの設定

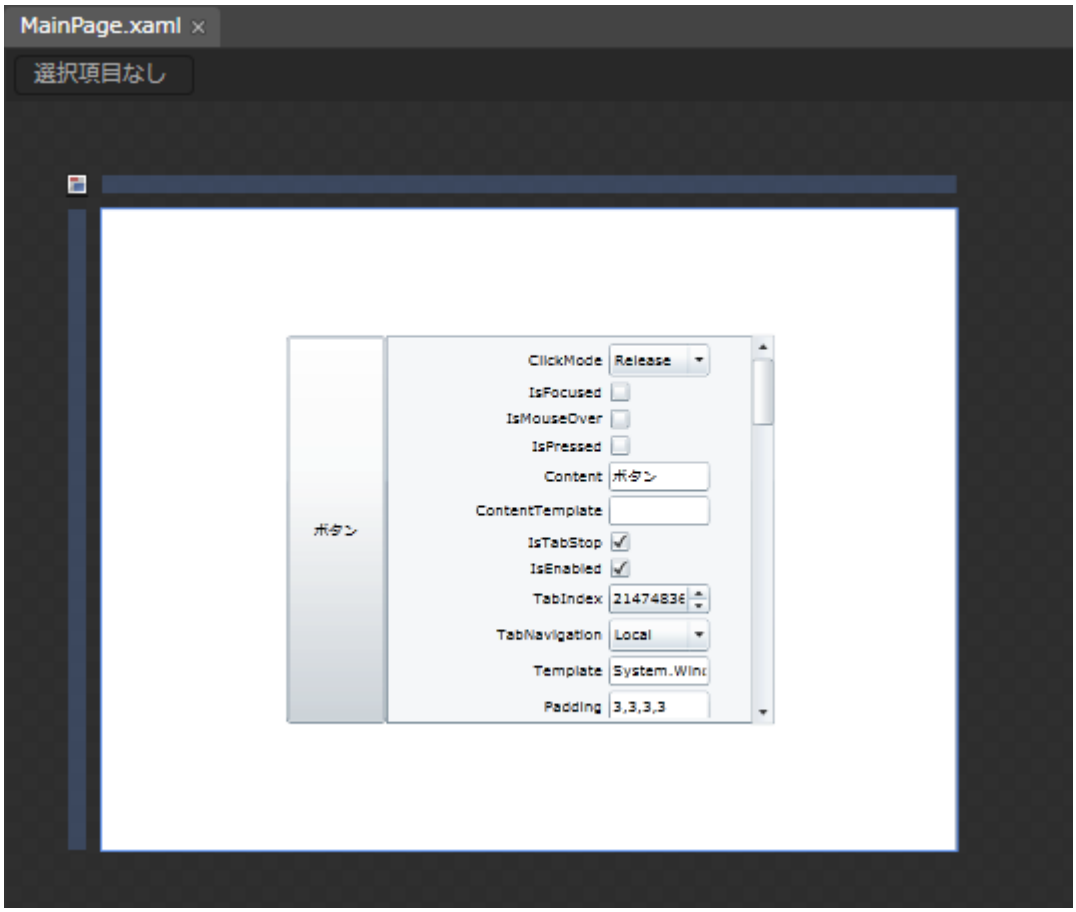
前の手順では、Silverlight アプリケーションを作成し、アプリケーションに **Button** コントロールと **C1PropertyGrid** コントロールを追加しました。この手順では、**Button** コントロールの特定のプロパティが表示されるように **C1PropertyGrid** コントロールをカスタマイズします。

C1PropertyGrid コントロールをカスタマイズして **Button** コントロールに接続するには、次の手順に従います。

1. **[オブジェクトとタイムライン]**ペインで **c1propertygrid** を選択し、[プロパティ]ウィンドウに移動します。
2. [プロパティ]ウィンドウで、**SelectedObject** プロパティを探します。その隣にある小さな四角形をクリックして高度なプロパティオプションにアクセスし、**[データバインド]**オプションを選択します。**[データのバインドの作成]**ダイアログボックスが表示されます。
3. **[データのバインドの作成]**ダイアログボックスで**[要素プロパティ]**タブをクリックします。
4. 左側の**[シーン要素]**ウィンドウで **button** を選択し、**[OK]**をクリックします。



C1PropertyGrid コントロールにボタンのプロパティがすべて表示されています。

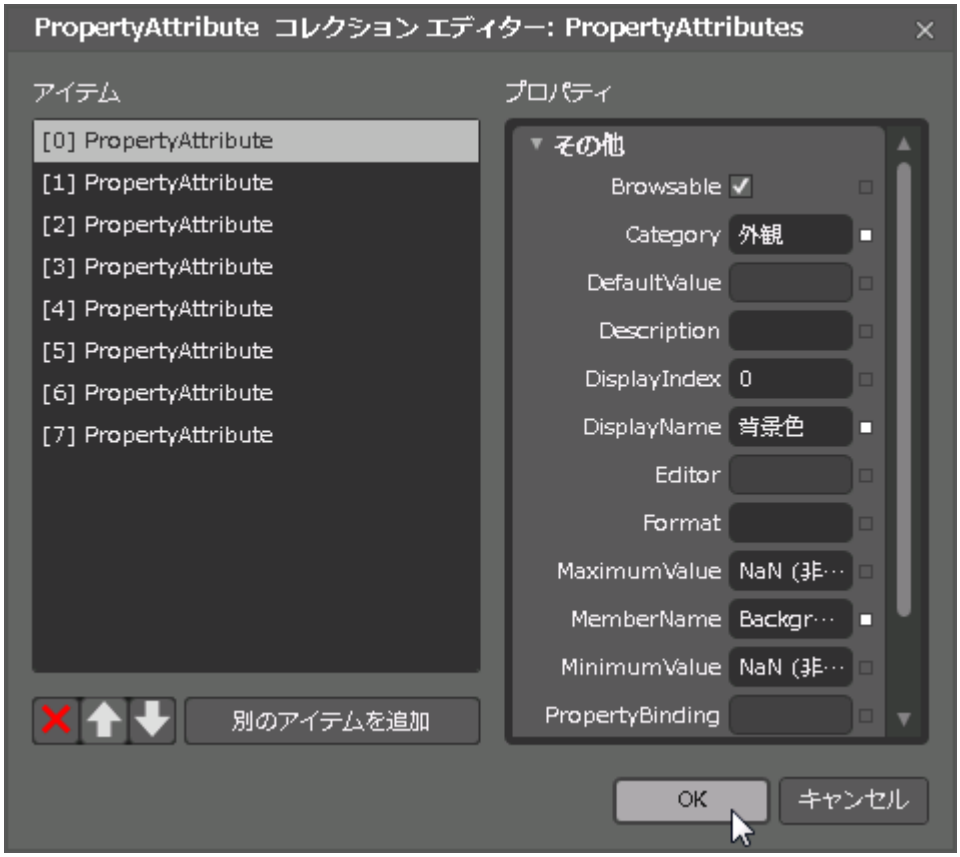


次の手順では、特定のプロパティのみが表示されるように、**C1PropertyGrid** コントロールをカスタマイズします。

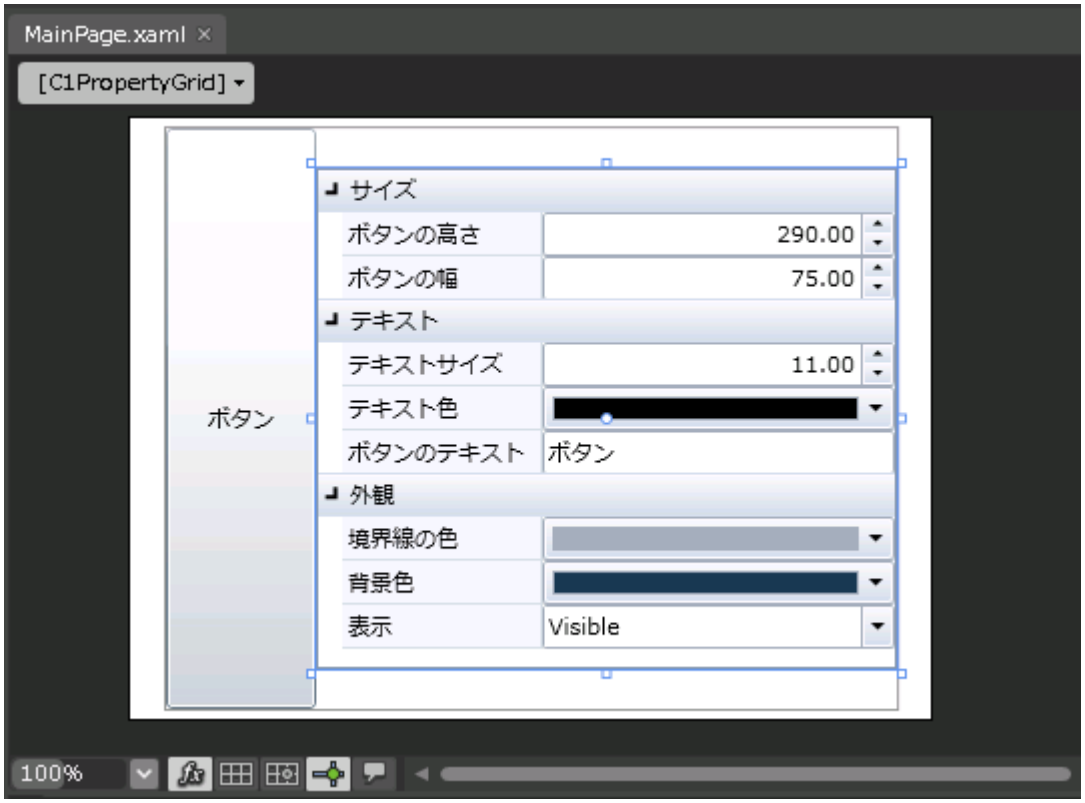
5. [プロパティ]ウィンドウで **AutoGenerateProperties** チェックボックスをオフにします。これで指定したプロパティ以外は表示されなくなります。
6. [プロパティ]ウィンドウで **PropertyAttributes** コレクションを探し、その隣にある省略符ボタンをクリックします。[プロパティ属性コレクションエディタ]ダイアログボックスが表示されます。
7. **[プロパティ属性コレクションエディタ]**ダイアログボックスで、**[別のアイテムを追加]**ボタンをクリックします。この手順をあと7回繰り返して、0から7までの番号が付いた合計8個の **PropertyAttribute** 項目を作成します。
8. 今追加した項目の右側の[プロパティ]ペインで、次のプロパティを設定します。

PropertyAttribute	Category	DisplayName	MemberName
[0] PropertyAttribute	Appearance	背景色	Background
[1] PropertyAttribute	Appearance	境界線の色	BorderBrush
[2] PropertyAttribute	Appearance	表示	Visibility
[3] PropertyAttribute	Size	ボタンの高さ	Height
[4] PropertyAttribute	Size	ボタンの幅	Width
[5] PropertyAttribute	Text	ボタンのテキスト	Content
[6] PropertyAttribute	Text	テキスト色	Foreground
[7] PropertyAttribute	Text	テキストサイズ	FontSize

Category は、項目が表示されるセクションを指定します。**DisplayName** は、項目に表示される名前を示します。**MemberName** は、メンバの実際の名前を示します。



9. **[OK]**ボタンをクリックして、**[プロパティ属性コレクションエディタ]**ダイアログボックスを閉じ、設定を変更します。設計時のページは次の図のようになります。



この手順では、**Button** コントロールの特定のプロパティが表示されるように **C1PropertyGrid** コントロールをカスタマイズしま

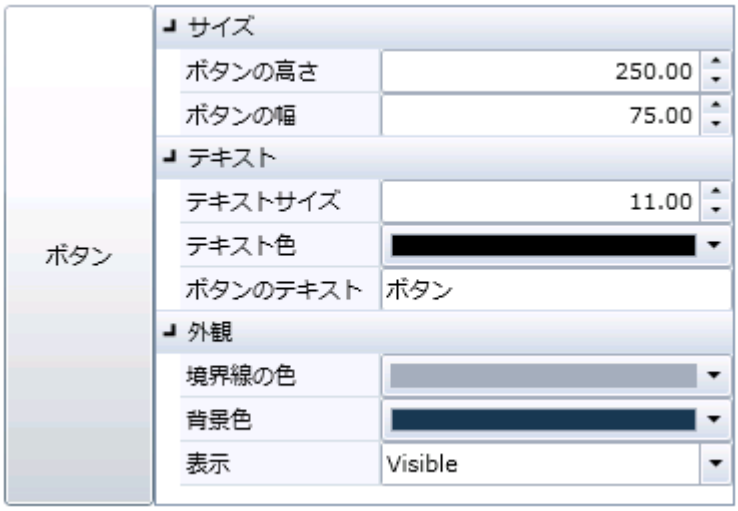
す。次の手順では、アプリケーションを実行し、実行時の操作をいくつか確認します。

手順 3:アプリケーションの実行

前の手順では、Microsoft Expression Blend で **PropertyGrid for Silverlight** を使って Silverlight アプリケーションを作成し、**Button** コントロールのプロパティを表示するように設定し、表示するプロパティをカスタマイズしました。この手順では、アプリケーションを実行し、**PropertyGrid for Silverlight** の実行時の操作をいくつか確認します。

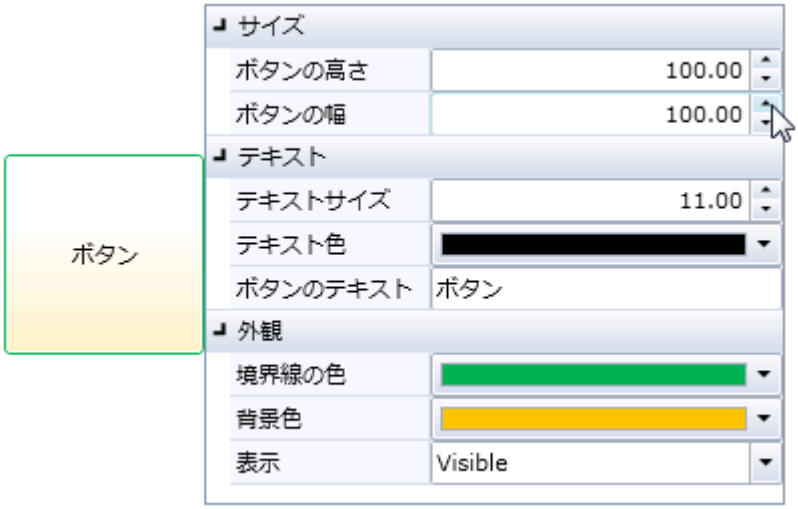
次の手順に従います。

1. Expression Blend で、**[プロジェクト]**→**[プロジェクトの実行]**を選択します。アプリケーションがデフォルトの Web ブラウザで開きます。指定したプロパティと入力した表示名がアプリケーションに表示されていることを確認します。

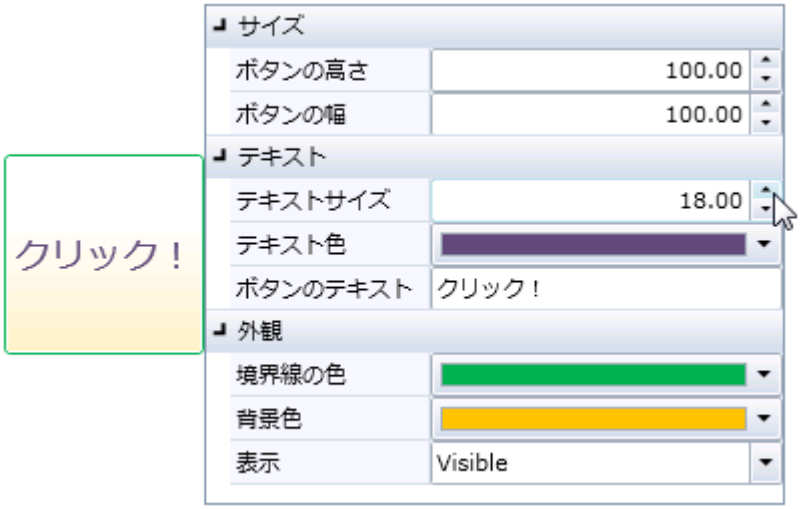


2. **[背景色]** のドロップダウン矢印をクリックし、表示されるカラーピッカーからオレンジなどの色を選択します。ボタンの背景色が選択した色に変わります。

3. **[境界線の色]** のドロップダウン矢印をクリックし、表示されるカラーピッカーから緑などの色を選択します。
4. **[ボタンの高さ]**および**[ボタンの幅]**数値ボックスに値を入力してボタンのサイズを変更します。たとえば、両方の値に**「100」**と入力します。アプリケーションは次の図のように表示されます。



5. **[ボタンテキスト]**ボックスに「クリック！」などの文字列を入力します。
6. **[テキスト色]**のドロップダウン矢印をクリックし、表示されるカラーピッカーから紫などの色を選択します。
7. **[テキストサイズ]**の隣にある**上向き**または**下向き**の矢印をクリックして、ボタンコントロールに表示されるテキストのサイズを変更します。たとえば、値を **18** に設定します。アプリケーションは次のようになります。



おめでとうございます!

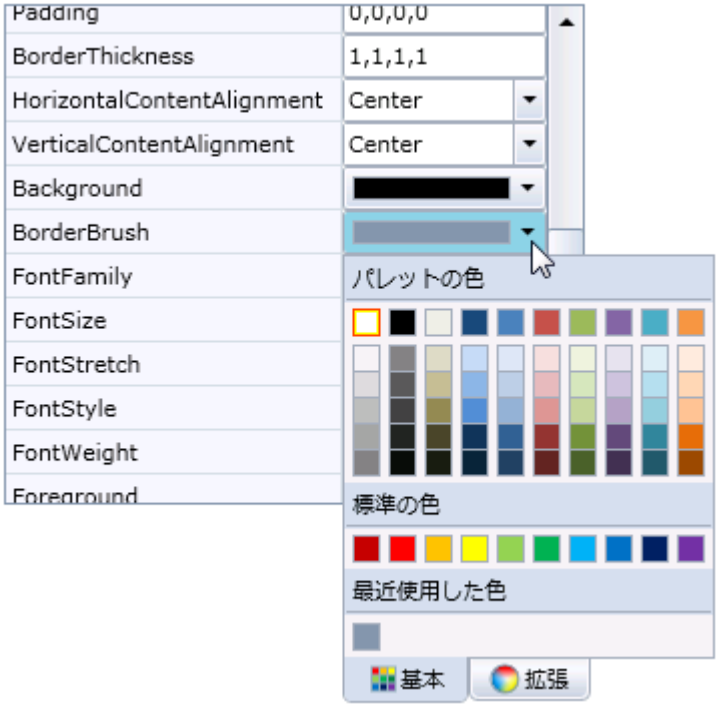
これで、**PropertyGrid for Silverlight** クイックスタートは終了です。このクイックスタートでは、ページに **C1PropertyGrid** コントロールと **Button** コントロールを追加し、**C1PropertyGrid** コントロールを **Button** にリンクしました。さらに、コントロールをカスタマイズし、**PropertyGrid for Silverlight** で実行時に可能な操作を見ました。

主な特長

PropertyGrid 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。次の主要な機能を利用して、PropertyGrid **for WPF** を最大限に活用してください。

- **使い慣れたプロパティウィンドウ**
  C1PropertyGrid コントロールは、プロパティを簡単に編集できる使い慣れたインターフェイスと組み込みエディタを提供します。

- **実行時のプロパティの編集**
  WPF アプリケーション設計の中に **C1PropertyGrid** コントロールを組み込み、これを使用して、エンドユーザーに直接プロパティを編集してもらうことができます。



- **動的な表示**
  C1PropertyGrid コントロールをフォームに追加し、プロパティを1つ設定するだけで、オブジェクトを編集するための UI が自動的に構築されます。

選択されたオブジェクト

SelectedObject プロパティは、どのオブジェクトのプロパティを **C1PropertyGrid** コントロールに表示して編集するかを決定します。SelectedObject プロパティは、任意のオブジェクトに設定できます。たとえば、「クイックスタート」のように、C1PropertyGrid コントロールを別のコントロールに接続したり、「クラスを連結する」のように、このコントロールをクラスに連結することができます。

XAML では、Binding ステートメントを使用して C1PropertyGrid コントロールをオブジェクトに接続します。たとえば、次の C1PropertyGrid コントロールはボタンオブジェクトにリンクされます。

XAML

```
<c1:C1PropertyGrid Margin="244,152,186,168" SelectedObject="{Binding
ElementName=button, Mode=OneWay}"/>
```

Blend のデザインビューで、[プロパティ]ウィンドウの SelectedObject 項目の隣にある四角い[高度なプロパティ]アイコンを選択し、**[データバインド]**を選択しても、**SelectedObject** プロパティを設定できます。**[データのバインドの作成]**ダイアログボックスが表示され、ここで連結するオブジェクトを選択できます。

メンバのソート

Visual Studio の[プロパティ]ウィンドウの[アルファベット]ビューと同様に、デフォルトでは、**C1PropertyGrid** コントロール内のプロパティとメソッドはアルファベット順に表示されます。ただし、**PropertySort** プロパティを設定すると、メンバのリスト方法をカスタマイズできます。C1PropertyGrid コントロールでは、次のいずれかの方法でプロパティをソートできます。

- **Alphabetical**:プロパティがアルファベット順にソートされます。これがデフォルト設定です。Visual Studio の[プロパティ]ウィンドウの[アルファベット]ビューと同様に表示されます。

- **Categorized**:カテゴリがアルファベット順に表示され、各カテゴリ内のプロパティは特定の順序でなく、SelectedObject から取得された順序で表示されます。
- **CategorizedAlphabetical**:カテゴリがアルファベット順に表示され、各カテゴリ内のプロパティもアルファベット順に表示されます。Visual Studio の[プロパティ]ウィンドウの[カテゴリ]ビューと同じです。
- **CategorizedCustom**:カテゴリがアルファベット順に表示され、各カテゴリ内のプロパティは、Display.Order 属性を使用してユーザーによって定義されたカスタムの順序で表示されます。
- **Custom**:プロパティは Display.Order 属性を使用してユーザーによって定義されたカスタムの順序で表示されます。
- **NoSort**:プロパティは SelectedObject から取得された順序で表示されます。プロパティグリッドのソート方法をカスタマイズするには、PropertySort プロパティを上のオプションのいずれかに設定します。

組み込みエディタ

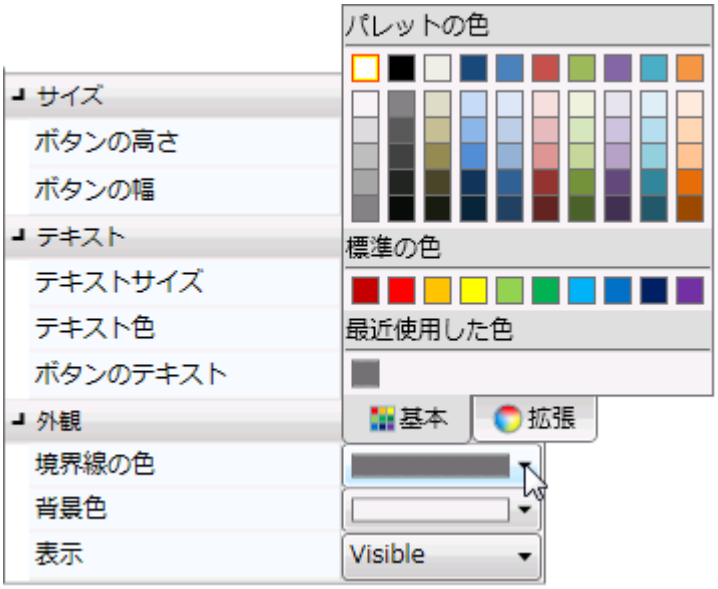
C1PropertyGrid コントロールには、いくつかの組み込みエディタがあります。エディタを指定しなかった場合、C1PropertyGrid には、各メンバのデフォルトのエディタが表示されます。エディタを指定したり、必要に応じてカスタムエディタを作成することもできます。AvailableEditors プロパティは、C1PropertyGrid コントロールで使用できるエディタを制御します。

C1PropertyGrid コントロールには、複雑なオブジェクト用のエディタは組み込まれていません。複雑なオブジェクトはデフォルトのエディタ(StringEditor)に表示され、そこに複雑なオブジェクトの文字列表現(tostring())が表示されます。複雑なオブジェクトを表示する場合は、カスタムエディタを作成することもできます。カスタムエディタの作成の詳細については、「カスタムエディタの作成」を参照してください。

次の組み込みエディタがあります。

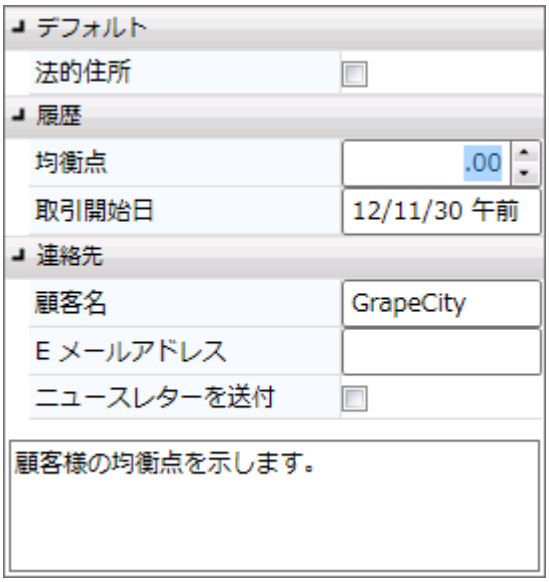
エディタ	説明
BoolEditor	ブール値を編集する際に C1PropertyGrid によって使用されるデフォルトのエディタ。
BrushEditor	ブラシ値を編集する際に C1PropertyGrid によって使用されるデフォルトのエディタ。
ColorEditor	色値を編集する際に C1PropertyGrid によって使用されるデフォルトのエディタ。
ColorPaletteEditor	カラーパレット値を編集する際に C1PropertyGrid によって使用されるデフォルトのエディタ。
EnumEditor	列挙型値を編集する際にC1PropertyGrid によって使用されるデフォルトのエディタ。
ImageSourceEditor	イメージ値を編集する際に C1PropertyGridによって使用されるデフォルトのエディタ。
NumericEditor	数値を編集する際に C1PropertyGrid によって使用されるデフォルトのエディタ。
StringEditor	文字列値を編集する際に C1PropertyGrid によって使用されるデフォルトのエディタ。
UriEditor	URI 値を編集する際に C1PropertyGrid によって使用されるデフォルトのエディタ。

たとえば、次の図では、実行時に色値を編集できます。



プロパティの説明の表示

デフォルトでは、**C1PropertyGrid** コントロールには、実行時にユーザーが編集できるプロパティの説明は表示されません。ただし、編集されるプロパティに関する情報を表示したい場合もあります。Visual Studio の**[プロパティ]**ウィンドウの[説明]ペインと同様に、**C1PropertyGrid** コントロールでは各プロパティの説明を表示できます。**ShowDescription** プロパティを True に設定すると、プロパティの説明を表示できます。これにより、**C1PropertyGrid** コントロールの下部に説明の領域が追加され、現在フォーカスがあるプロパティの説明が表示されます。説明は Display.Description 属性を使用して追加でき、次の図のように表示されます。



プロパティ値のリセット

C1PropertyGrid には、ユーザーが変更したプロパティを（DefaultValue 属性で指定される）デフォルト値にリセットするためのリセットボタンが用意されています。リセットボタンは、プロパティエディタの隣にある小さな四角形です。

デフォルトでは、**リセット**ボタンは表示されませんが、**ShowResetButton** プロパティを **True** に設定すると、**リセット**ボタンを表示できます。更新された値を反映するには、**SelectedObject** が **INotifyPropertyChanged** インターフェイスを実装している必要があります。連結されているプロパティが値の変更をエディタに通知して値を更新できるようにする必要があるためです。

サポートされている属性

属性を使用して、プロパティグリッドに表示される項目の表示方法とコンテンツをカスタマイズできます。**C1PropertyGrid** コントロールは、次のような属性をサポートします。

- **Display.Name**:各プロパティに表示されるテキストラベルを設定します。
- **Display.Order**:プロパティを表示する順序を指定します(カスタムソートを使用している場合)。
- **Display.Description**:プロパティの説明を設定します。この説明は、編集されているプロパティがフォーカスを得ると、**C1PropertyGrid** の下部に表示されます。
- **DefaultValue**:プロパティのデフォルト値を設定します。デフォルト値は、プロパティが他の値を持たない(初期化されていないか、null 値を持つ場合)場合、またはリセットボタンが押された場合に適用されます。
- **Editor**:現在のプロパティのカスタムエディタを設定します(デフォルトで割り当てられているエディタを上書きする)。
- **MaximumValue**:プロパティの最大値を設定します。**MaximumValue** は、数値エディタなど、意味を持つエディタでのみ考慮されます。
- **MinimumValue**:プロパティの最小値を設定します。**MinimumValue** は、数値エディタなど、意味を持つエディタでのみ考慮されます。
- **Browsable**:false に設定すると、プロパティは C1PropertyGrid に表示されません。
- **Category**:プロパティが表示されるカテゴリを設定します。
- **PropertyBinding**:プロパティとエディタとのデフォルトの連結を上書きします。たとえば、独自のコンバータを設定する場合や、プロパティのデフォルトの検証方法では不都合な場合に使用します。
- **DisplayFormat**:プロパティの値を表示する書式を指定します。DisplayFormat は、数値エディタなど、意味を持つエディタでのみ考慮されます。
- **ReadOnly**:True に設定すると、プロパティは **C1PropertyGrid** に表示されますが、値を変更することはできません。

添付プロパティの使用

C1PropertyGrid コントロールは、親コンテナ内で定義された要素に設定された添付プロパティをサポートします。

C1PropertyGrid コントロールの **AutoGenerateProperties** プロパティが true に設定されている場合、添付プロパティは親コ

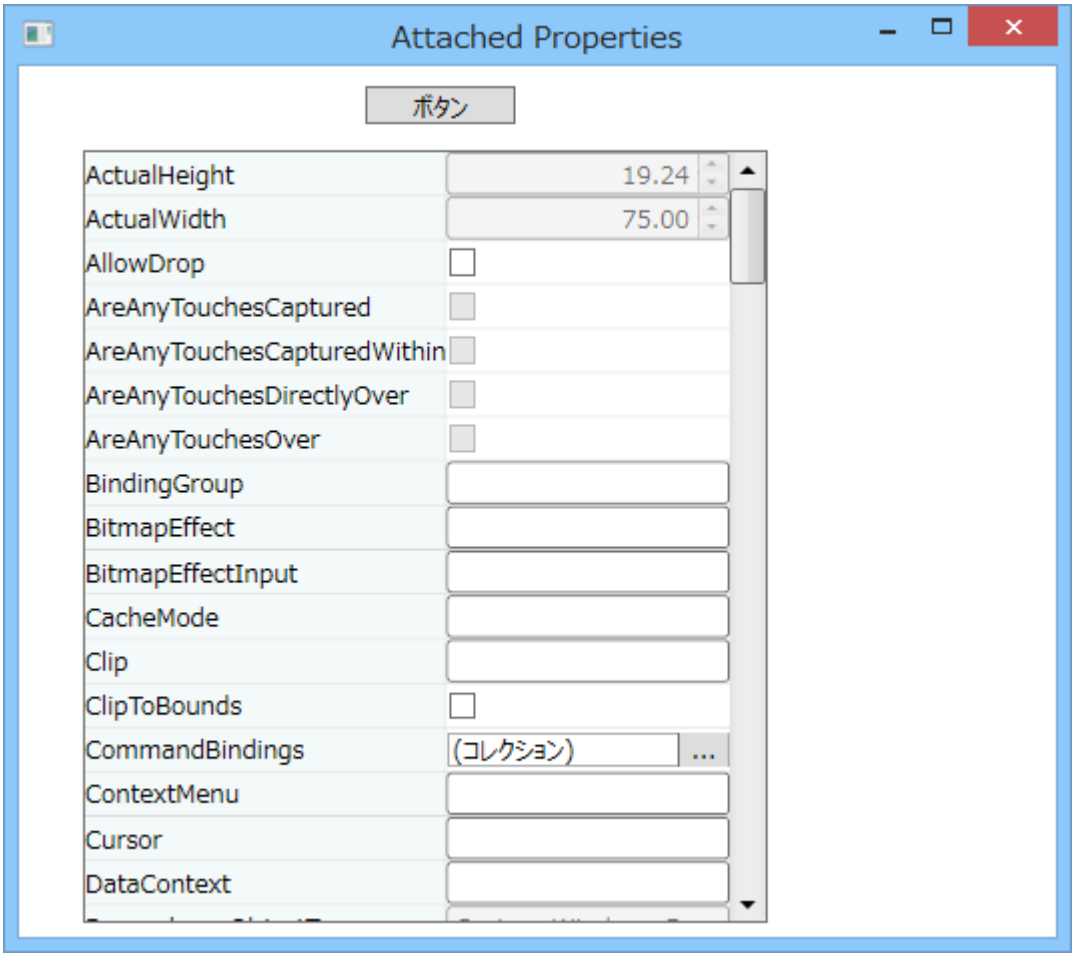
ントロールに依存します。たとえば、あるコントロールが Grid に挿入されている場合、プロパティグリッドには Grid の添付プロパティ（Grid.Row、Grid.Column など）が表示されます。親コントロールが Canvas の場合は、Canvas の添付プロパティ（Canvas.Left、Canvas.Top など）が表示されます。

次のコード例では、Canvas の添付プロパティである Canvas.Left プロパティを Button コントロールに割り当てる方法を示します。

XAML

```
<Canvas>
    <!--ボタンをプロパティグリッドに連結し、キャンバスに対するプロパティグリッドの相対位置を設定します-->
    <c1:C1PropertyGrid x:Name="c1PropertyGrid1" Height="386" Width="342"
                       SelectedObject="{Binding ElementName=Button1}"
                       Canvas.Left="32" Canvas.Top="42">
        <c1:C1PropertyGrid.PropertyAttributes>
            <!--ボタンによって継承される Canvas.Left 添付プロパティ-->
            <c1:PropertyAttribute Category="Geometry" Browsable="True"
DisplayName="Left" MemberName="Canvas.Left"
                                  MaximumValue="300" MinimumValue="20" />
        </c1:C1PropertyGrid.PropertyAttributes>
    </c1:C1PropertyGrid>
    <Button x:Name="Button1" Content="Button" Width="75"/>
</Canvas>
```

アプリケーションを実行し、Button コントロールのプロパティグリッドに Canvas の追加プロパティが表示されることを確認します。



レイアウトおよび外観

以下のトピックでは、**C1PropertyGrid** 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。テーマを使用することで、グリッドの外観をカスタマイズしたり、WPF/Silverlight の XAML ベースのスタイル設定を活用することができます。また、テンプレートを使用して、コントロールを書式設定およびレイアウトしたり、コントロールの操作をカスタマイズすることもできます。

ComponentOne ClearStyle 技術

ComponentOne 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 WPF/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。

ClearStyle プロパティ

次の表に、**C1PropertyGrid** コントロールの ClearStyle をサポートするプロパティと、それらの説明を一覧します。

プロパティ	説明
Background	コントロールの背景を描画するブラシを取得または設定します。デフォルトの **Background** 色は LightBlue です。
CategoryBackground	コントロールのカテゴリ背景の外観を定義するために使用されるブラシです。
CategoryForeground	コントロールのカテゴリ前景の外観を定義するために使用されるブラシです。
MouseOverBrush	マウスポインタが置かれているコントロールの外観を定義するために使用されるブラシです。

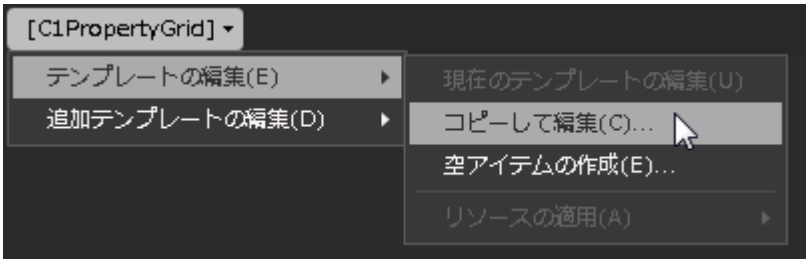
PressedBrush	押されているコントロールの外観を定義するために使用されるブラシです。

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。WPF 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、**PropertyGrid 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、C1PropertyGrid 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。

スタイル

PropertyGrid for WPF/Silverlight の **C1PropertyGrid** コントロールは、コントロールの外観を変更するために使用できるスタイルのプロパティを提供します。これらのスタイルの一部について、次の表で説明します。

スタイル	説明
CategoryContainerStyle	すべての生成されるカテゴリコンテナに適用される Style を取得または設定します。
FontStyle	フォントスタイルを取得または設定します。これは依存プロパティです。
LabelStyle	SelectedObject のプロパティ用に生成されるすべてのラベルに適用される Style を取得または設定します。
Style	この要素のレンダリング時に使用されるスタイルを取得または設定します。これは依存プロパティです。

テンプレートパーツ

Microsoft Expression Blend では、新しいテンプレートを作成して、テンプレートパーツを表示および編集できます。たとえば、**C1PropertyGrid** コントロールをクリックして選択し、**[オブジェクト]**→**[テンプレートの編集]**→**[コピーの編集]**を選択します。新しいテンプレートを作成すると、テンプレートのパーツが**[パーツ]**ウィンドウに表示されます。

[パーツ]ウィンドウにパーツを表示するには、**ControlTemplate** を選択する必要があります。**[パーツ]**ウィンドウで、任意の要素をダブルクリックすると、そのパーツをテンプレートに作成できます。

C1PropertyGrid コントロールでは、次のテンプレートパーツを使用きます。

名前	タイプ	説明
Description	ContentControl	コンテンツを1つ含むコントロールを表します。ここでは、コントロールの下部の説明領域を表します。
DescriptionArea	Border	別の要素の周囲に境界線、背景、またはその両方を描画します。ここでは、境界線はコントロールの下部の説明領域を囲みます。
Methods	ItemsControl	項目のコレクションを表すために使用できるコントロールを表します。ここでは、メソッドのコレクションを表します。
Properties	ItemsControl	項目のコレクションを表すために使用できるコントロールを表します。ここでは、プロパティのコレクションを表します。

表示状態

Microsoft Expression Blend で、カスタム状態や状態グループを追加して、ユーザーコントロールの状態ごとに異なる外観を定義できます。たとえば、マウスが置かれたときのコントロールの表示状態を変更できます。 新しいテンプレートを作成し、新しいテンプレートパーツを追加することで、表示状態を表示および編集できます。これで、そのパーツで利用可能な表示状態が**[表示状態]**ウィンドウに表示されます。

よく使用される状態としては、項目の通常の外観を示す **Normal**、マウスが置かれている項目を示す **MouseOver**、有効でない項目を示す **Disabled** などがあります。フォーカスの状態には、項目にフォーカスがないときの **Unfocused**、項目にフォーカスがあるときの **Focused** などがあります。

タスク別ヘルプ

次のタスク別ヘルプトピックは、ユーザーの皆様が Visual Studio および Expression Blend に精通しており、**C1PropertyGrid** コントロールの一般的な使用方法を理解していることを前提としています。**PropertyGrid for WPF/Silverlight** 製品を初めて使用される場合は、まず「PropertyGrid for WPF クイックスタートまたはPropertyGrid for Silverlight 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、**PropertyGrid for WPF/Silverlight** 製品を使用して特定のタスクを行うための方法を提供します。また、ほとんどのタスク別ヘルプトピックは、新しい WPF プロジェクトが作成されており、そのプロジェクトに **C1PropertyGrid** コントロールが追加されていることを前提としています。

クラスを連結する

PropertyGrid for WPF/Silverlight を使用すると、このコントロールをクラスに簡単に連結できます。実行時は、**C1PropertyGrid** コントロールを使用して、クラス内の項目を参照および編集できます。たとえば、次のように定義されたプロジェクトに単純な **Customer** クラスを追加します。

VisualBasic

```
Private _Name As String
    Public Property Name() As String
        Get
            Return _Name
        End Get
       Set(ByVal value As String)
```

```
            _Name = value
        End Set
    End Property
Private _EMail As String
    Public Property EMail() As String
        Get
            Return _EMail
        End Get
        Set(ByVal value As String)
            _EMail = value
        End Set
    End Property
Private _Address As String
    Public Property Address() As String
        Get
            Return _Address
        End Get
        Set(ByVal value As String)
            _Address = value
        End Set
    End Property
Private _CustomerSince As DateTime
    Public Property CustomerSince() As DateTime
        Get
            Return _CustomerSince
        End Get
        Set(ByVal value As DateTime)
            _CustomerSince = value
        End Set
    End Property
Private _SendNewsletter As Boolean
    Public Property SendNewsletter() As Boolean
        Get
            Return _SendNewsletter
        End Get
        Set(ByVal value As Boolean)
            _SendNewsletter = value
        End Set
    End Property
Private _PointBalance As System.Nullable(Of Integer)
    Public Property PointBalance() As System.Nullable(Of Integer)
        Get
            Return _PointBalance
        End Get
        Set(ByVal value As System.Nullable(Of Integer))
            _PointBalance = value
        End Set
    End Property
End Class
```

C#

```
public class Customer
{
  public string Name { get; set; }
  public string EMail { get; set; }
  public string Address { get; set; }
  public DateTime CustomerSince { get; set; }
  public bool SendNewsletter { get; set; }
  public int? PointBalance { get; set; }
}
```

次のコードを使用して、顧客を表示および編集するためのユーザーインターフェイスを作成できます。

VisualBasic

```
Public Sub New()
    InitializeComponent()
        ' 参照するオブジェクトを作成します
    Dim customer = New Customer()
        ' C1PropertyGrid を作成します
    Dim pg = New C1PropertyGrid()
    LayoutRoot.Children.Add(pg)
        ' 顧客のプロパティを表示します
    pg.SelectedObject = customer
End Sub
```

C#

```
public Page()
{
  InitializeComponent();
  // 参照するオブジェクトを作成します
  var customer = new Customer();
  // C1PropertyGrid を作成します
  var pg = new C1PropertyGrid();
  LayoutRoot.Children.Add(pg);
  // 顧客のプロパティを表示します
  pg.SelectedObject = customer;
}
```

アプリケーションを実行し、結果のアプリケーションが次の図のように表示されていることを確認します。

名前	田中太郎
Eメールアドレス	田中@gmail.com
住所	123 東京
取引開始日	2001/12/12 0:00:00
ニュースの送付	☑
均衡点	12

この簡単な UI を使用して、ユーザーは **Customer** オブジェクトのすべてのプロパティを編集できます。この UI は、オブジェク

トのプロパティに基づいて自動的に作成され、**Customer** クラスのプロパティが追加または変更されると自動的に更新されます。

デフォルトでは、プロパティはアルファベット順で表示されます。この設定は、**PropertySort** プロパティを設定することで変更できます。詳細については、「メンバのソート」を参照してください。

レイアウトをカスタマイズする

このコントロールでカスタマイズできる最初の要素は、レイアウトです。コントロールには、ラベルから成る列とエディタから成る列の2つの列があります。これらの列のサイズはデフォルトでは同じですが、**LabelWidth** プロパティと **EditorWidth** プロパティの値を変更すると、サイズを変更できます。

たとえば、1行のコードを追加して、上の例のラベル列を狭くすることができます。

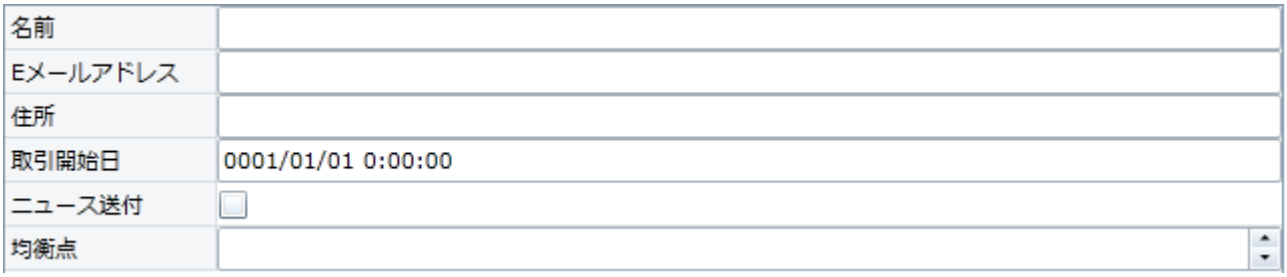
VisualBasic

```
Public Sub New()
    InitializeComponent()
        ' 参照するオブジェクトを作成します
    Dim customer = New Customer()
        ' C1PropertyGrid を作成します
    Dim pg = New C1PropertyGrid()
    LayoutRoot.Children.Add(pg)
          ' PropertyGrid レイアウトをカスタマイズします
          pg.LabelWidth = 100
        ' 顧客のプロパティを表示します
    pg.SelectedObject = customer
End Sub
```

C#

```
public Page()
{
  InitializeComponent();
  // 参照するオブジェクトを作成します
  var customer = new Customer();
  // C1PropertyGrid を作成します
  var pg = new C1PropertyGrid();
  LayoutRoot.Children.Add(pg);
        // PropertyGrid レイアウトをカスタマイズします
        pg.LabelWidth = 100;
  // 顧客のプロパティを表示します
  pg.SelectedObject = customer;
}
```

結果は次のようになります。



ご覧のように、ラベル列が狭くなり、エディタ部分の領域が広がりました。フォームのサイズを変更しても、ラベル列の幅は変わりません。

表示名をカスタマイズする

デフォルトでは、各プロパティの横に表示されるラベルには、プロパティ名が表示されます。多くの場合はこれで十分ですが、よりわかりやすい名前にするために、表示名をカスタマイズすることもできます。最も簡単な方法は、カスタム属性でオブジェクトのプロパティを修飾することです。**Display** 属性の **Name** プロパティを設定します（Display 属性は、System.ComponentModel,DataAnnotations アセンブリの System.ComponentModel.DataAnnotations 名前空間で定義されています）。

たとえば、次のコードのように、クラスで **Display** 属性を定義し、**Name** プロパティの値を設定できます。

VisualBasic

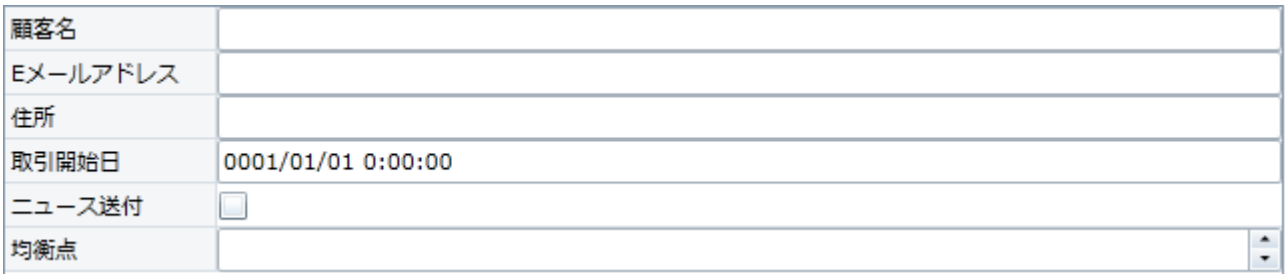
```
Public Class Customer
Private _Name As String
    <Display(Name:="顧客名")> _
    Public Property Name() As String
        Get
            Return _Name
        End Get
        Set(ByVal value As String)
            _Name = value
        End Set
    End Property
    Private _EMail As String
    <Display(Name:="E メールアドレス")> _
    Public Property EMail() As String
        Get
            Return _EMail
        End Get
        Set(ByVal value As String)
            _EMail = value
        End Set
    End Property
    Private _Address As String
    <Display(Name:="住所")> _
    Public Property Address() As String
        Get
            Return _Address
        End Get
        Set(ByVal value As String)
            _Address = value
```

```
        End Set
    End Property
    Private _CustomerSince As DateTime
    <Display(Name:="取引開始日")> _
    Public Property CustomerSince() As DateTime
        Get
            Return _CustomerSince
        End Get
        Set(ByVal value As DateTime)
            _CustomerSince = value
        End Set
    End Property
  Private _SendNewsletter As Boolean
    <Display(Name:="ニュース送付")> _
    Public Property SendNewsletter() As Boolean
        Get
            Return _SendNewsletter
        End Get
        Set(ByVal value As Boolean)
            _SendNewsletter = value
        End Set
    End Property
  Private _PointBalance As System.Nullable(Of Integer)
    <Display(Name:="均衡点")> _
    Public Property PointBalance() As System.Nullable(Of Integer)
        Get
            Return _PointBalance
        End Get
        Set(ByVal value As System.Nullable(Of Integer))
            _PointBalance = value
        End Set
    End Property
End Class
```

C#

```
public class Customer
{
  [Display(Name = "顧客名")]
  public string Name { get; set; }
  [Display(Name = "E メールアドレス")]
  public string EMail { get; set; }
  public string Address { get; set; }
  [Display(Name = "取引開始日")]
  public DateTime CustomerSince { get; set; }
  [Display(Name ="ニュース送付")]
  public bool SendNewsletter { get; set; }
  [Display(Name ="均衡点")]
  public int? PointBalance { get; set; }
}
```

C1PropertyGrid は、この追加情報を使用して、次のように顧客を表示します。



この方法には、**C1PropertyGrid** に表示されるクラスへのアクセス権が必要です。表示文字列を変更しようとしても、表示されているクラスを変更できない場合は、**PropertyAttributes** プロパティを使用して、**C1PropertyGrid** に表示する各プロパティについて明示的に情報を提供する必要があります。

カスタムエディタの作成

組み込みエディタが目的のユーザーインターフェイスの設計には適さない場合、カスタムエディタを簡単に作成でき、**C1PropertyGrid**コントロールに使用できます。
次の手順に従います。

1. **ITypeEditorControl** インターフェイスを実装するクラスを作成します。
2. このクラスのインスタンスをC1PropertyGrid コントロールのAvailableEditors のコレクションに追加します。

または

プロパティの定義に**EditorAttribute** クラスを追加して、このクラスを特定なプロパティで使用するエディタとして指定します。

ITypeEditorControl インターフェイスで公開されるメンバを以下に示します。

- **bool Supports(PropertyAttribute プロパティ)**
  PropertyAttribute で記述されたプロパティ値をエディタがサポートするかどうかを判断します。
- **void Attach(PropertyAttribute プロパティ)**
  現在のプロパティにエディタを初期化します。
- **void Detach(PropertyAttribute プロパティ)**
  エディタのインスタンスを解放します。
- **ITypeEditorControl Create()**
  エディタの新しいインスタンスを作成して初期化します。
- **event PropertyChangedEventHandler ValueChanged**
  現在のプロパティの値が変化すると発生します。このイベントは検証するエディタによって使用されます。

カスタムエディタの詳細については、**WPF Edition**にインストールしている**ControlExplorer**サンプルをご参照してください。

Rating（WPF のみ）

Rating for WPF を使用すると、本、映画、オンラインブログ、レシピなどの Web コンテンツをレーティングするために、カスタマイズ可能なレーティングコントロールを導入できます。クリックするだけで正確にレーティングを選択できるため、評価を示す新しい方法をユーザーに提供できます。

標準的なレーティングコントロールとは対照的に、**Rating for WPF** には、アニメーションの追加、アニメーションのカスタマイズ、レーティングアイコンのカスタマイズ、コントロールの方向や向きの変更など、さまざまな機能があります。

メモ:このセクションの内容は、ComponentOne for WPF にのみ適用されます。

主な特長

アプリケーションのユーザーインタフェースの設計に柔軟性をもたらす Rating コントロールの主要な機能を以下にいくつか示します。

- **アニメーション**: レーティングコントロールにアニメーションを追加して、目を引く動きを付けることができます。
- **方向**: レーティングコントロールをページの左から右に読み取るか、右から左に読み取るかを指定します。
- **向き**: レーティングコントロールをページの水平方向に配置するか、垂直方向に配置するかを指定します。
- **カスタムアイコンの使用**:星やサムアップといったさまざまなアイコンを使用して、レーティングコントロールの見栄えを向上させます。

Rating for WPF クイックスタート

このクイックスタートセクションでは、最初に本をレーティングするための WPF アプリケーションを作成します。以下の手順では、プロジェクトに Rating コントロールを追加し、設計時にコントロールの外観をカスタマイズし、実行時にそれを使用して、コントロールの外観と動作を確認します。

手順1:アプリケーションへの Rating コントロールの追加

クイックスタートセクションの最初の手順では、WPF プロジェクトを作成し、それにレーティングコントロールを追加するだけです。

1. Visual Studio で WPF プロジェクトを作成します。
2. **[デザイン]**タブをクリックして、デザイナを表示します。
3. ツールボックスに移動し、**C1Rating** アイコンを見つけます。
4. **C1Rating** アイコンをダブルクリックして、メインウィンドウに追加します。

次の手順では、設計時にレーティングコントロールの外観をカスタマイズし、レーティング対象の画像を追加します。

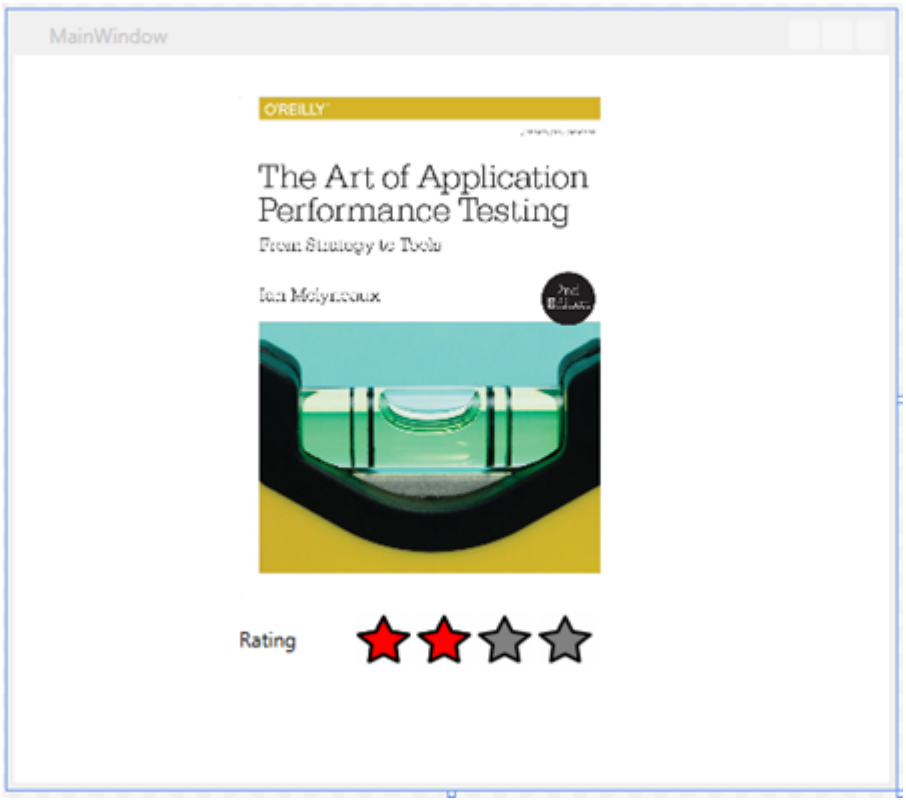
手順2:Rating コントロールの外観のカスタマイズ

前の手順では、WPF プロジェクトを作成し、それに Rating コントロールを追加しました。この手順では、設計時に **C1Rating** クラスのいくつかのプロパティを使用して、Rating コントロールの外観をカスタマイズすることから始めます。この例では本をレーティングし、本の画像をプロジェクトに追加します。

1. メインウィンドウでレーティングコントロールを選択し、**[プロパティ]**ペインに移動します。
2. **AutoGeneratedItemsCount** プロパティを **4** に設定して、アプリケーションの実行時にコントロールに合計 4 つの星が表示されるようにします。
3. **RatingPrecision** プロパティを **Half** に設定して、星 0.5 個分のレーティングを許可します。
4. **HoveredBrush** プロパティを **Green** に設定して、マウスポイント状態のブラシの色を設定します。デフォルトでは、

HoveredBrush プロパティは **Yellow** に設定されています。

5. **Value** プロパティを **2** に設定して、レーティングの初期値として星 2 個を表示します。
6. レーティングされる本の表紙を表示する画像をプロジェクトに追加します。
7. 再度ツールボックスに移動し、デザイナに標準的な TextBlock コントロールを追加します。それを画像の下に配置します。
8. **[プロパティ]**ウィンドウで TextBlock の **Text** プロパティを「Rating」に設定します。
9. C1Rating コントロールを画像の下の TextBlock の横に配置します。**デザイン**ビューは、次の図のように表示されます。

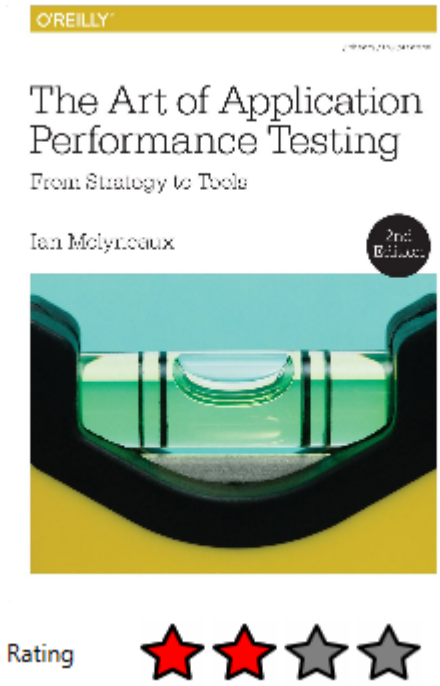


これで、Rating コントロールを追加し、その外観をカスタマイズできました。次の手順では、アプリケーションを実行して Rating コントロールの外観と動作を確認します。

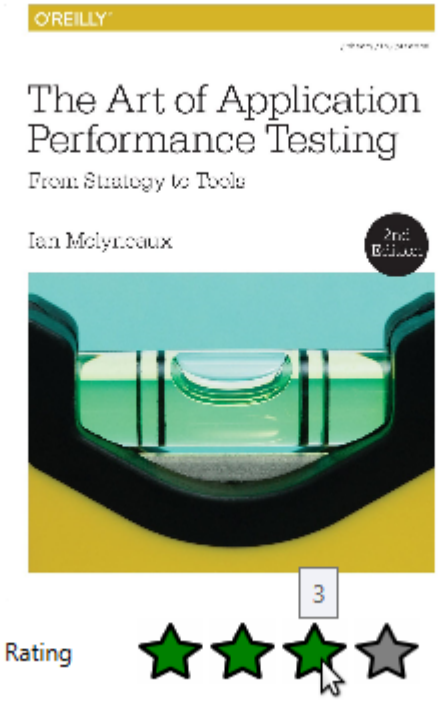
手順3：設計時に Rating コントロールの使用

前の手順では、設計時に Rating コントロールのいくつかのプロパティを設定して外観をカスタマイズしました。この手順では、アプリケーションを実行してコントロールの外観と動作を確認します。

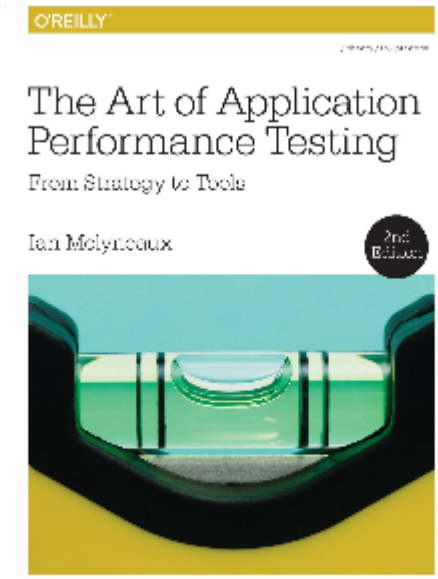
1. **[F5]**キーを押してアプリケーションをビルドします。出力ウィンドウは、次の図のように表示されます。



2. **手順 2** で行ったカスタマイズを反映して、レーティングとして星 2 個が表示されることがわかります。
3. 実行時に Rating コントロールにマウスポインタが置かれると、コントロールに表示される星の色が緑になります。



4. **RatingPrecision** プロパティを **Half** に設定したので、レーティングとして 0.5 個分の星を与えることもできます。



これで、クイックスタートセクションのすべての手順を完了しました。

機能

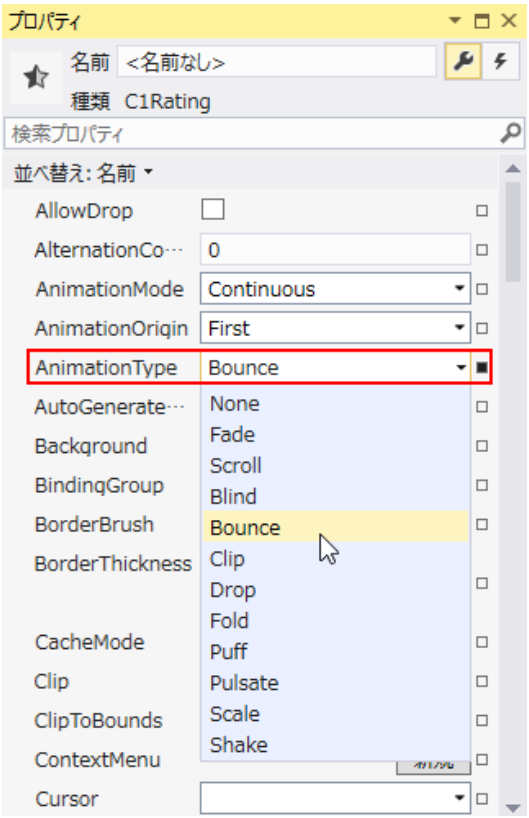
このセクションでは、**Rating for WPF** に関連する重要な機能、プロパティなどの項目について説明します。このセクションは、Visual Studio のプログラミング環境にある程度慣れていることを前提としています。このセクションで説明する機能を実装すると、**C1Rating** クラスのさまざまなプロパティを実演する動的なアプリケーションを作成し、その機能を理解できます。

C1Rating コントロールにアニメーションの追加

Rating for WPF は、**C1Rating** クラスの **AnimationType** プロパティを使用して、Rating コントロールにアニメーション効果を追加する機能を提供します。

設計時の場合

1. デザインビューで、メインウィンドウの Rating コントロールを 1 回クリックして選択します。
2. [プロパティ]ウィンドウに移動し、**AnimationType** プロパティを見つけます。
3. ドロップダウン矢印をクリックし、Rating コントロールに適用するアニメーション効果を選択します。デフォルトでは、**AnimationType** プロパティは **None** に設定されています。この例では、**Bounce** を選択して、コントロールにバウンド効果を適用します。



4. **[F5]**キーを押してプログラムを実行します。星をクリックしてアニメーション効果を確認します。

コードの場合

Rating コントロールにアニメーション効果を追加するには、<c1:C1Rating> タグ間に次のマークアップを追加します。

XAML copyCode

```
<c1:C1Rating x:Name="Rating1" HorizontalAlignment="Left" VerticalAlignment="Top" AnimationType="Bounce"/>
```

Rating コントロールのためアニメーションのカスタマイズ

Rating for WPF は、**IAnimationFactory** クラスのプロパティとメソッドを継承して、アニメーション効果をカスタマイズするオプションを提供します。デフォルトのアニメーション効果コレクションを利用できるほか、Rating コントロールに適用するアニメーション効果をカスタマイズすることができます。

コードの場合

1. XAML の操作ロジックに次のコードを追加して、CustomAnimationFlickerFactory クラスのインスタンスを作成します。

C# copyCode

```
private IAnimationFactory customAnimationFlickerFactory = new CustomAnimationFlickerFactory();

public MainWindow()
{
    InitializeComponent();
    c1Rating.AnimationFactory = customAnimationFlickerFactory;
}
```

2. 次のコードを CustomAnimationFlickerFactory.cs ファイルに追加して、IAnimationFactory クラスをオーバーライドします。

C# copyCode

```
using C1.WPF.Extended;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
```

```
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Media.Media3D;

namespace Rating_WPF_Animation
{
    class CustomAnimationFlickerFactory : IAnimationFactory
    {
        public Storyboard GetForwardAnimation(FrameworkElement target, AnimationType animationType)
        {
            return GetFadeAnimation(target, true);
        }
        public Storyboard GetBackwardAnimation(FrameworkElement target, AnimationType animationType)
        {
            return GetFadeAnimation(target, false);
        }
        private Storyboard GetFadeAnimation(FrameworkElement target, bool forward)
        {
            RotateTransform st = new RotateTransform();
            st.Angle = 360;
            Point p = new Point(0.5, 0.5);
            target.RenderTransformOrigin = p;
            target.RenderTransform = st;
            Storyboard sb = new Storyboard();
            var duration = TimeSpan.FromSeconds(0.25);
            EasingFunctionBase easing = new ElasticEase()
            {
                EasingMode = EasingMode.EaseInOut,
                Oscillations = 20,
                Springiness = 5
            };
            DoubleAnimation da1 = new DoubleAnimation();
            DoubleAnimation da2 = new DoubleAnimation();

            if (forward)
            {
                da1.From = 1.0;
                da1.To = 0.0;
                da2.To = 1.0;
            }
            else
            {
                da1.From = 0.0;
                da1.To = 1.0;
            }
            da1.Duration = duration;
            da2.Duration = duration;

            da1.EasingFunction = easing;
            da2.EasingFunction = easing;

            sb.Children.Add(da1);
            sb.Children.Add(da2);

            Storyboard.SetTargetProperty(da2, new PropertyPath("RenderTransform.Angle"));
            Storyboard.SetTargetProperty(da1, new PropertyPath("Opacity"));
            return sb;
        }
    }
}
```

3. **[F5]**キーを押してアプリケーションを実行し、カスタマイズしたちらつき効果が Rating コントロールに適用されたことに注目してください。

方向と向きの変更

Rating for WPF には、コントロールのレーティング方向（左から右または右から左）とコントロールが表示される向き（垂直または水平）をカスタマイズするオプションがあります。

設計時の場合

次の手順を実行して、方向をカスタマイズします。

1. デザインビューで、Rating コントロールを選択し、**FlowDirection** プロパティを RightToLeft に設定します。
2. **[F5]**キーを押してアプリケーションを実行します。次の図のように、コントロールのレーティング方向が右から左になることがわかります。

次の手順を実行して、向きをカスタマイズします。

1. **[プロパティ]**ペインで、Rating コントロールの **Orientation** プロパティを **Vertical** に設定します。
2. **[F5]**キーを押してアプリケーションを実行します。コントロールが垂直に表示されることがわかります。

ソースビューの場合

Rating コントロールの方向と向きをカスタマイズするには、コードによって FlowDirection プロパティを **RightToLeft** に設定し、**Orientation** プロパティを Vertical に設定します。コードは、次のようになります。

XAML copyCode

```
<c1:C1Rating HorizontalAlignment="Left" VerticalAlignment="Top" Margin="56,33,0,0"
             FlowDirection="RightToLeft" RatingPrecision="Precise" Orientation="Vertical"/>
```

アイコンのカスタマイズ

Rating for WPF を使用すると、サムアップなどの別の項目テンプレートを使用して、Rating コントロールの外観をカスタマイズできます。

サムアップアイコンを使用して Rating コントロールを表示するには、<Grid> タグ間に次のマークアップを追加します。

XAML copyCode

```
<c1:C1Rating HorizontalAlignment="Left" VerticalAlignment="Top" Margin="56,33,0,0"
             FlowDirection="RightToLeft" RatingPrecision="Precise" Orientation="Vertical"/>
```

[F5]キーを押してプログラムを実行します。次の図のように、サムアップレーティングアイコンがメインウィンドウに表示されます。

Reflector (Silverlight のみ)

Reflector for Silverlight により、テキストや UI 要素の 2D または 3D リフレクションを表示できます。付属する不透明効果または Silverlight の標準効果（ぼかし、ドロップシャドウなど）を使用して、リフレクションの外観を変更します。

メモ:このセクションの内容は、ComponentOne for Silverlight にのみ適用されます。

主な特長

Reflector for Silverlight を使用して、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。以下の主な特長をうまく利用して、**Reflector for Silverlight** を最大限に活用してください。

- **3D リフレクション**
  Reflector for Silverlight は、Silverlight の3面投影をサポートします。リフレクションを生成するには、**ReflectionEffects** コレクションに効果を追加します。**C1Reflector** には不透明効果が付属しますが、ぼかし、ドロップシャドウなどの Silverlight の標準効果を使ってリフレクションに適用することもできます。
- **任意の UI 要素の使用**
  Reflector for Silverlight は、コンテンツとして任意の UIElement をサポートします。複数の UIElement を追加するには、グリッドなどの外側のコンテナ要素を1つ追加し、そこに多数のサブ要素を含めます。
- **リフレクションの外観のカスタマイズ**
  不透明効果とぼかし効果を適用して設定します。たとえば不透明効果では、コンテンツ全体の不透明効果の速度および開始値を設定できます。
- **自動更新**
  コンテンツが変更されると、リフレクションは自動的に更新されます。

クイックスタート

このクイックスタートは、**Reflector for 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、最初に 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 **C1Reflector** コントロールを追加し、コンテンツを **C1Reflector** コントロールに追加し、**C1Reflector** コントロールの外観およびそのコンテンツをカスタマイズします。

手順 1:アプリケーションの作成

この手順では、最初に Visual Studio で **Reflector for Silverlight** を使用する Silverlight アプリケーションを作成します。

次の手順に従います。

1. Visual Studio で、[ファイル]→[新しいプロジェクト]を選択します。
2. [新しいプロジェクト]ダイアログボックスの左ペインから言語を選択し、テンプレートリストから[Silverlight アプリケーション]を選択します。プロジェクトの名前を入力し、[OK]をクリックします。[新しい Silverlight アプリケーション]ダイアログボックスが表示されます。
3. [OK]をクリックすると、[新しい Silverlight アプリケーション]ダイアログボックスが閉じ、プロジェクトが作成されます。
4. プロジェクトの XAML ウィンドウで、<UserControl> タグの DesignWidth="400" DesignHeight="300" を DesignWidth="Auto" DesignHeight="Auto" に変更して、UserControl をサイズ変更します。次のようになります。

XAML

```
<UserControl x:Class="C1Reflector.MainPage"
```

```
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"
mc:Ignorable="d" d:DesignWidth="Auto" d:DesignHeight="Auto">
```

これで、**UserControl** は、中に置かれた内容に合わせてサイズ変更されるようになります。

5. プロジェクトの XAML ウィンドウで、カーソルを <Grid7gt; タグと </Grid> タグの間に置き、1回クリックします。
6. ツールボックスに移動し、C1Reflector アイコンをダブルクリックして、コントロールをグリッドに追加します。XAML は次のようになります。

XAML

```
<Grid x:Name="LayoutRoot">
<c1:C1Reflector></c1:C1Reflector>
</Grid>
```

この状態でプロジェクトを実行すると、**C1Reflector** コントロールには今からコンテンツを追加する必要があるので、空白のページが表示されます。これについては、次の手順で説明します。

これで、**C1Reflector** コントロールを含む Silverlight アプリケーションを作成できました。次の手順では、**C1Reflector** コントロールに2つのコントロールを追加します。

手順 2：コンテンツの追加

クイックスタートチュートリアルのこのセクションでは、**C1Reflector** コントロールに2つのコントロールを追加しますが、**Content** プロパティは一度に1つのコントロールしか受け取ることができません。この問題は、**C1Reflector** コントロールのコンテンツとしてパネルベースのコントロールを追加し、パネルに複数のコントロールを重ねて追加することによって回避できます。**C1Reflector** コントロールは、パネルに追加されるすべてのコントロールのリフレクションを作成します。

次の手順に従います。

1. カーソルを <c1:C1Reflector> タグと </c1:C1Reflector> タグの間に置き、[Enter]キーを押します。
2. ツールボックスに移動し、**[StackPanel]**アイコンをダブルクリックして、**MainPage.xaml** にパネルを追加します。
3. カーソルを <StackPanel> タグと </StackPanel> タグの間に置き、[Enter]キーを押します。
4. ツールボックスに移動し、**[C1DateTimePicker]**アイコンをダブルクリックして、**StackPanel** コントロールに **C1DateTimePicker** コントロールを追加します。C1DateTimePicker コントロールは対話式なので、C1Reflector コントロールの自動更新機能を確認できます。
5. <c1:C1DateTimePicker> タグに Width="190" を追加し、コントロールの幅を設定します。
6. <c1:C1DateTimePicker> タグの後にカーソルを置き、[Enter]キーを押します。
7. ツールボックスに移動し、**[TextBox]**アイコンをダブルクリックして、**C1DateTimePicker** コントロールの下に TextBox コントロールを追加します。
8. <TextBox> タグに Text="日時を変更するとリフレクションも変更されます" を追加して、コントロールにテキストを追加します。
9. <TextBox> タグに FontSize="18" を追加して、フォントのサイズを変更します。

これで、**C1Reflector** コントロールにカスタムコンテンツが追加されました。次の手順では、Blend を使って **C1Reflector** コントロールの外観を変更します。

手順 3:コントロールの設定

このクイックスタートの最初の2つの手順では、Silverlight プロジェクトを作成し、**C1Reflector** コントロールを追加し、**C1Reflector** コントロールにコンテンツを追加しました。この時点で、プロジェクトは動作します。これを実行すると、2次元の単色リフレクションが表示されます。ただし、これで完成ではなく、設定することによってプロジェクトの外観をさらに向上させることができるプロパティがあります。この手順では、**Expression Blend** を使ってリフレクションにぼかし効果と不透明効果を追加し、コントロールを3次元平面に配置します。

次の手順に従います。

1. Solution Explorer で **MainPage.xaml** を右クリックしてコンテキストメニューを開き、**[Expression Blend で開く]**を選択します。Expression Blend でプロジェクトが開きます。
2. [オブジェクトとタイムライン]タブで C1Reflector を選択して、[プロパティ]ウィンドウにプロパティを開きます。
3. 次の手順に従って、リフレクションに不透明効果とぼかし効果を追加します。
    a. **ReflectionEffects** の省略符ボタンをクリックし、**[Effect コレクション エディター: ReflectionEffects]**ダイアログボックスを開きます。
    b. [別のアイテムを追加]をクリックします。**[オブジェクトの選択]**ダイアログボックスが開きます。
    c. リストから ReflectionOpacityEffect を選択し、**[OK]**をクリックすると、効果がコントロールに追加され、**[Effect コレクションエディター: ReflectionEffects]**ダイアログボックスに戻ります。
    d. **[プロパティ]**グリッドで、次のプロパティを設定します。
    Coefficient プロパティを「1」に設定します。
    Offset プロパティを「0.5」に設定します。
    e. [別のアイテムを追加]をクリックします。 **[オブジェクトの選択]**ダイアログボックスが開きます。
    f. リストから **BlurEffect** を選択し、**[OK]**をクリックすると、効果がコントロールに追加され、**[Effect コレクション エディター: ReflectionEffects]**ダイアログボックスに戻ります。
    g. **[プロパティ]**グリッドで、**Radius** プロパティを「2」に設定します。数値が低いため、コントロールの周囲にソフトぼかし効果が適用されます。
    h. **[OK]**をクリックして、**[Effect コレクションエディター: ReflectionEffects]**ダイアログボックスを閉じます。
4. **C1Reflector** コントロールの面を変更するには、**ContentProjection** プロパティを見つけ、**[Rotation]**タブで次の手順に従います。
    a. [X]テキストボックスに「25」と入力します。
    b. [Y]テキストボックスに「45」と入力します。
    c. [Z]テキストボックスに「1」と入力します。

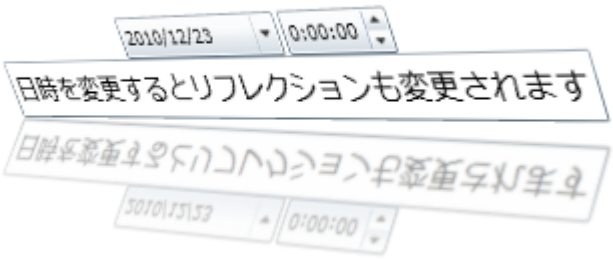
この手順では、プロパティの設定によって **C1Reflector** コントロールおよびそのコンテンツの外観を変更しました。次の手順では、プロジェクトを実行します。

手順 4:アプリケーションの実行

このクイックスタートの最初の3つの手順では、**C1Reflector** コントロールを含む Silverlight アプリケーションを作成し、**C1Reflector** コントロールにコンテンツを追加し、**C1Reflector** コントロールの外観を変更しました。この手順では、プロジェクトを実行し、各手順の結果を確認します。

次の手順に従います。

1. **[F5]**キーを押して Silverlight プロジェクトを実行します。アプリケーションは、次の図のように表示されます。



2. 日付のドロップダウンをクリックし、カレンダーから別の日付を選択します。この例では、2009 年9月 22 日を選択します。日付のリフレクションが 9/22/09 に変化することを確認します。これが C1Reflector コントロールの自動更新機能の効果です。

3. 時刻ピッカーの矢印キーを使って時刻を変更します。時間と分をスクロールすると、それぞれの変更がリアルタイムで反映されることを確認します。これも C1Reflector コントロールの自動更新機能の効果です。

おめでとうございます!

これで、**Reflector for Silverlight** クイックスタートは終了です。このクイックスタートでは、**Reflector for Silverlight** のアプリケーションを作成し、コンテンツを **C1Reflector** コントロールに追加し、**C1Reflector** コントロールの外観をカスタマイズし、**C1Reflector** コントロールの実行時機能をいくつか確認しました。

Reflector の使い方

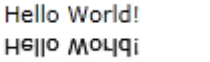
Reflector for Silverlight には、テキストと Silverlight コントロールの 2D と 3D のリフレクションを作成するために使用できる **C1Reflector** というコンテンツコントロールがあります。これをプロジェクトに追加する場合は、投影するコンテンツを追加する必要があります。これは、Blend で簡単なドラッグアンドドロップ操作を使用するか、XAML またはコードで実行できます。このセクションでは、**C1Reflector** コントロールの基礎について説明します。

リフレクタのコンテンツ

C1Reflector コントロールには、そのコンテンツとしてテキストコンテンツ、Silverlight のコントロール、および DataTemplate を含めることができます。**C1Reflector** コントロールは、コンテンツのリフレクションを投影します。

テキストコンテンツ

短い語句を投影する場合は、単に **Content** プロパティに文字列を設定します。次のようになります。



上の例は単純ですが、プロパティをいくつか設定することによってテキストを飾ることができます。フォントを Silverlight の組み込みフォントに変更するか、または独自のフォントをプロジェクトに追加し、フォントの色を変更し、Blend の[プロパティ]ウィンドウでフォントのサイズをカスタマイズできます。次の図は、36 ピクセルの赤の Curlz MT フォントを使用しています。

テキストコンテンツを追加する方法のタスク別ヘルプについては、「リフレクタへテキストを追加する」を参照してください。

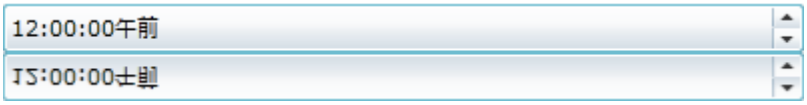
コントロールのコンテンツ

コントロールのコンテンツは、極めて簡単に追加できます。Blend で **C1Reflector** コントロールにコントロールを追加するには、コントロールのアイコンを選択し、ドラッグアンドドロップ操作を使って **C1Reflector** コントロールのコンテナに追加します。XAML を使用する場合は、コントロールを <c1:C1Reflector> タグペアで囲みます。マークアップは次のようになります。

XAML

```
<c1:C1Reflector Width="400">
<c1:C1TimeEditor Width="400"/>
</c1:C1Reflector>
```

上の XAML マークアップの実行結果は、次のようになります。



C1Reflector コントロールでコントロールを使用する方法のタスク別ヘルプについては、「リフレクタへコントロールを追加する」を参照してください。

複数のコントロールの追加

Content プロパティは、一度に1つのコントロールしか受け取ることができません。ただし、パネルベースのコントロールを C1Reflector コントロールの子要素として追加することで、この制限を回避できます。**StackPanel** コントロールなどのパネルベースのコントロールは、複数の要素を保持できます。パネルベースのコントロール自身が複数の要素を保持できるため、これを使用すると、**C1Reflector** コントロールはコントロールを1つだけ保持できるという制限を満たしつつ、コンテンツ領域に複数のコントロールを同時に表示できます。

リフレクションの効果

C1Reflector コントロールには、項目のリフレクションを変更できる **ReflectionOpacityEffect** という効果が付属しています。Coefficient プロパティは、効果による不透明度を計算する関数に適用される係数を設定します。Offset プロパティは、不透明度を計算する関数に適用されるオフセットを設定します。

BlurEffect 効果、**DropShadowEffect** 効果などの Silverlight の標準効果を使用して、リフレクションの外観を変更することもできます。次の表に、それぞれの効果のサンプルを示します。

効果	サンプル画像
ReflectionOpacityEffect	

BlurEffect	
DropShadowEffect	

ReflectionEffects プロパティは、複数の効果のコレクションを受け取ることができます。これにより、一度に複数の効果を使用できます。たとえば、コンテンツが透明ガラスではなくすりガラスに投影されている効果を与える場合は、**BlurEffect** と **ReflectionOpacityEffect** の両方を使って試すことができます。

リフレクション効果を与える場合のタスク別ヘルプについては、「ドロップシャドウ効果を使用する」、「ぼかし効果を使用する」、および「不透明効果を使用する」を参照してください。

自動更新

C1Reflector コントロールは、デフォルトでコントロールのコンテンツが更新されるたびに自動的にリフレクションを更新します。これは、C1Reflector コントロール、**C1Accordion** コントロールなどの対話式パーツと共にコントロールを使用する場合に役立ちます。自動更新により、これらのコントロールのすべての動作がリフレクションに投影されます。たとえば、ユーザーがアコーディオンペインを展開すると、リフレクションのアコーディオンペインも展開されます。

コンテンツを自動的に更新しない場合は、**AutoUpdate** プロパティを **False** に設定します。

メモ:`AutoUpdate` を使用すると、特に複数のリフレクタを使用する場合にアプリケーションの速度が低下することがあります。

面の投影

Reflector for Silverlight は、面の投影をサポートします。これにより、2次元コントロールを3次元の面の上に描画できます。コントロールのすべてのパーツは、2次元の面の上に存在しているかのように動作します。

3次元効果は、X、Y、Z の3つの面に沿ってコントロールを回転することによって作成されます。3つの面それぞれについて、回転、回転の中心、グローバルオフセット、およびローカルオフセットを設定できます。したがって、**C1Reflector** コントロールの投影を変更するためのプロパティは合計で 12 あります。次の表では、各プロパティについて説明します。

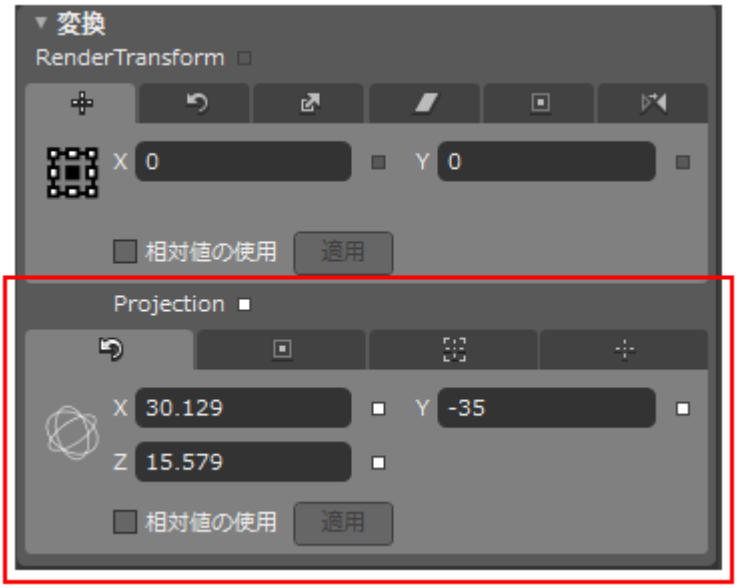
プロパティ	説明
RotationX	X 軸の周囲でオブジェクトを回転する度数を取得または設定します。
RotationY	Y 軸の周囲でオブジェクトを回転する度数を取得または設定します。
RotationZ	Z 軸の周囲でオブジェクトを回転する度数を取得または設定します。
CenterOfRotationX	回転するオブジェクトの中心の X 座標を取得または設定します。

CenterOfRotationY	回転するオブジェクトの中心の Y 座標を取得または設定します。
CenterOfRotationZ	回転するオブジェクトの中心の Z 座標を取得または設定します。
GlobalOffsetX	画面の X 軸方向に沿ってオブジェクトが変換される距離を取得または設定します。
GlobalOffsetY	画面の Y 軸方向に沿ってオブジェクトが変換される距離を取得または設定します。
GlobalOffsetZ	画面の Z 軸方向に沿ってオブジェクトが変換される距離を取得または設定します。
LocalOffsetX	オブジェクトの面の X 軸方向に沿ってオブジェクトが変換される距離を取得または設定します。
LocalOffsetY	オブジェクトの面の Y 軸方向に沿ってオブジェクトが変換される距離を取得または設定します。
LocalOffsetZ	オブジェクトの面の Z 軸方向に沿ってオブジェクトが変換される距離を取得または設定します

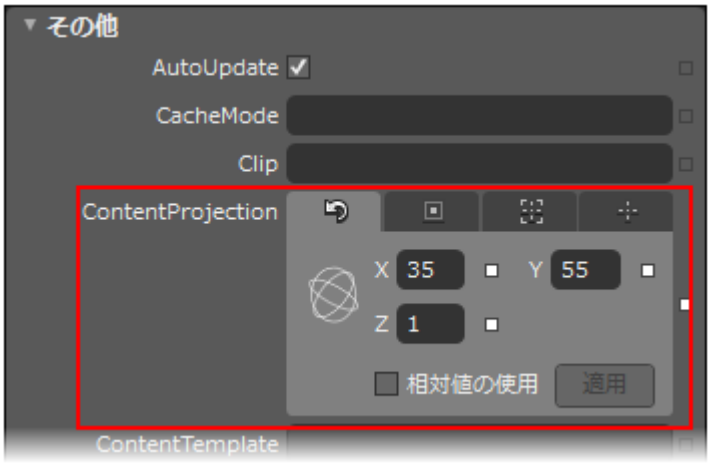
C1Reflector コントロール全体の投影を変更することも、コントロールのコンテンツのみの投影を変更することもできます。結果は、わずかに異なります。コントロール全体の投影を変更すると、リフレクションの両方のコンテンツの投影が変化します。コンテンツの投影を変更すると、コンテンツの投影のみが変化し、リフレクションはその変化をミラーリングします。次の表に、コントロール投影とコンテンツ投影の違いを示します。

投影の設定	コントロール投影	コンテンツ投影
RotationX = 35 RotationY = 55 RotationZ = 1		

コントロール投影とコンテンツ投影のどちらも XAML またはコードで設定できますが、必要な外観が得られるまで Blend で設定を調整する方が簡単です。コントロールの投影プロパティは、**[Projection]**セパレータの下の**[Transform]**セクションにあります。



コントロールの投影プロパティは、**ContentProjection** プロパティの横にある**[その他]**セクションにあります。



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 でのプログラミングに精通しており、**C1Reflector** コントロールの一般的な使用方法を理解していることを前提としています。**Reflector for 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、**Reflector for Silverlight** 製品を使って特定のタスクを行うための方法を提供します。

また、タスク別ヘルプトピックは、新しい 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。

リフレクタへテキストを追加する

Content プロパティに文字列を設定すると、**C1Reflector** コントロールに簡単なテキストを追加できます。テキストを追加すると、テキストとそのリフレクション、および C1Reflector コントロールが表示されます。

Blend での設計時

簡単なテキストをコントロールに追加するには、次の手順に従います。

1. **C1Reflector** コントロールを1回クリックして選択します。
2. [プロパティ]ウィンドウで、Content プロパティを "Hello World!" に設定します。

XAML の場合

コントロールに簡単なテキストを追加するには、Content ="Hello World" を <c1:C1Reflector> タグに追加します。マークアップは次のようになります。

XAML

```
<c1:C1Reflector Content="Hello World!">
```

コードの場合

簡単なテキストをコントロールに追加するには、次の手順に従います。

1. x:Name="C1Reflector1" を <c1:C1Reflector> タグに追加します。これによってコントロールに一意の識別子が与えられ、それを使用することによってコードからコントロールを呼び出すことができます。
2. コードビューに切り替えて、InitializeComponent() メソッドの下に次のコードを追加します。

VisualBasic

```
C1Reflector1.Content = "Hello World!"
```

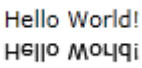
C#

```
C1Reflector1.Content = "Hello World!";
```

3. プログラムを実行します

このトピックの作業結果

このトピックの結果は、次の画像のようになります。



リフレクタへコントロールを追加する

C1Reflector コントロールは、Silverlight UI コントロールを受け取ることができます。この機能の例として、このトピックでは、リフレクタに標準の Silverlight ボタンを追加します。コントロールにボタンを追加すると、**C1Reflector** コントロールにコントロールとそのリフレクションが表示されます。

Blend での設計時

コントロールを追加するには、次の手順に従います。

1. **[オブジェクトとタイムライン]タブ**で**[すべて]**を選択して、使用できるすべてのコントロールを表示します。
2. コントロールのリストで**[Button]**アイコンを選択し、ドラッグアンドドロップ操作を使って C1Reflector コンテナに追加します。

XAML の場合

コントロールを追加するには、<c1:C1Reflector> タグと </c1:C1Reflector> タグの間に次のマークアップを配置します。

XAML

```
<Button Content="ボタン"></Button>
```

コードの場合

コントロールを追加するには、次の手順に従います。

1. x:Name="C1Reflector1" を <c1:C1Reflector> タグに追加します。これによってコントロールに一意の識別子が与えられ、それを使用することによってコードからコントロールを呼び出すことができます。
2. コードビューに切り替えて、InitializeComponent() メソッドの下に次のコードを追加します。

VisualBasic

```
'ボタンオブジェクトを作成して名前を付ける
```

```
Dim Button1 As New Button()
Button1.Content = "ボタン"
'ボタンに C1Reflector コントロールのコンテンツを設定する
C1Reflector1.Content = Button1
```

C#

```
//ボタンオブジェクトを作成して名前を付ける
Button Button1 = new Button();
Button1.Content = "ボタン";
//ボタンに C1Reflector コントロールのコンテンツを設定する
C1Reflector1.Content = Button1;
```

3. プログラムを実行します

このトピックの作業結果

このトピックの結果は、次の画像のようになります。

ドロップシャドウ効果を使用する

Silverlight の標準ドロップシャドウ効果を使用すると、リフレクションにドロップシャドウを追加できます。このトピックでは、Blend、XAML、およびコードで、ドロップシャドウ効果を追加します。

Blend での設計時

ドロップシャドウ効果を使用するには、次の手順に従います。

1. Blend プロジェクトに **C1Reflector** コントロールを追加します。
2. **C1Reflector** コントロールを1回クリックして選択します。
3. [プロパティ]ウィンドウで ReflectionEffects の省略符ボタンをクリックし、[Effect コレクションエディター: ReflectionEffects]ダイアログボックスを開きます。
4. **[別のアイテムを追加]**をクリックします。

   [オブジェクトの選択]ダイアログボックスが開きます。
5. リストから DropShadowEffect を選択し、[OK]をクリックすると、効果がコントロールに追加され、[Effect コレクションエディター: ReflectionEffects]ダイアログボックスに戻ります。
6. [プロパティ]グリッドで、次のプロパティを設定します。
   - **BlurRadius** プロパティを "7" に設定します。
   - **Direction** プロパティを "180" に設定します。
   - **Opacity** プロパティを "95%" に設定します。
   - **ShadowDepth** を「8」に設定します。

7. [OK]をクリックして、[Effect コレクションエディター: ReflectionEffects]ダイアログボックスを閉じます。

Blend での設計時

ドロップシャドウ効果を使用するには、次の手順に従います。

1. Blend プロジェクトに C1Reflector コントロールを追加します。
2. <c1:C1Reflector> タグに Content="C1Reflector" を追加し、文字列コンテンツを設定します。
3. <c1:C1Reflector> タグと </c1:C1Reflector> タグの間に次の XAML を配置し、ドロップシャドウ効果を追加してそのプロパティを設定します。

XAML

```
<c1:C1Reflector.ReflectionEffects>
<DropShadowEffect BlurRadius="7" Opacity="0.95" ShadowDepth="8"
Direction="180"/>
      </c1:C1Reflector.ReflectionEffects>
```

コードの場合

ドロップシャドウ効果を使用するには、次の手順に従います。ドロップシャドウ効果を使用するには、次の手順に従います。

1. Blend プロジェクトに C1Reflector コントロールを追加します。
2. <c1:C1Reflector> タグに Content="C1Reflector" を追加し、文字列コンテンツを設定します。
3. x:Name="C1Reflector1" を <c1:C1Reflector> タグに追加します。これによってコントロールに一意の識別子が与えられ、それを使用することによってコードからコントロールを呼び出すことができます。
4. コードビューに切り替えて、次の名前空間をインポートします。

VisualBasic

```
Imports System.Windows.Media.Effects
```

C#

```
using System.Windows.Media.Effects;
```

5. InitializeComponent() メソッドの下に次のコードを追加します。

VisualBasic

```
'DropShadowEffect オブジェクトを作成してプロパティを設定します。
Dim newDropShadowEffect As New DropShadowEffect() newDropShadowEffect.BlurRadius
= 7
newDropShadowEffect.Direction = 180
newDropShadowEffect.Opacity = 95
newDropShadowEffect.ShadowDepth = 8
'C1Reflector コントロールに DropShadowEffect オブジェクトを追加します。
```

```
C1Reflector1.ReflectionEffects.Add(newDropShadowEffect)
```

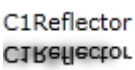
C#

```
//DropShadowEffect オブジェクトを作成してプロパティを設定します。
DropShadowEffect newDropShadowEffect = new DropShadowEffect();
newDropShadowEffect.BlurRadius = 7;
newDropShadowEffect.Direction = 180;
newDropShadowEffect.Opacity = 95;
newDropShadowEffect.ShadowDepth = 8;
//C1Reflector コントロールに DropShadowEffect オブジェクトを追加します。
C1Reflector1.ReflectionEffects.Add(newDropShadowEffect);
```

6. プログラムを実行します

このトピックの作業結果

プロジェクトを実行すると、**C1Reflector** コントロールおよびそのコンテンツは次の画像のようになります。



ぼかし効果を使用する

Silverlight の標準ドロップシャドウ効果を使用すると、リフレクションにぼかし効果を追加できます。このトピックでは、Blend、XAML、およびコードで、ドロップシャドウ効果を追加します。

Blend での設計時

ぼかし効果を使用するには、次の手順に従います。

1. Blend プロジェクトに **C1Reflector** コントロールを追加します。
2. **C1Reflector** コントロールを1回クリックして選択します。
3. [プロパティ]ウィンドウで ReflectionEffects の省略符ボタンをクリックし、[Effect コレクションエディター: ReflectionEffects]ダイアログボックスを開きます。
4. [別のアイテムを追加]をクリックします。
   [オブジェクトの選択]ダイアログボックスが開きます。
5. リストから BlurEffect を選択し、[OK]をクリックすると、効果がコントロールに追加され、[Effect コレクションエディター: ReflectionEffects]ダイアログボックスに戻ります。
6. [プロパティ]グリッドで、Radius プロパティを「3」に設定します。
7. [OK]をクリックして、[Effect コレクションエディター: ReflectionEffects]ダイアログボックスを閉じます。

XAML の場合

ぼかし効果を使用するには、次の手順に従います。

1. Blend プロジェクトに C1Reflector コントロールを追加します。
2. <c1:C1Reflector> タグに Content="C1Reflector" を追加し、文字列コンテンツを設定します。

3. <c1:C1Reflector> タグと </c1:C1Reflector> タグの間に次の XAML を配置し、ぼかし効果を追加してそのプロパティを設定します。

XAML

```
<c1:C1Reflector.ReflectionEffects>
<BlurEffect Radius="3"/>
</c1:C1Reflector.ReflectionEffects>
```

コードの場合

ぼかし効果を使用するには、次の手順に従います。

1. Blend プロジェクトに C1Reflector コントロールを追加します。
2. <c1:C1Reflector> タグに Content="C1Reflector" を追加し、文字列コンテンツを設定します。
3. x:Name="C1Reflector1" を <c1:C1Reflector> タグに追加します。これによってコントロールに一意の識別子が与えられ、それを使用することによってコードからコントロールを呼び出すことができます。
4. コードビューに切り替えて、次の名前空間をインポートします。

VisualBasic

```
Imports System.Windows.Media.Effects
```

C#

```
using System.Windows.Media.Effects;
```

5. InitializeComponent() メソッドの下に次のコードを追加します。

VisualBasic

```
'BlurEffect オブジェクトを作成し、Radius プロパティを設定します。
Dim newBlurEffect As New BlurEffect() newBlurEffect.Radius = 3
'C1Reflector コントロールに BlurEffect オブジェクトを追加する
C1Reflector1.ReflectionEffects.Add(newBlurEffect);
```

C#

```
//BlurEffect オブジェクトを作成し、Radius プロパティを設定します。
BlurEffect newBlurEffect = new BlurEffect();
newBlurEffect.Radius = 3;
//C1Reflector コントロールに BlurEffect オブジェクトを追加します。
C1Reflector1.ReflectionEffects.Add(newBlurEffect);
```

6. プログラムを実行します

このトピックの作業結果

プロジェクトを実行すると、**C1Reflector** コントロールおよびそのコンテンツは次の画像のようになります。

不透明効果を使用する

Silverlight の標準ドロップシャドウ効果を使用すると、リフレクションに不透明効果を追加できます。このトピックでは、Blend、XAML、およびコードで、ドロップシャドウ効果を追加します

Blend での設計時

不透明効果を使用するには、次の手順に従います。

1. Blend プロジェクトに **C1Reflector** コントロールを追加します。
2. **C1Reflector** コントロールを1回クリックして選択します。
3. **[プロパティ]**ウィンドウで **ReflectionEffects** の省略符ボタンをクリックし、[Effect コレクションエディター: ReflectionEffects]ダイアログボックスを開きます。
4. **[別のアイテムを追加]**をクリックします。
   [オブジェクトの選択]ダイアログボックスが開きます。
5. リストから ReflectionOpacityEffect を選択し、[OK]をクリックすると、効果がコントロールに追加され、[Effect コレクションエディター: ReflectionEffects]ダイアログボックスに戻ります。
6. [プロパティ]グリッドで、次のプロパティを設定します。
   - **Coefficient** プロパティを「1」に設定します。
   - **Offset** プロパティを「0.5」に設定します。
7. [OK]をクリックして、[Effect コレクションエディター: ReflectionEffects]ダイアログボックスを閉じます。

Blend での設計時

不透明効果を使用するには、次の手順に従います。

1. Blend プロジェクトに C1Reflector コントロールを追加します。
2. <c1:C1Reflector> タグに Content="C1Reflector" を追加し、文字列コンテンツを設定します。
3. <c1:C1Reflector> タグと </c1:C1Reflector> タグの間に次の XAML を配置し、不透明効果を追加してそのプロパティを設定します。

XAML

```
<c1:C1Reflector.ReflectionEffects>
<c1:ReflectionOpacityEffect Coefficient="1" Offset="0.5"/>
</c1:C1Reflector.ReflectionEffects>
```

コードの場合

不透明効果を使用するには、次の手順に従います。

1. Blend プロジェクトに C1Reflector コントロールを追加します。

2. <c1:C1Reflector> タグに Content="C1Reflector" を追加し、文字列コンテンツを設定します。
3. x:Name="C1Reflector1" を <c1:C1Reflector> タグに追加します。これによってコントロールに一意の識別子が与えられ、それを使用することによってコードからコントロールを呼び出すことができます。
4. コードビューに切り替えて、次の名前空間をインポートします。

VisualBasic

```
Imports C1.Silverlight.Extended
```

C#

```
using C1.Silverlight.Extended;
```

5. InitializeComponent() メソッドの下に次のコードを追加します。

VisualBasic

```
'ReflectionOpacityEffect オブジェクトを作成してプロパティを設定します。
Dim newOpacity As New ReflectionOpacityEffect()
newOpacity.Coefficient = 1
newOpacity.Offset = 0.5
'C1Reflector1 コントロールに ReflectionOpacityEffect オブジェクトを追加します。
C1Reflector1.ReflectionEffects.Add(newOpacity)
```

C#

```
//ReflectionOpacityEffect オブジェクトを作成してプロパティを設定します。
ReflectionOpacityEffect newOpacity = new ReflectionOpacityEffect();
newOpacity.Coefficient = 1;
newOpacity.Offset = 0.5;
//C1Reflector1 コントロールに ReflectionOpacityEffect オブジェクトを追加します。
C1Reflector1.ReflectionEffects.Add(newOpacity);
```

6. プログラムを実行します

このトピックの作業結果

プロジェクトを実行すると、**C1Reflector** コントロールおよびそのコンテンツは次の画像のようになります。